Computer Vision HW2 Report

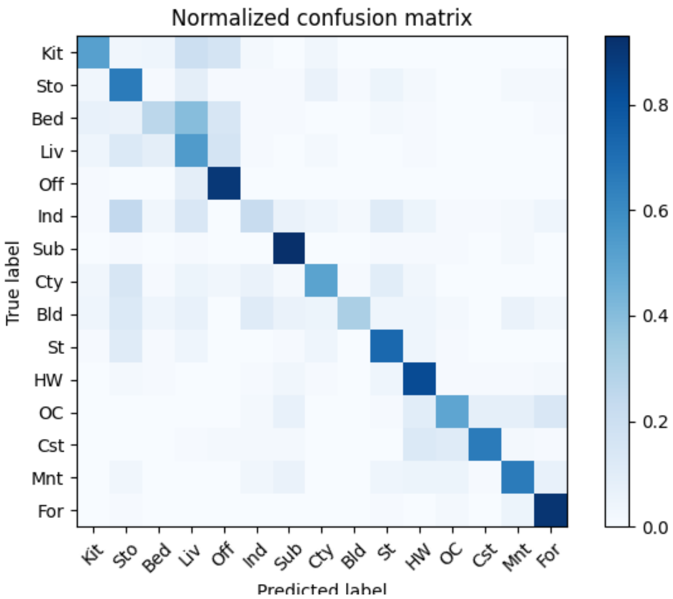
Student ID: B07303024
Name: 李品樺

Part 1. (10%)

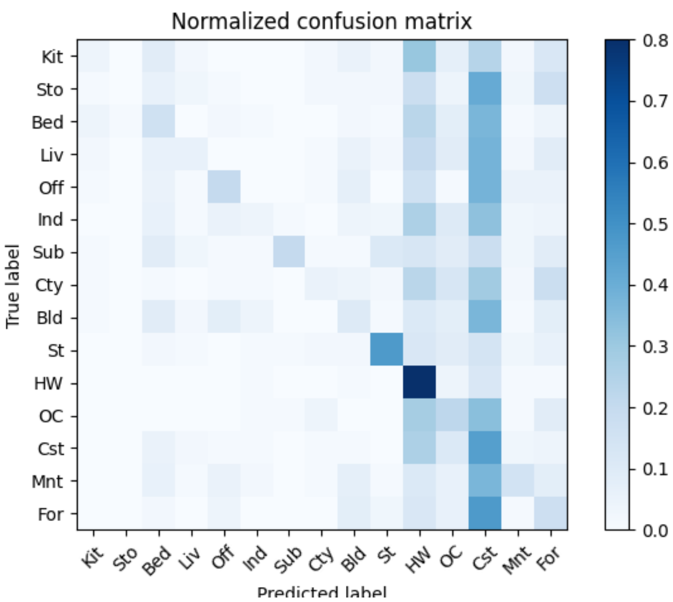
• Plot confusion matrix of two settings. (i.e. Bag of sift and tiny image representation) (5%)

Ans:

- Bag of sift



- Tiny image representation



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

- Performance

Type	Bag of sift	Tiny image
Accuracy	60.93%	20.73%

- Experiment setting
  For tiny image, we resize the image to 8*8. When the tiny image size is smaller, the performance is usually better. For Bag of sift, there are several hyperparameters to choose from in dsift. We set the function as below in get_bags_of_sift.py.

```
dsift(img, step = [3, 3], window_size=4, fast=True)
```

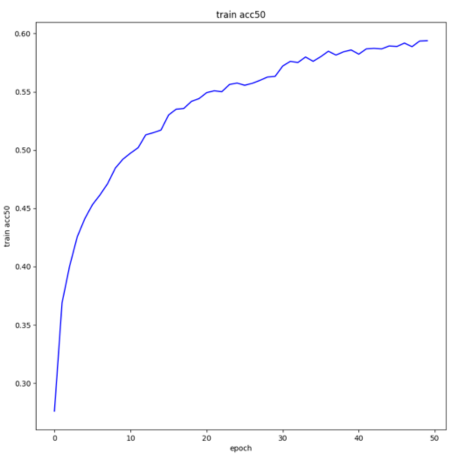
  In the KNN classifier, we choose k as 5, and use cdist from scipy.spatial.distance with metric ‘cityblock’. Then, we utilize the function, np.argsort(), to sort the distancesfrom smallest to largest. Finally, we can compute the mode to get the predicted result. From the confusion matrix of tiny image representation above, we can find out that several columns of the class are darker. The tiny image feature tends to guess the image these classes.

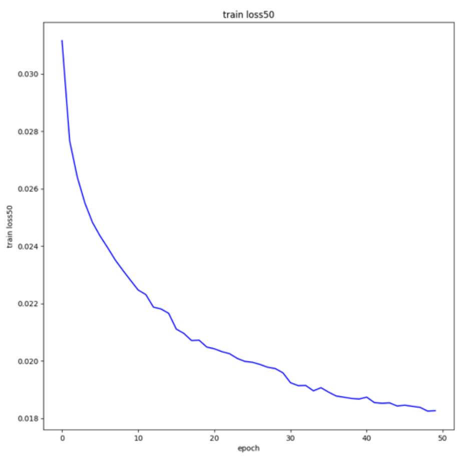
Part 2. (35%)

• Compare the performance on residual networks and LeNet. Plot the learning curve (loss and accuracy) on both training and validation sets for both 2 schemes. 8 plots in total. (20%)
Ans:

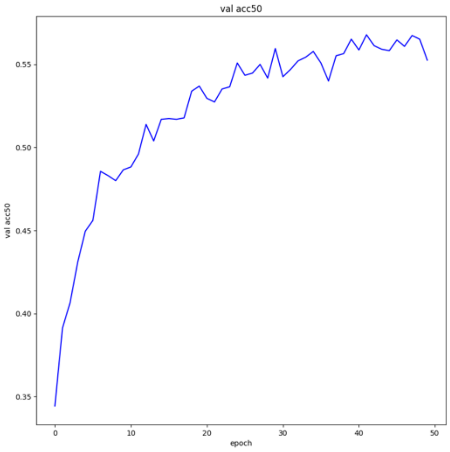
- The performance of MyResnet is clearly better than LeNet as the images show. The residual net can reach about 79% on validation set but LeNet only reaches 57%.
- **LeNet**

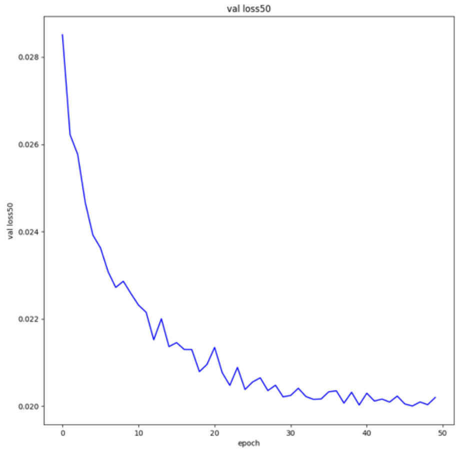
  - **Train**





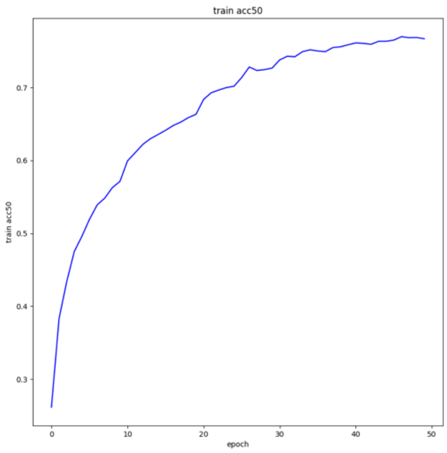
■ Validation

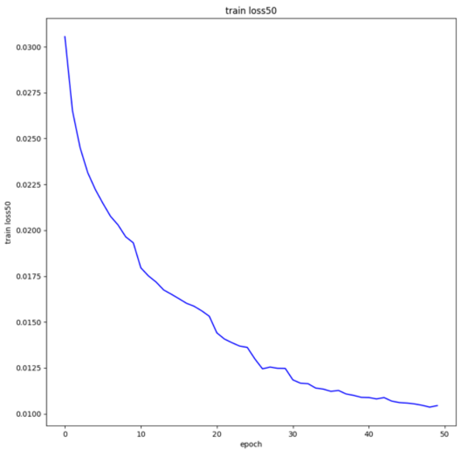




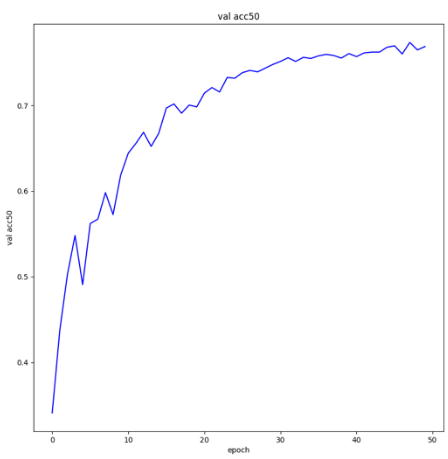
● MyResnet

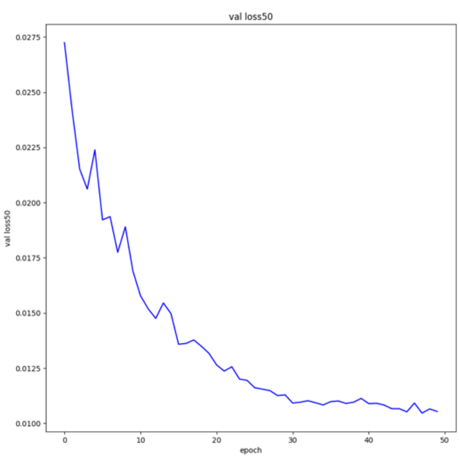
■ Train





■ Validation





• Attach basic information of the model you use including model architecture and number of the parameters. (5%)

Ans:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
            Conv2d-2           [-1, 64, 32, 32]          36,928
       BatchNorm2d-3           [-1, 64, 32, 32]             128
              ReLU-4           [-1, 64, 32, 32]               0
            Conv2d-5           [-1, 64, 32, 32]           4,160
              ReLU-6           [-1, 64, 32, 32]               0
    residual_block-7           [-1, 64, 32, 32]               0
            Conv2d-8          [-1, 128, 32, 32]          73,856
         MaxPool2d-9          [-1, 128, 16, 16]               0
      BatchNorm2d-10          [-1, 128, 16, 16]             256
             ReLU-11          [-1, 128, 16, 16]               0
           Conv2d-12          [-1, 128, 16, 16]         147,584
      BatchNorm2d-13          [-1, 128, 16, 16]             256
             ReLU-14          [-1, 128, 16, 16]               0
           Conv2d-15          [-1, 128, 16, 16]          16,512
             ReLU-16          [-1, 128, 16, 16]               0
   residual_block-17          [-1, 128, 16, 16]               0
           Conv2d-18          [-1, 128, 16, 16]         147,584
      BatchNorm2d-19          [-1, 128, 16, 16]             256
             ReLU-20          [-1, 128, 16, 16]               0
           Conv2d-21          [-1, 256, 16, 16]         295,168
      BatchNorm2d-22          [-1, 256, 16, 16]             512
             ReLU-23          [-1, 256, 16, 16]               0
           Conv2d-24          [-1, 128, 16, 16]          32,896
             ReLU-25          [-1, 128, 16, 16]               0
   residual_block-26          [-1, 128, 16, 16]               0
          Dropout-27          [-1, 128, 16, 16]               0
           Conv2d-28          [-1, 256, 16, 16]         295,168
        MaxPool2d-29            [-1, 256, 8, 8]               0
      BatchNorm2d-30            [-1, 256, 8, 8]             512
             ReLU-31            [-1, 256, 8, 8]               0
           Conv2d-32            [-1, 256, 8, 8]         590,080
      BatchNorm2d-33            [-1, 256, 8, 8]             512
             ReLU-34            [-1, 256, 8, 8]               0
           Conv2d-35            [-1, 256, 8, 8]          65,792
             ReLU-36            [-1, 256, 8, 8]               0
   residual_block-37            [-1, 256, 8, 8]               0
          Dropout-38            [-1, 256, 8, 8]               0
           Conv2d-39            [-1, 256, 8, 8]         590,080
        MaxPool2d-40            [-1, 256, 4, 4]               0
      BatchNorm2d-41            [-1, 256, 4, 4]             512
             ReLU-42            [-1, 256, 4, 4]               0
           Conv2d-43            [-1, 256, 4, 4]         590,080
      BatchNorm2d-44            [-1, 256, 4, 4]             512
             ReLU-45            [-1, 256, 4, 4]               0
           Conv2d-46            [-1, 256, 4, 4]          65,792
             ReLU-47            [-1, 256, 4, 4]               0
   residual_block-48            [-1, 256, 4, 4]               0
          Dropout-49            [-1, 256, 4, 4]               0
AdaptiveAvgPool2d-50            [-1, 256, 1, 1]               0
           Linear-51                  [-1, 128]          32,896
             ReLU-52                  [-1, 128]               0
          Dropout-53                  [-1, 128]               0
           Linear-54                   [-1, 10]           1,290
================================================================
Total params: 2,991,114
Trainable params: 2,991,114
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 12.19
Params size (MB): 11.41
Estimated Total Size (MB): 23.61
----------------------------------------------------------------
```

• Briefly describe what method do you apply? (e.g. data augmentation, model architecture, loss function, semi-supervised etc.) (10%)
Ans:

For training set, we use the following data augmentation:

```
train_transform = transforms.Compose([
            transforms.RandomHorizontalFlip(p=0.5),
            transforms.RandomRotation(25),
            transforms.ColorJitter(),
            transforms.ToTensor(),
            transforms.Normalize(means, stds),
        ])
```

The model architecture is composed of 5 modules, each module includes a CNN layer follows by a residual block. Some dropout layers are added after the block to prevent overfitting. After the 5 modules, we use an average pooling layer then flatten the result. Finally, the flattened vector is fed into the classifier that output a 10-dimension logit that represents each class. We choose Adam as optimizer with step learning rate scheduler and gamma equals to 0.6 on epoch 10, 20, 25 30. Using cross entropy as loss function, and implemented the method mentioned above, we train for 50 epochs. The accuracy on the public testing set can reach about 84.22%.