**Computer Vision HW2 Report**

Student ID: R10625016

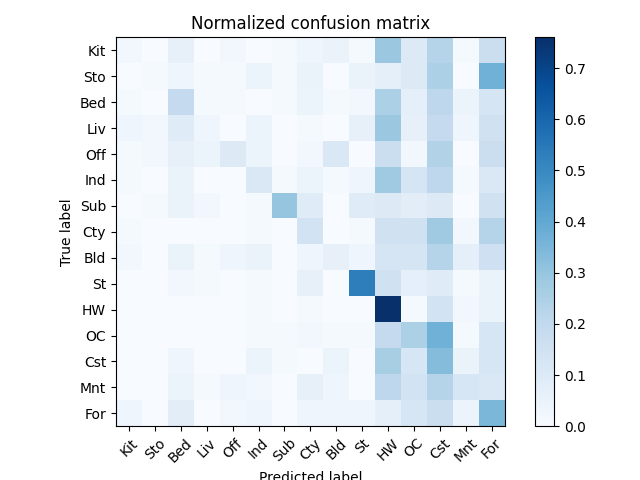
Name: 許致銓

**Part 1. (10%)**

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

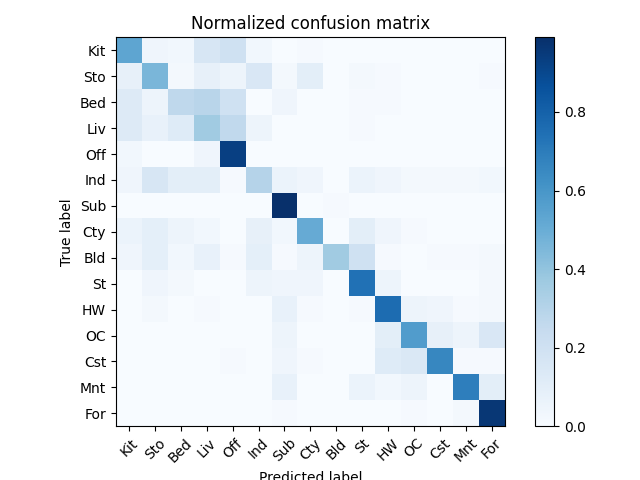
**Ans:**

**Tiny images:**

****

**(Fig.1)**

**Bag of sifts:**

****

**(Fig.2)**

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

1. **Tiny\_images:**

**accuracy = 0.22**

At first, I try to manage the KNN method by K in order to get a better performance. However, it didn’t work. Later, I found if I change the model by “cdist(metric)”, the performance will change much more than by K.

However, it is not enough for the 0.2 accuracy.

I resized the image to (10, 10). The result is much better than resizing to (16, 16). I think that is due to the pivotal image position, which is different from every image. Hence, if I only have a small part of the image, it may not get that much noisy information in this “low accuracy scale”, even get a little higher accuracy (0.22).

1. **Bag\_of\_sifts:**

**acc = 0.606**

In the “build\_vocabulary.py”, firstly, I think it is way important to get as much descriptors as I can. So, I set the step = 1. However, it didn’t work, and RAM even crashed. Later I found I don’t need that much descriptors at first, as long as I can get the right one to classify the images in the “get\_bags\_of\_sifts.py”. This is due to the purpose to build the vocabulary, which is not to compare, but merely to construct the standard. In so doing, as long as I can get the right descriptors in the latter part, I can still have a great classification.

As a result, I adjust the step of dsift to a larger one = 20 in “build\_vocabulary.py” to build the standard descriptors for classification centroids. Later, I use step = 2 to get descriptors of my training & testing data. This indeed help because of the scale. If I have a larger scale at first for centroids, I can have a better result when I have a more detailed scale to classify the training & testing data.

Eventually, I got accuracy = 0.606 with step adjustment and Kmean & KNN methos. Definitely, it is better than “tiny\_images” function.

**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%)**

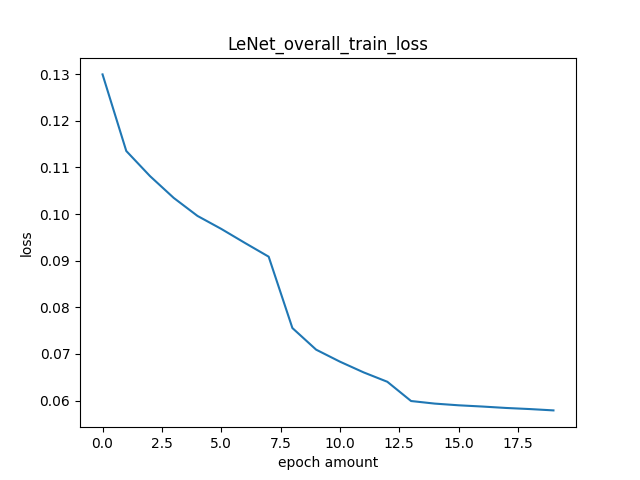
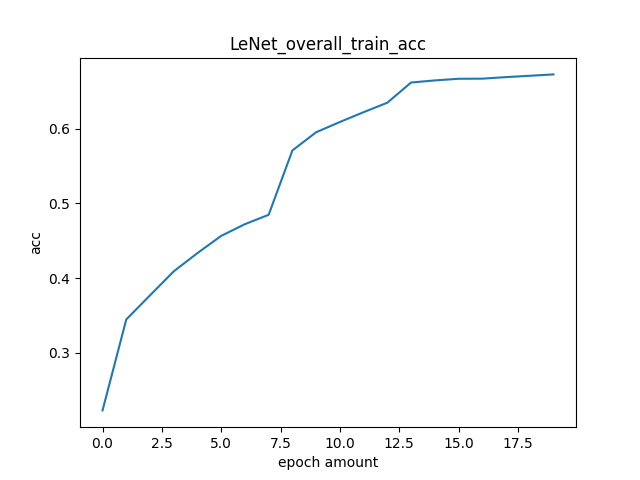
**Ans:**

**Comparision:**

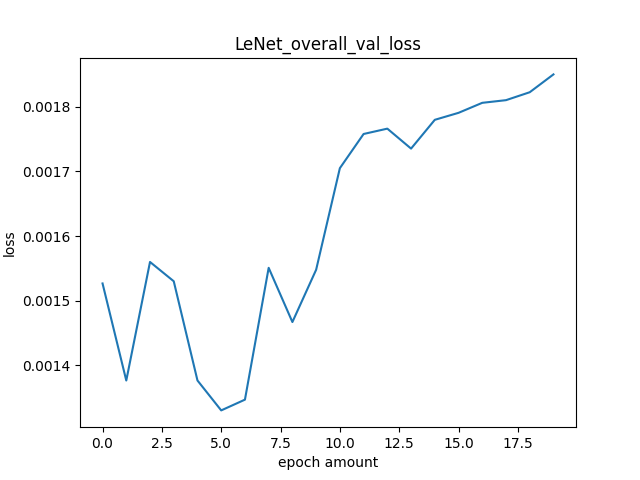
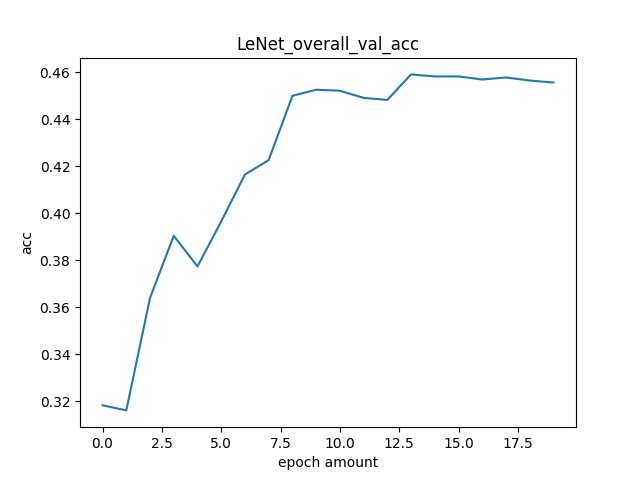
“myResnet” is better than “LeNet”. I can easily tell from the Fig.10 & Fig.6. Owing to myResnet has more CNN layers and FC layers. It tends to understand the data better, and has a strong ability to fit validation and testing data. Moreover, I think the ResNet NN is famous for its Residual Block, which is able to avoid being stuck in the local minimum. I think this helps a lot in finding the better minimum than LeNet. As a result, the accuracy on public testing data is way from 0.2 or so.

**LeNet:**

**Accuracy (on public test set):0.5192**

****

**(Fig.3) (Fig.4)**

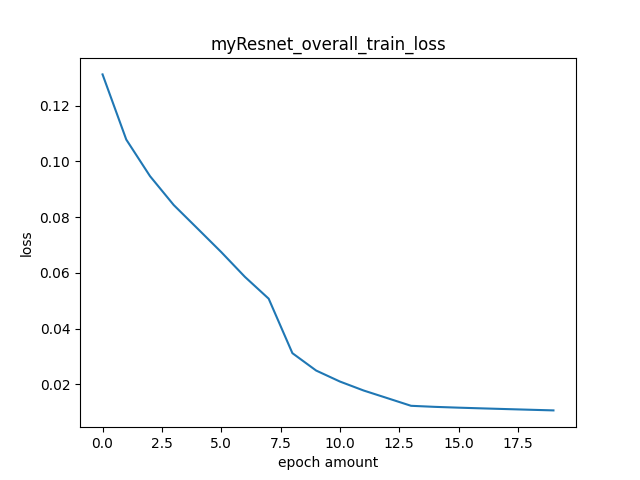
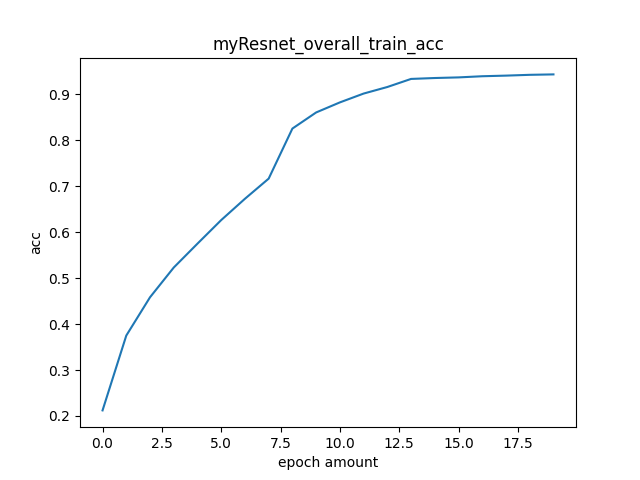
****

**(Fig.5) (Fig.6)**

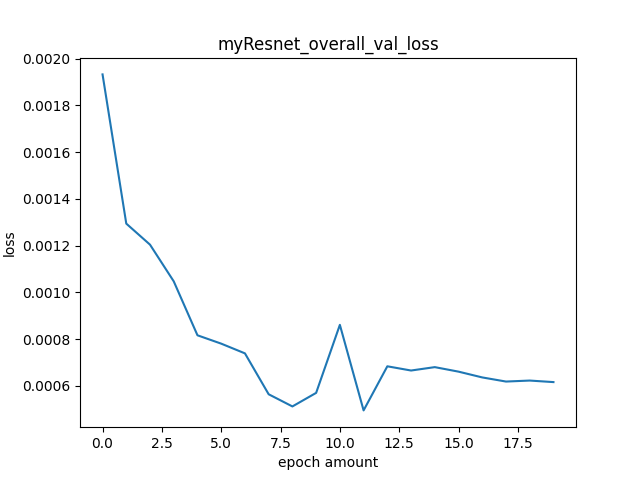
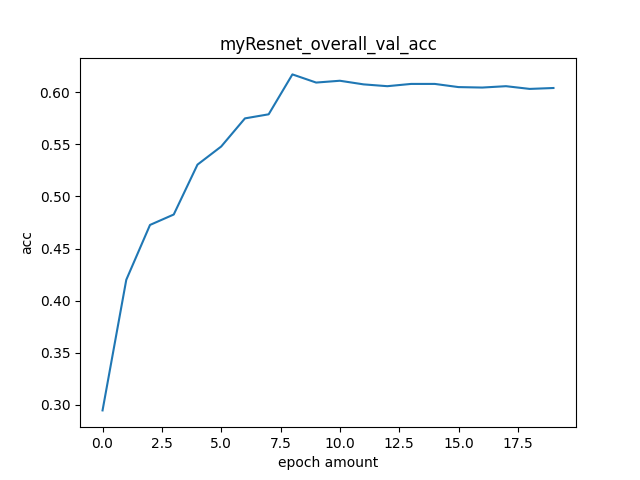
This is a little bit awkward in validation loss. I have no idea in this figure why it goes high. And I adjust the learning rate to 0.005, it has a better performance. However, I want to keep the original one to show this special (worse performance actually) figure.

**myResnet:**

**Accuracy (on public test set):0.6914**

****

**(Fig.7) (Fig.8)**

****

**(Fig.9) (Fig.10)**

After I adjust the learning rate to 0.004, I have a better loss result in validation set. I think this is mainly resulted from the smaller learning step, the less chance to jump out of the local minimum.

**• Attach basic information of the model you use including model architecture and the number of the parameters. Besides, report the accuracy you performed on the public test set. (5%)**

**Ans:**

**Resnext50:**

1. **Architecture:**

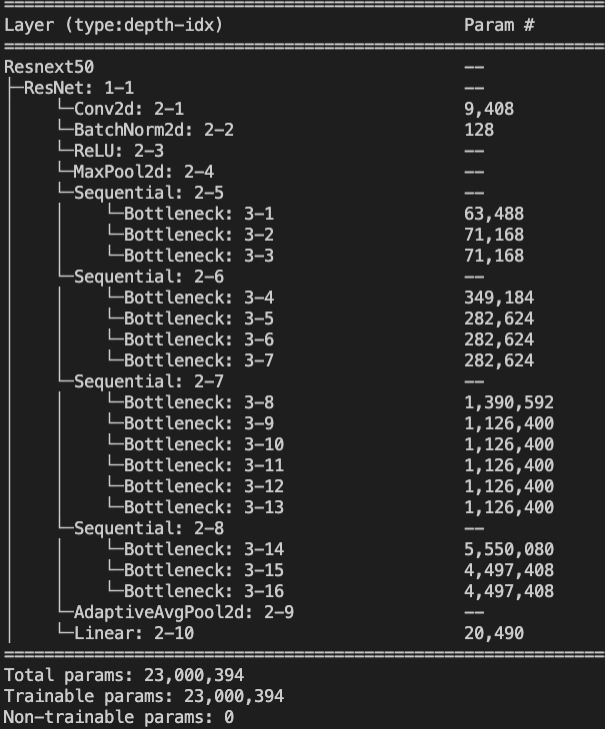
As you can see in the below figure, the. Architecture of ResNext50 is based on ResNet. Fisrtly, go through the Convolution2D, Batch Normalization, ReLU, and MaxPooling. Later on, go through the block of “Bottleneck”, which consists of three Convolution2D, three Batch Normalization, and ReLU.

Each Bottleneck has different parameters as input and output adjustment, and it can add Downsample if needed.

1. **Number of the parameters**

This is what I got from the “summary(models)”. There are 2300494 parameters used in Resnext50.

1. **Accuracy (on public test set): 0.8016**

****

**(Fig.11)**

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

**Ans:**

**Resnext50:**

1. **Data augmentation:**

I add “CentralCrop(16, 16)”, “RandomVerticalFlip()”, and “RandomHorizontalFlip()” as my transformation of data.

1. **Model architecture:**

In order to minimize the model.pt size, I try to change the fc layer to directly output the “num\_out” = 10.

This is because the more the fc layer exists, the larger the parameters I need to train, and model size will exceed 100MB.

**Though I use Resnext50, I want to have an introduction to my work on myResnet.**

**myResnet:**

1. **Data augmentation:**

To begin with, I used “CentralCrop(16, 16)”, “RandomVerticalFlip()”, and “RandomHorizontalFlip()”.

Later, I added “RandonCrop(16)”, “RandomRotation(60)”, and “RandomInvert(0.2)”, which make the model a bit better.

1. **Model architecture:**

myResnet consists of three Convolution2D, ReLU, and Maxpooling. Between Convolution2D layers, there residual block that makes the shortcut of the same dimension output. Lastly, there are five fc layer.

1. **Learning rate:**

0.004 is best for myResnet.

1. **Loss function:**

CrossEntropy(). With MSELoss(), it will result in the dimension problems.