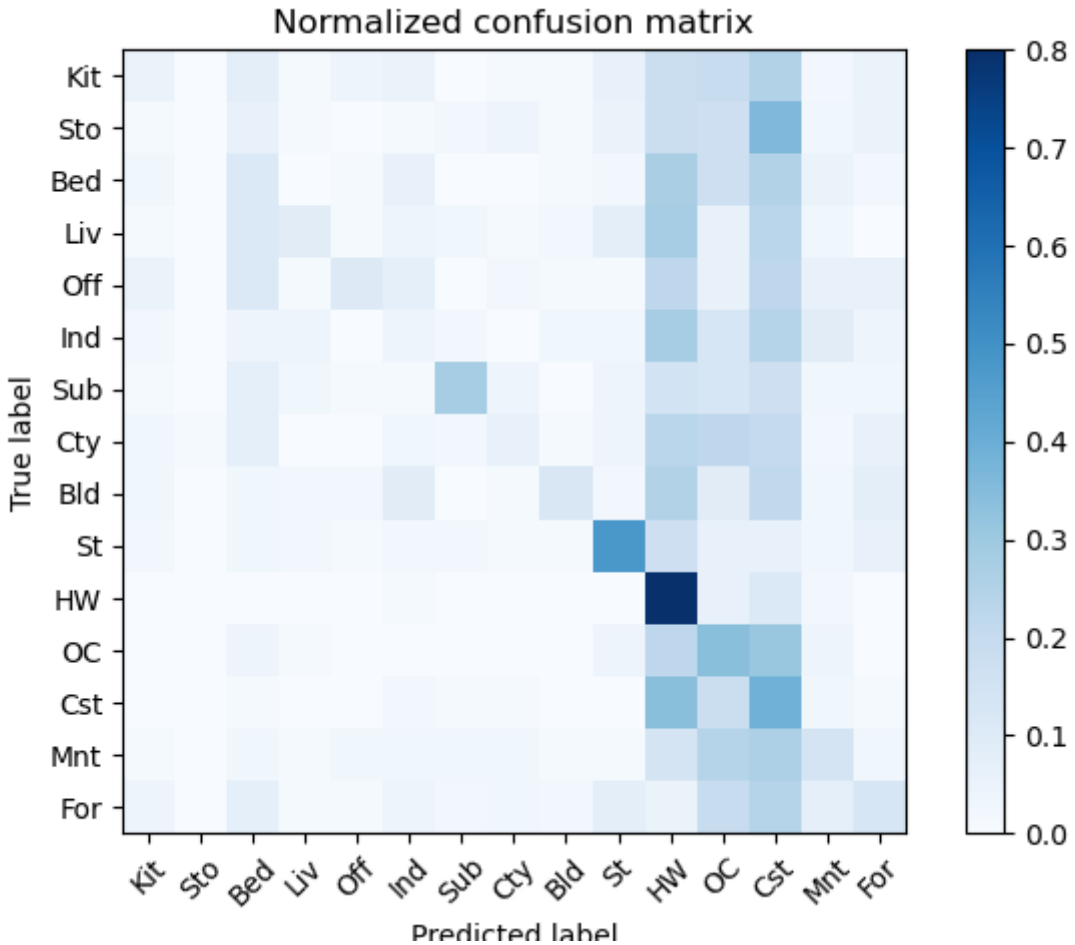
Computer Vision, Spring 2023 HW2

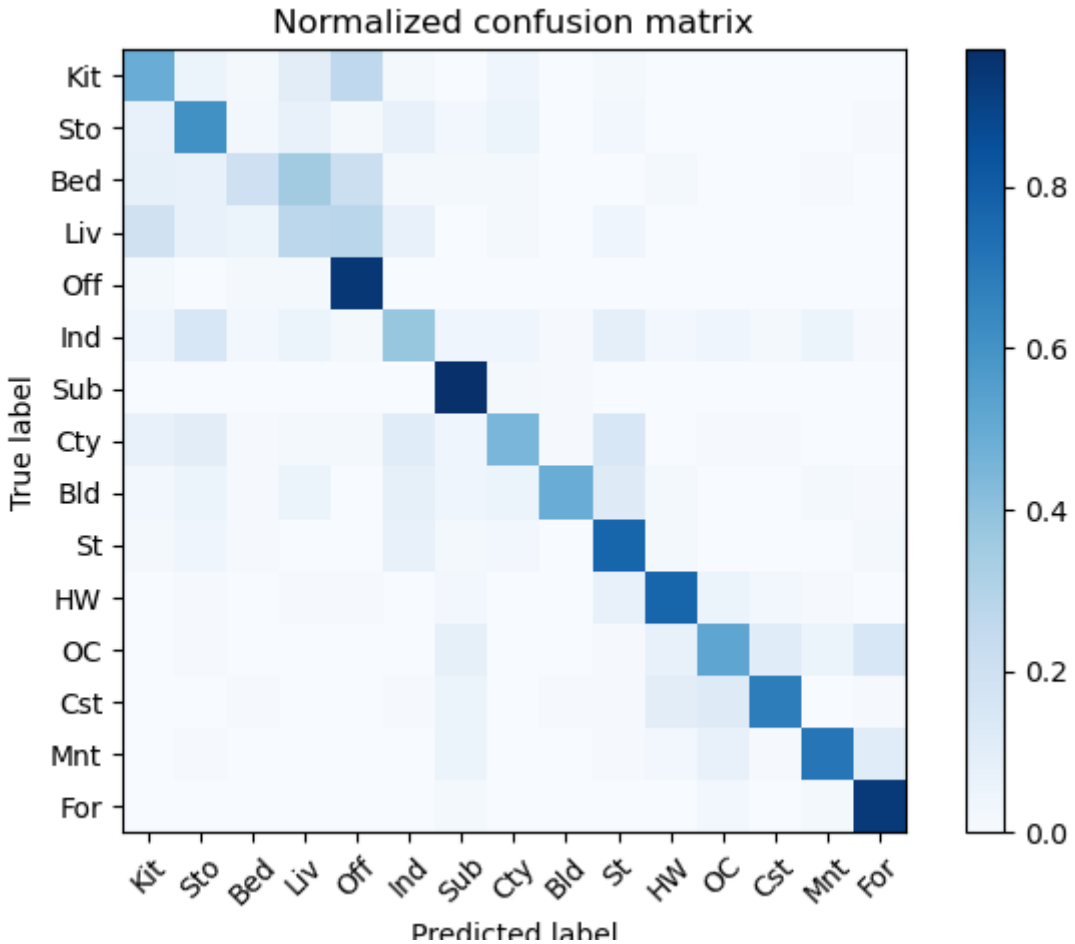
B11705009 An-Che, Liang

Part 1: Bag-of-Words Scene Recognition

Feature representation: Tiny image



Feature representation: Bag of SIFT



Discussion:

The steps of Tiny image Scene Recognition are:

1. Resize the image to small size.
2. Use K-Nearest Neighbor algorithm on pixel values to get images with high similarity of the test image, then use those images's label to predict the test image's label.

The steps of Bag-of-Words Scene Recognition are:

1. Extract features from the orignal image.
2. Use K-means clustering to partition our features into K (my vocab size is 1000) clusters in such a way that the sum of the distances between the objects and their assigned cluster center is minimized.
3. Build BoW histogram for evey images with regard of different cluster centers.
4. Use K-Nearest Neighbor algorithm on BoW histogram to get images with high similarity of the test image, then use those images's label to predict the test image's label.

The accuracy of the Tiny image algorithm is quite low ($\mathrm{acc} = 0.209$). I think the reason is that the classification is based on raw pixel values, but two picture in the same category could potencially have drastically different values, and two picture in different categories could have similar values as well.

The accuracy of the Bag of SIFT algorithm is quite high ($\mathrm{acc} = 0.612$). I think that's because the SIFT feature can better describe the image than the minified version of the original image (the Tiny image approach), thus yield better performance.

Part 2 : CNN Image Classification

Performance

The best performance of `mynet` is $\text{acc} = 0.612$, the best performance of `resnet18` is $\text{acc} = 0.907$.

Model architectures

In [ ]:
```
from p2.model import MyNet , ResNet18
def checkModelArchitectures(model_type):
    print(f"Model type={model_type}")
    if model_type == "mynet":
        model = MyNet()
    elif model_type == "resnet18":
        model = ResNet18()
    print("Model architecture:")
    print(model)
    print("Number of parameters:")
    model_total_params = sum(p.numel() for p in model.parameters() if p.requires_grad
    print(model_total_params)

for model_name in ["mynet" , "resnet18"]:
    checkModelArchitectures(model_name)
```

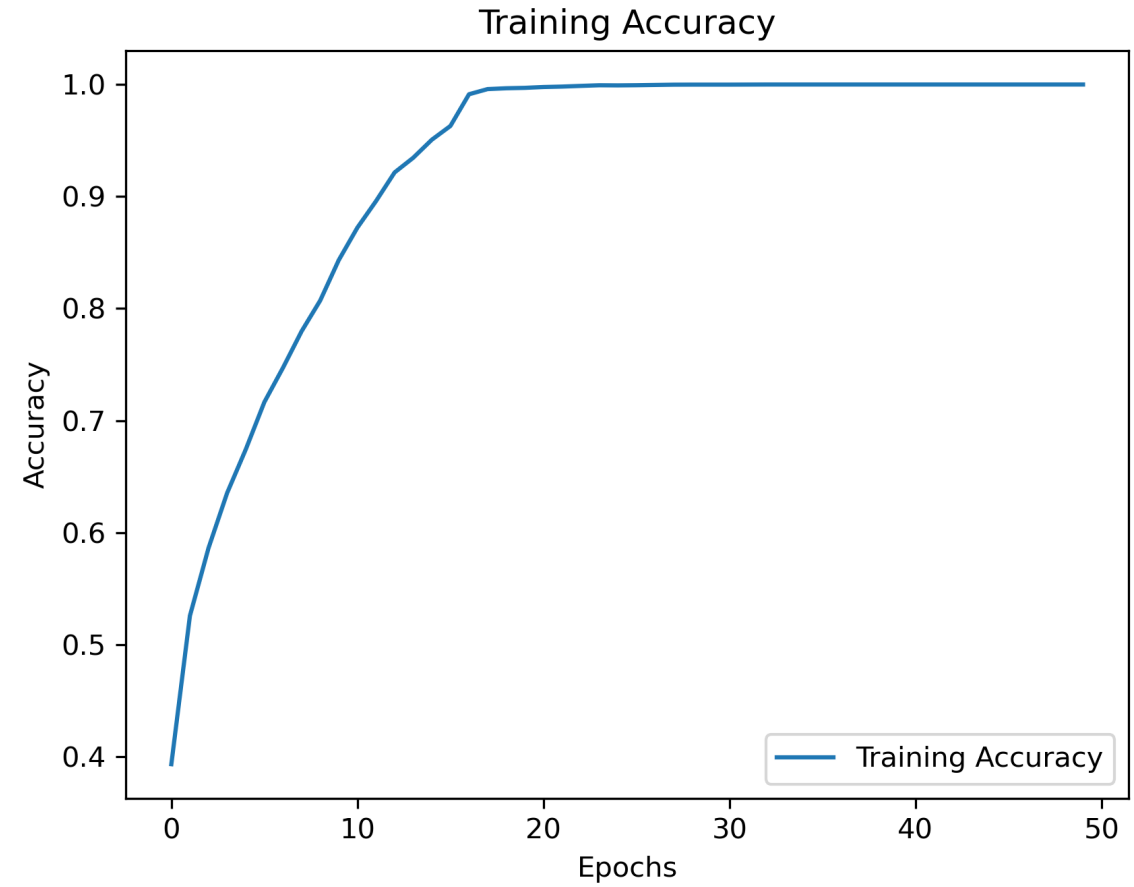
```
Model type=mynet
Model architecture:
MyNet(
  (model): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Flatten(start_dim=1, end_dim=-1)
    (10): Linear(in_features=1024, out_features=512, bias=True)
    (11): ReLU()
    (12): Linear(in_features=512, out_features=10, bias=True)
  )
)
Number of parameters:
586250
Model type=resnet18
Model architecture:
ResNet18(
  (resnet): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(3, 3), bias=Fa
lse)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
    (relu): ReLU(inplace=True)
    (maxpool): Identity()
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), b
ias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
```

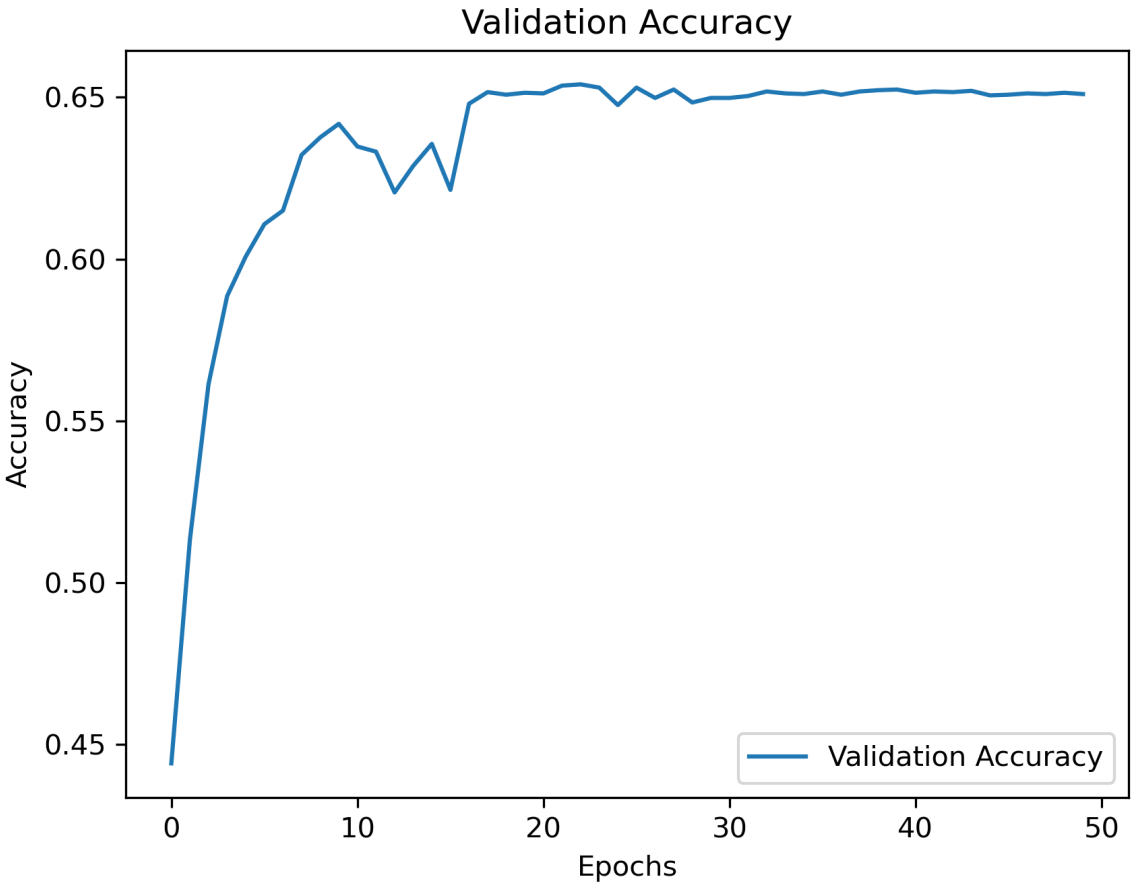
```
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
```

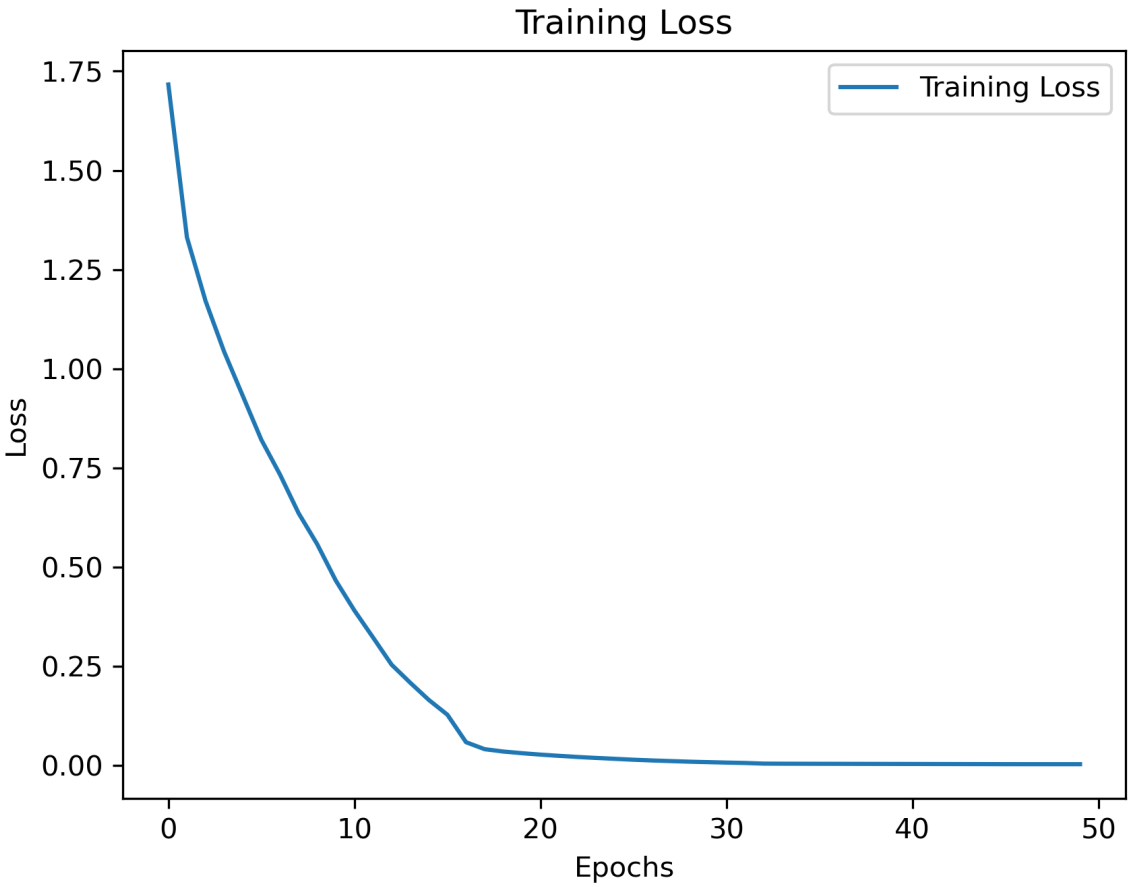
```
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=10, bias=True)
  )
)
Number of parameters:
11181642
```

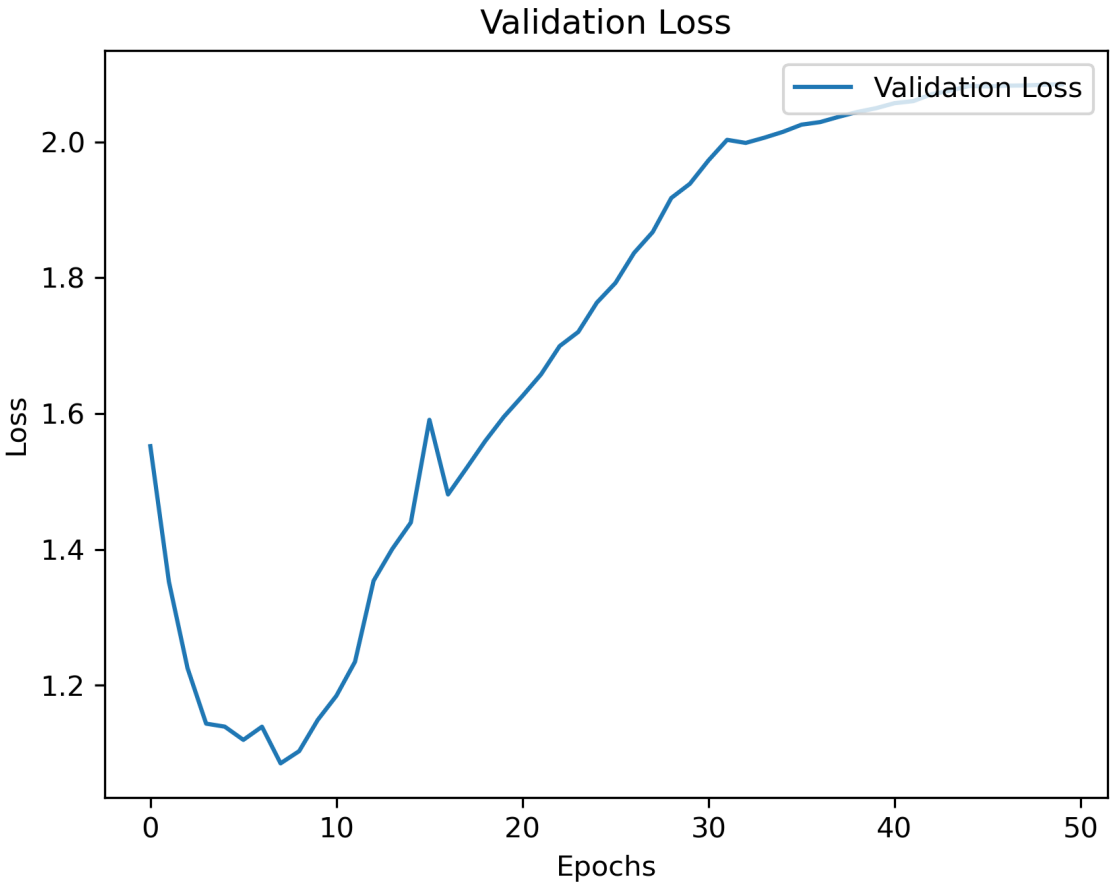
Plots

Mynet

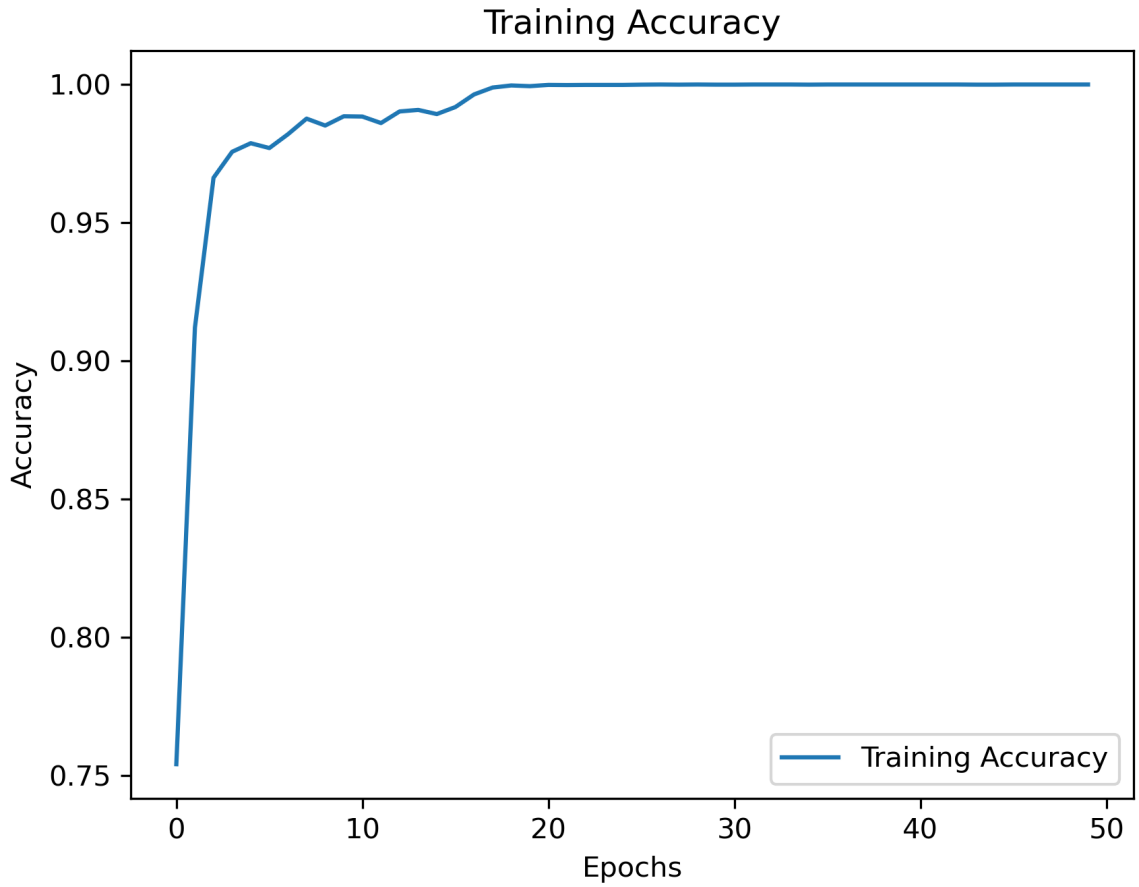

Training Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Accuracy
0
10
20
30
40
50
Epochs
Training Accuracy

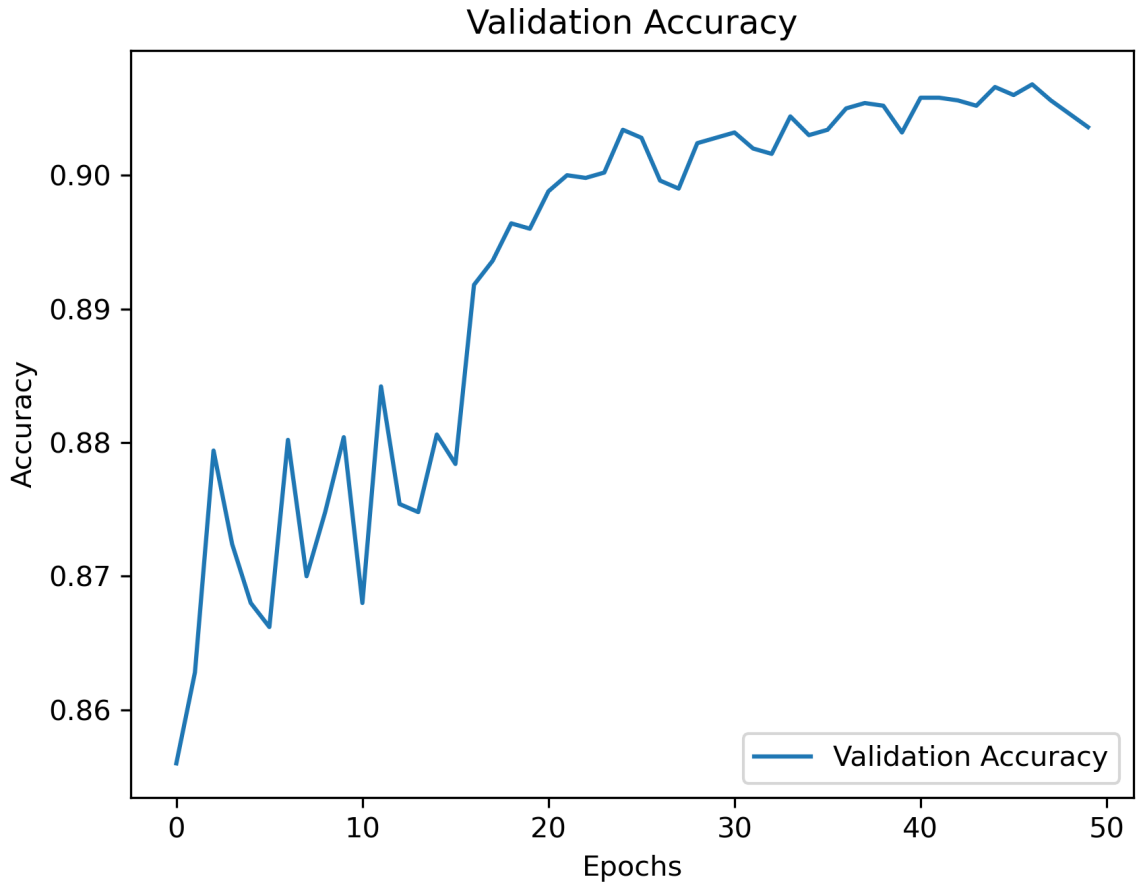

Validation Accuracy
0.65
0.60
0.55
0.50
0.45
Accuracy
0
10
20
30
40
50
Epochs
Validation Accuracy

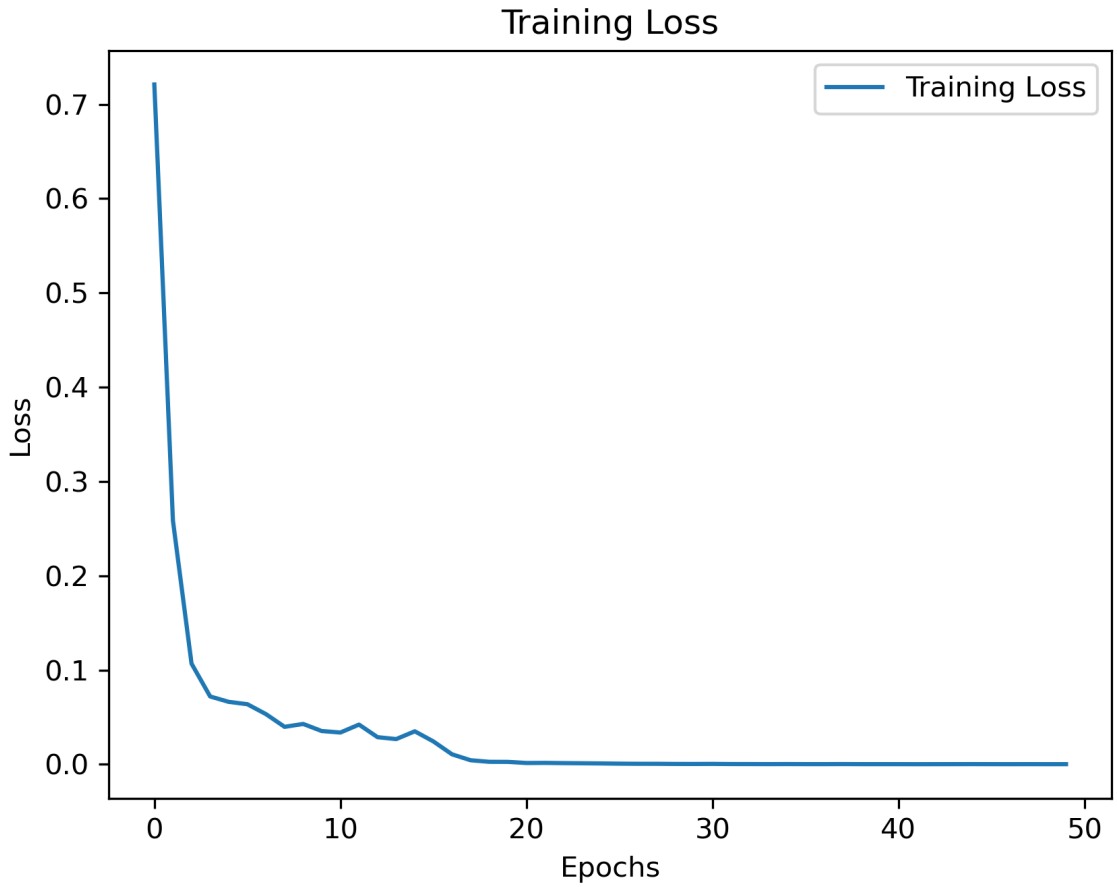

Training Loss
Training Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Loss
0
10
20
30
40
50
Epochs

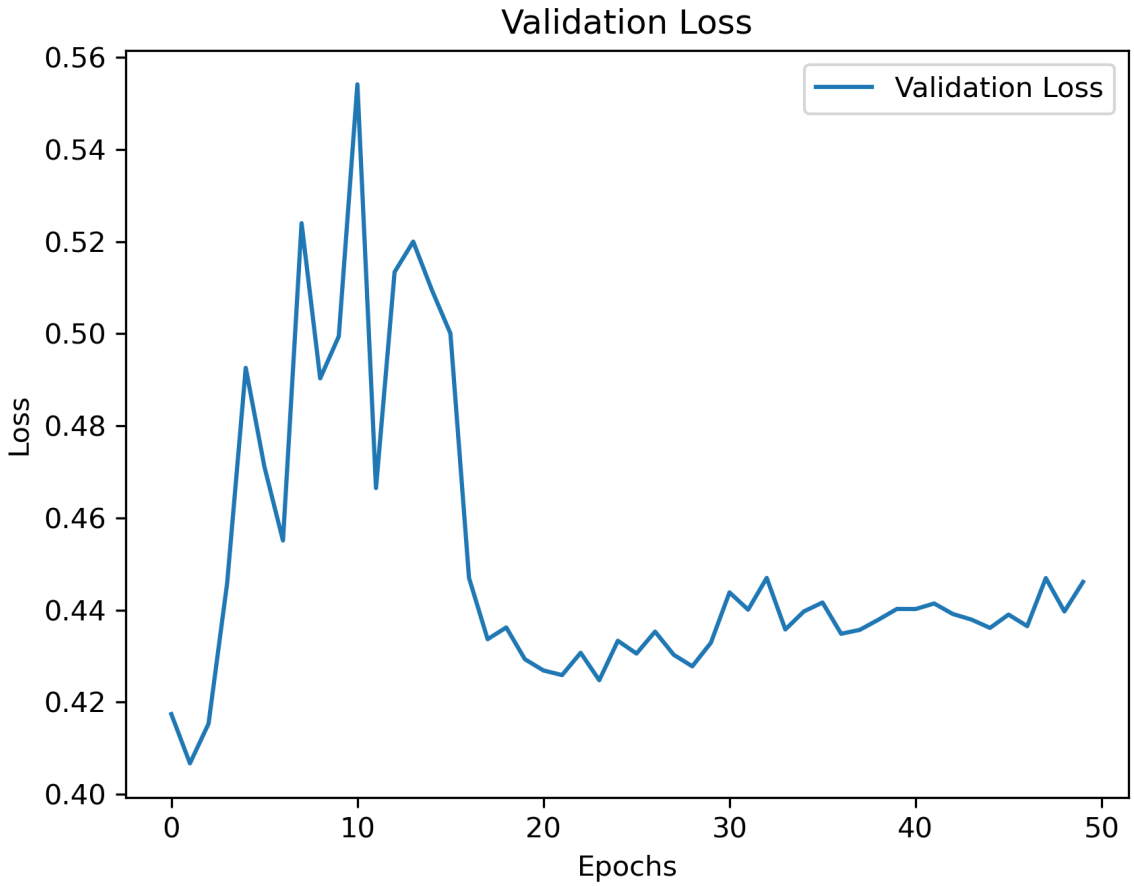

Validation Loss
Validation Loss
2.0
1.8
1.6
1.4
1.2
Loss
0
10
20
30
40
50
Epochs

Resnet 18


Training Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
Accuracy
0
10
20
30
40
50
Epochs
Training Accuracy


Validation Accuracy
0.90
0.89
0.88
0.87
0.86
Accuracy
0
10
20
30
40
50
Epochs
Validation Accuracy


Training Loss
Training Loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Validation Loss
Validation Loss
Loss
Epochs
0.40
0.42
0.44
0.46
0.48
0.50
0.52
0.54
0.56
0
10
20
30
40
50

Discussion

On my best performing model, I used the `resnet18` model with pretrained weight and `Adam` optimizer, with that setup alone breaks the simple bassline. Then I conduct data augmentation with `transforms.RandAugment()`, I also tried to implement data augmentation by combining `transforms.RandomHorizontalFlip()`, `transforms.RandomVerticalFlip()`, `transforms.RandomRotation()`, `transforms.ColorJitter()` but yield poor performance. This setup breaks the medium bassline. Finally I tweak the architecture of `resnet18` by reducing the kernal size of the fist convolution layer from $(7, 7)$ to $(3, 3)$, reduce the stride from 2 to 1. I also remove the first max pool layer. This setup breaks the strong bassline.