Assignment 4

LI XINYA (G2004358J)

Step 1: Split the dataset:

As the assignment instruction says, we should use first 100 images in each class as the train, and then split the rest data into training and validation datasets. In this case, I split the 80% data as training dataset and 20% data as validation dataset. When splitting the dataset, I split different classes of pictures proportionally to avoid the problem with unbalanced datasets.

Then all the images will be divided into three parts: training dataset, validation dataset and testing dataset. Each dataset file contains four sub-files, which present four different classes of samples.

Step 2: Build the convolutional neural network:

Then I build the convolutional neural network through calling functions in “pytorch” packages. The CNN structure is shown below:

First layer:

Start with convolutional layer: the number of input channels is 3, the number of output channels is 64. And then add a maxpool function layer.

Second layer:

Start with convolutional layer: the number of input channels is 64, the number of output channels is 128. And then add a maxpool function layer.

Third layer:

Start with convolutional layer: the number of input channels is 128, the number of output channels is 256. And then add a maxpool function layer.

Output layer:

First is the linear layer with 1024 input features and 512 output features. And then add a dropout layer in case of overfit. The last layer is a linear layer with 512 input features and 4 output features, acting as the output result.

Step 3: Train and test the model:

The next step is training and testing the models. The initial parameters during training are:

Parameters	Value
Epoch	10
Learning rate	0.001
Batch size	32
Optimizer	Adam algorithm
Loss function	Cross Entropy Loss

And the result of the training is shown as below:

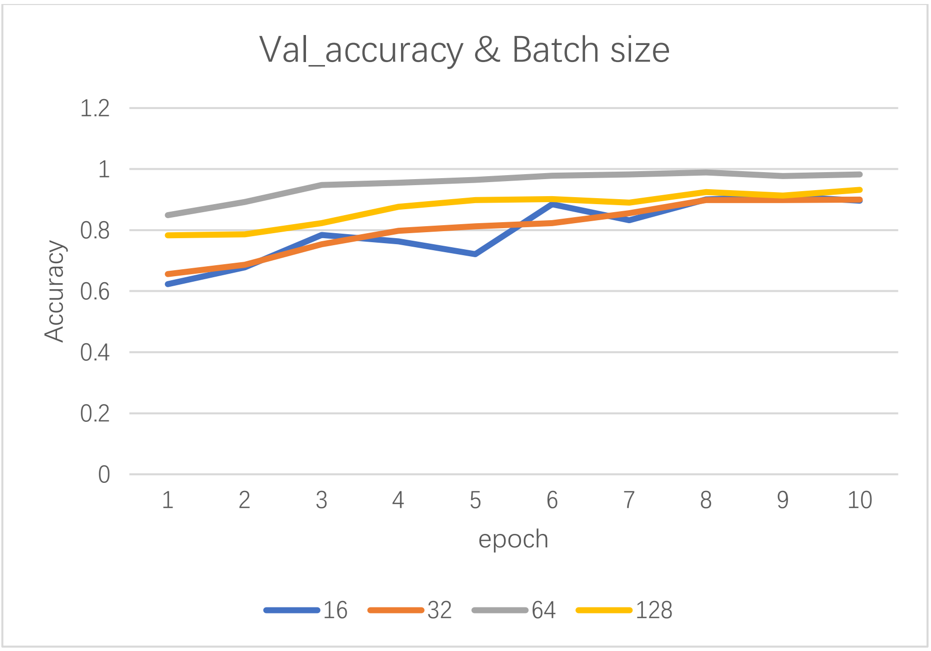
```
Processing the 0 epoch
epoch0 :validation loss: 0.44124635770553494, validation accuracy: 0.8379537448307551
Processing the 1 epoch
epoch1 :validation loss: 0.33224927739399235, validation accuracy: 0.8816051462704855
Processing the 2 epoch
epoch2 :validation loss: 0.2231911449397846, validation accuracy: 0.9156072905498545
Processing the 3 epoch
epoch3 :validation loss: 0.11316911774437602, validation accuracy: 0.960637157298208
Processing the 4 epoch
epoch4 :validation loss: 0.1496759559983308, validation accuracy: 0.9451677132792158
Processing the 5 epoch
epoch5 :validation loss: 0.061764819631573514, validation accuracy: 0.9796293459947925
Processing the 6 epoch
epoch6 :validation loss: 0.08192057683382456, validation accuracy: 0.9695206003982233
Processing the 7 epoch
epoch7 :validation loss: 0.08214488843364273, validation accuracy: 0.9693674375861541
Processing the 8 epoch
epoch8 :validation loss: 0.0438032423930364, validation accuracy: 0.9840710675448001
Processing the 9 epoch
epoch9 :validation loss: 0.02996016971642015, validation accuracy: 0.9908102312758462
```

Step 4: Find the smallest number of training images with high test accuracy:

If we blindly increase the batch size, the memory utilization will increase, but the memory capacity may be short; and the number of iterations required to run an epoch (full data set) is reduced, but it takes a lot of time to achieve the same accuracy If the batch size is increased to a certain extent, the correction of the parameters will appear to be slower; in addition, if the batch size is large to a certain extent, the determined declining direction has basically no longer changed.

However, within a reasonable range, increasing batch size also has advantages: memory utilization is improved, and the parallelization efficiency of large matrix multiplication is improved; And the number of iterations required to run an epoch (full data set) is reduced, and the processing speed for the same amount of data is further accelerated; in addition, within a certain range, generally speaking, the larger the batch size, the more accurate the determined descending direction will be, and the smaller the training shock will be.

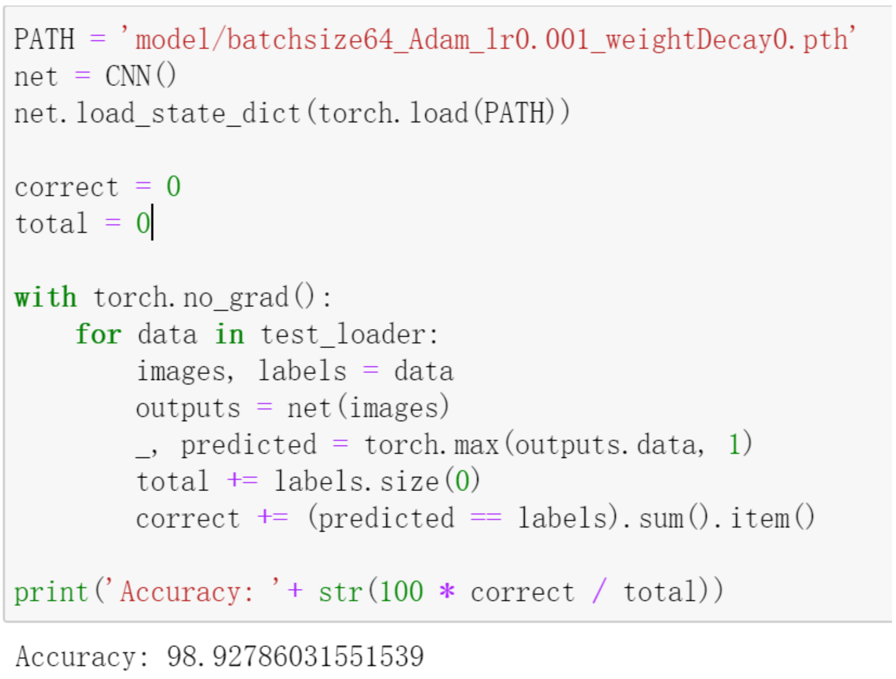
So, in this case, I choose different batch size under the same epoch numbers and CNN weights, then save the result in “Batch size.csv” files. And then draw a line plot to show the trends:



As you can see in the plot, when the batch size is 64, the accuracy will be the highest.

Result:

After test in test dataset (400 un-toughed images), the accuracy is around 98.9%.



```
PATH = 'model/batchsize64_Adam_lr0.001_weightDecay0.pth'
net = CNN()
net.load_state_dict(torch.load(PATH))

correct = 0
total = 0

with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy: '+ str(100 * correct / total))
```

```
Accuracy: 98.92786031551539
```