**Computer Vision HW2 Report**

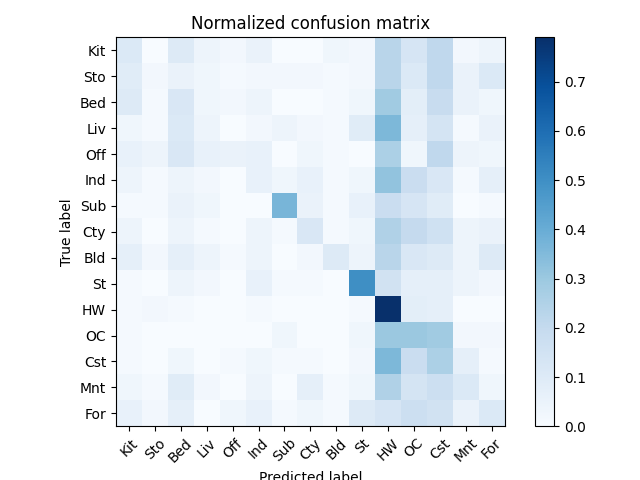
Student ID: B10705034

Name: 許文鑫

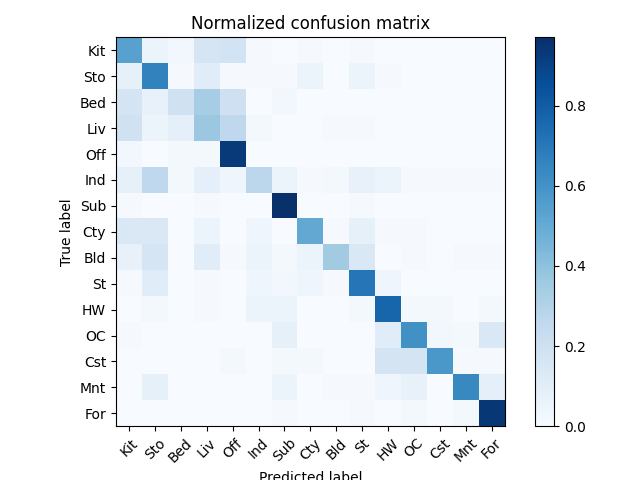
**Part 1. (10%)**

**• Plot confusion matrix of two settings. (i.e. Bag of sift and tiny image) (5%)**

**Tiny image:**

****

**Bag of sift:**

****

**• Compare the results/accuracy of both settings and explain the result. (5%)**

Tiny image accuracy: 0.204, Bag of sift accuracy: 0.6026667.

The accuracy of tiny image is very low. I think the reason is that the method compares pictures directly without extracting any features. So even two pictures are in the same class, their features might show in different position, and the classifier can’t classify them properly.

Unlike the tiny image method, bag of sift method has a quite high accuracy, and I think the reason is that it extracts features from a part of the picture. So even if two picture in the same class have features at different position, these features can be detected and used to classify these two pictures.

**Part 2. (25%)**

**• Report accuracy of both models on the validation set. (2%)**

Mynet: 0.721

Resnet: 0.9063

**• Print the network architecture & number of parameters of both models. What is the main difference between ResNet and other CNN architectures? (5%)**

**Mynet:**

Number of parameters: 586250

MyNet(

(cnn): 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)

)

)

**Resnet:**

Number of parameters: 11173962

ResNet18(

(resnet): ResNet(

(conv1): Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(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), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer2): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=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\_stats=True)

)

)

(avgpool): AdaptiveAvgPool2d(output\_size=(1, 1))

(fc): Linear(in\_features=512, out\_features=10, bias=True)

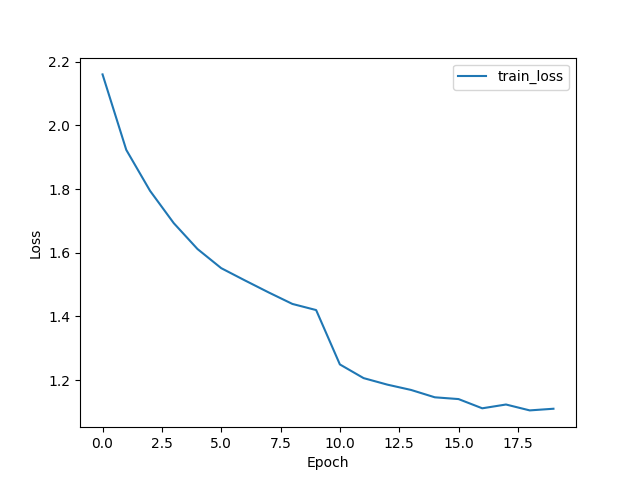
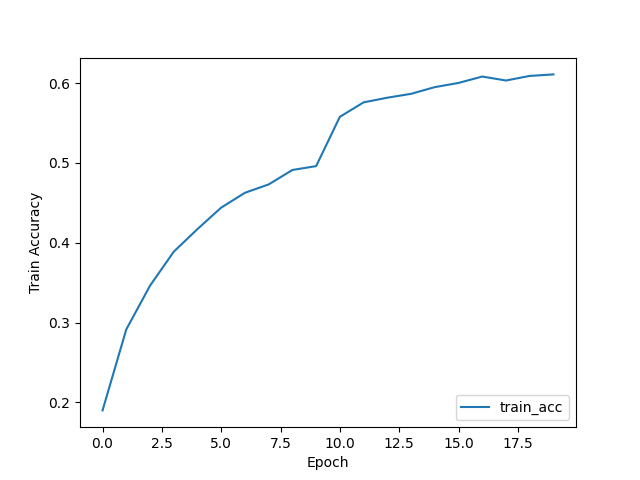
)

)

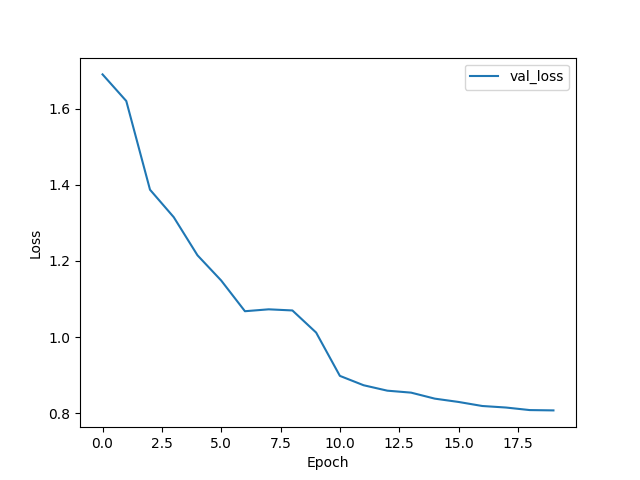
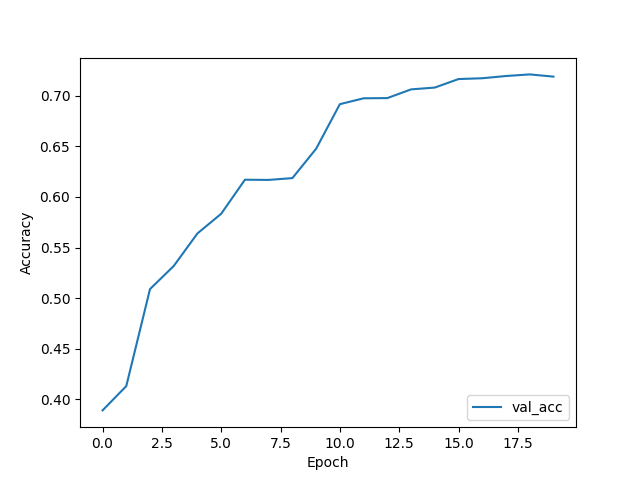
**• Plot four learning curves (loss & accuracy) of the training process (train/validation) for both models. Total 8 plots. (8%)**

**Mynet:**

* **Train**

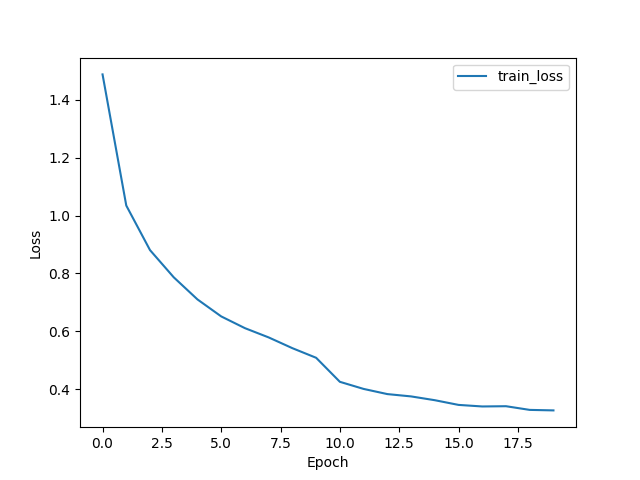
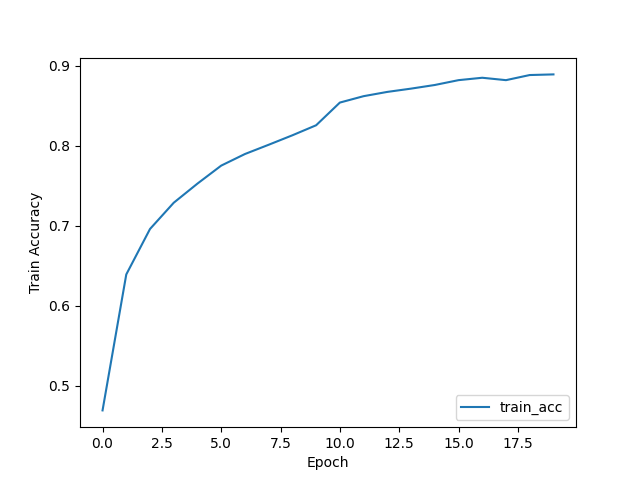
****

* **Validation**

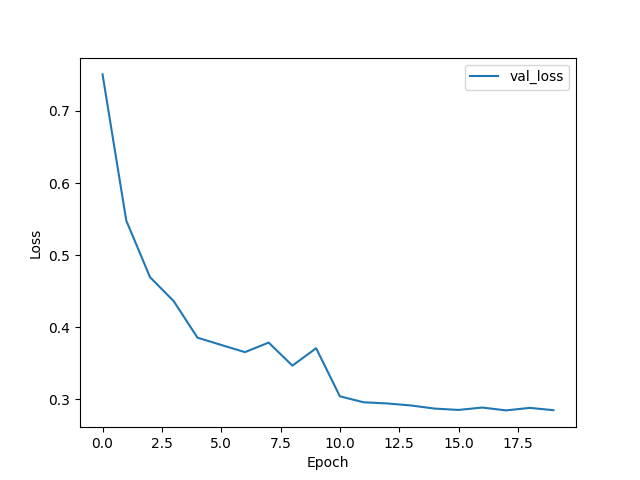
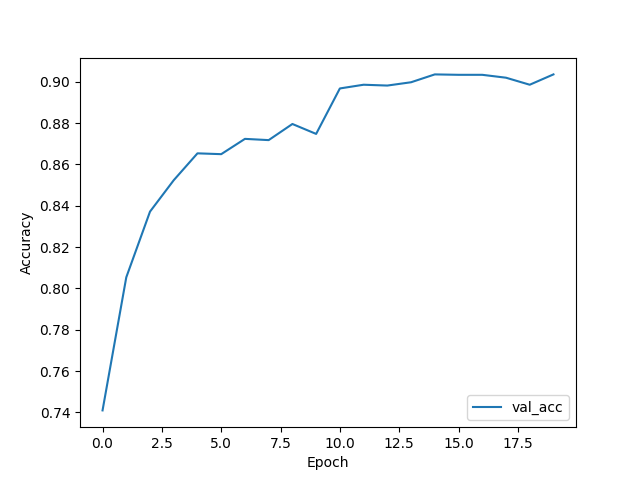
****

**Resnet18:**

* **Train**

****

* **Validation**

****

**• Briefly describe what method do you apply on your best model? (e.g. data augmentation, model architecture, loss function, etc) (10%)**

My best model is resnet18 with pretrained weight, conv1’s kernel size = 3, replacing first maxpool layer with Identity(), data augmentation, and use adam.

For data augmentation, I use a function RandAugment(4,15) in torchvision.transforms. This function can randomly apply 4 augmentations with magnitude = 15. And model with these setups has a quite high accuracy.