Handwritten digits (MNIST) recognition

INT305 Assignment 2

Yuliang Xiao
School of advanced technology
Xi'an Jiaotong-liverpool University
Suzhou, China
yuliang.Xiao19@student.xjtlu.edu.cn

***Abstract*—In this assignment, we train a convolutional neural network with two convolutional layers, two fully connected layers, and one max pooling layer, while based on MNIST data set. After training, the accuracy is 98.74%. Then, we shown the loss of the process of training when batch sizes are 64 and 512, respectively. Furthermore, we built and modified a ResNet18 to input the performance this network, while the accuracy is 99.38% while the to be convergent quickly.**

I. Introduction of convolutional neural network

A. Convolutional kernel

In image processing, a kernel, convolution matrix, or mask is a small matrix used for blurring, sharpening, embossing, edge detection, and more. The weight of the matrix is defined by a function, which is called convolution matrix. In the calculation, the pixels in a small area of the input image are average weighted via the convolutional neural and then, become each corresponding pixel in the output image, just like the example in Fig. 1. LeNet is the first convolutional neural network and in this network, it uses 5×5 convolutional kernel [1]. After that, 3×3 convolutional kernel be widely used firstly in VGGNet [2] and then, such size of convolutional kernel be used in most of other convolutional neural networks.

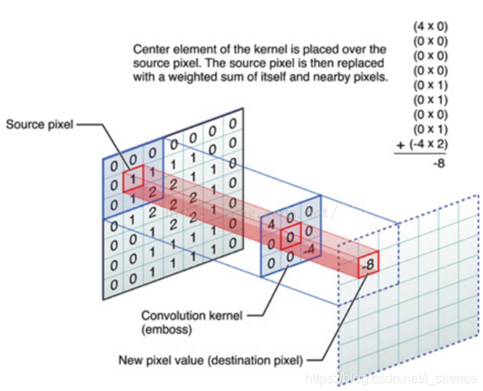


Fig. 1: The processing that how to use Convolutional kernel to calculate

In this CNN, there are two convolutional layers with 3×3 convolutional kernel. In pytorch library, the method $nn.Conv2d()$ is used to build the network. The code of this CNN is show in Appendix A.

B. Cross entropy

In this assignment, the loss function we used is cross entropy, whose calculation can be considered as SoftMax and logistic regression. Normally, it is expected to achieve ten categories after the pixel information from the image information thought the forward propagation network. After that, the confidence values will be calculated by SoftMax activation function, which can used to replace the loss function if it is necessary to know the predicted value directly.

For every input $x^{(i)}$, the result of the hypothesis function is the possibility $p(y^{(i)} = k|x^{(i);\theta})$ for class j. The hypothesis function is shown as

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i);\theta}) \\ p(y^{(i)} = 2|x^{(i);\theta}) \\ \vdots \\ p(y^{(i)} = k|x^{(i);\theta}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix} \tag{1}$$

Where the element $\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}$ is used to normalize the probability, which make sure the sum of each possibility is 1.

In the model of SoftMax, there is an indicator function, which expressed as

$$\mathbb{I}\{y^{(i)} = j\} = \begin{cases} 1 & \text{if } y = j \\ 0 & \text{if } y \neq j \end{cases} \tag{2}$$

And the cost function is

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} \sum_{j=1}^{k} \mathbb{I}\{y^{(i)} = j\} \log \left(p(y^{(i)} = j|x^{(i)}; \theta) \right) \right] \tag{3}$$

Where the possibility that input x be judged as class j via SoftMax is

$$p(y^{(i)} = j|x^{(i)};\theta) = \frac{e^{\theta_j^T x^{(i)}}}{\sum\limits_{l=1}^{k} e^{\theta_l^T x^{(i)}}} \tag{4}$$

Thus, equation (3) can be simplified as

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}\log\frac{e^{\theta_j^T x^{(i)}}}{\sum\limits_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right]$$
$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}\left[\theta_j^T x^{(i)} - \log\left(\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}\right)\right] \tag{5}$$

When we calculate the gradient of the cost function $\frac{\partial J(\theta)}{\partial \theta_j}$, we could consider this function has two parts, which is

$$\begin{cases} f(\theta_j) = \sum\limits_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}\theta_j^T x^{(i)} \\ g(\theta_j) = \sum\limits_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}\log\left(\sum\limits_{l=1}^{k} e^{\theta_l^T x^{(i)}}\right) \end{cases} \tag{6}$$

For $f(\theta_j) = \sum\limits_{j=1}^{k}\mathbb{I}y^{(i)} = j\theta_j^T x^{(i)}$, we have

$$\frac{\partial f(\theta_j)}{\partial \theta_j} = \mathbb{I}\{y^{(i)} = j\}x^{(i)} \tag{7}$$

For $g(\theta_j) = \sum\limits_{j=1}^{k}\mathbb{I}y^{(i)} = j\log\left(\sum\limits_{l=1}^{k} e^{\theta_l^T x^{(i)}}\right)$. Firstly, we need to get the derivative of the part of log function, which is

$$\frac{\partial\log\left(\sum\limits_{l=1}^{k} e^{\theta_l^T x^{(i)}}\right)}{\partial \theta_j} = \frac{e^{\theta_j x^{(i)}}x^{(i)}}{\sum\limits_{l=1}^{k} e^{\theta_l x^{(i)}}} \tag{8}$$

Then, we have the gradient of $g(\theta_j)$ as

$$\frac{\partial g(\theta_j)}{\partial \theta_j} = \sum_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}\frac{e^{\theta_j x^{(i)}}x^{(i)}}{\sum\limits_{l=1}^{k} e^{\theta_l x^{(i)}}} \tag{9}$$

As there is

$$\sum_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\} = 1 \tag{10}$$

The result of the gradient of $g(\theta_j)$

$$\frac{\partial g(\theta_j)}{\partial \theta_j} = \frac{e^{\theta_j x^{(i)}}x^{(i)}}{\sum\limits_{l=1}^{k} e^{\theta_l x^{(i)}}} = p(y^{(i)} = k|x^{(i);\theta}) \tag{11}$$

In the end, the whole derivation process to obtain the gradient for multiclass classification with SoftMax is

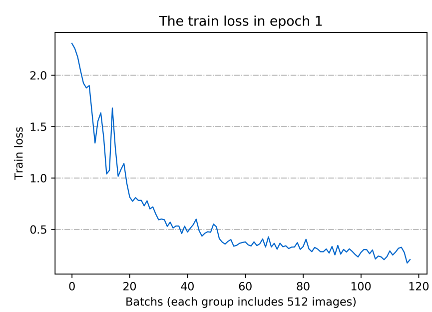
$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}x^{(i)} - \frac{e^{\theta_j x^{(i)}}x^{(i)}}{\sum\limits_{l=1}^{k} e^{\theta_l x^{(i)}}}\right]$$
$$= -\frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{k}\mathbb{I}\{y^{(i)} = j\}x^{(i)} - p(y^{(i)} = j|x^{(i)};\theta)\right] \tag{12}$$

II. The train and test of the CNN framework

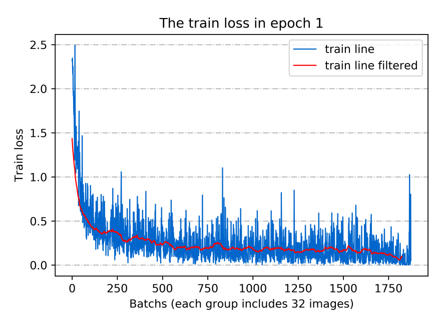
The total code used to train this CNN is shown in Appendix B.

A. Loss curve in training

In the training, we set the learning rate is 1 but not fixed, and it will be changed via StepLR method, which means the learning rate will multiply by the value of γ after finishing one epoch of train. In the training, we set γ is 0.7 and epoch is 15. Since a large batch size leads to a faster speed of train and a smoother loss curve [3], we set the batch size is 512, and the train time is 109s. However, a smaller batch size will lead to a better generalization performance [4] so we set the batch size is 32 and trained it again, which cost 167s. After testing these two models, we found both of their accuracy are 98.75%. However, after reading some reference, we known that the critical batch size is about 8000 [3], which means in our train, a large batch size can be better.



(a) The loss curve when batch size is 512



(b) The loss curve when batch size is 32

Fig. 2: The loss curve when batch size is different.

In Fig. 2, we show the loss curve in training when we set different batch size. It is clearly that when batch size is 512, the curve in Fig. 2(a) is smoother and it is decreased and convergent. However, the curve in Fig. 2(b), when batch size is 32, is not as visual as the front one. After dealing these data though Kalman filter, we found it has the same trend with the front one.

B. Accuracy

After training, the accuracy is 98.75% and the accuracy of each epoch is shown in Fig. 3. It is seemed that after the sixth epoch, the accuracy is stable.

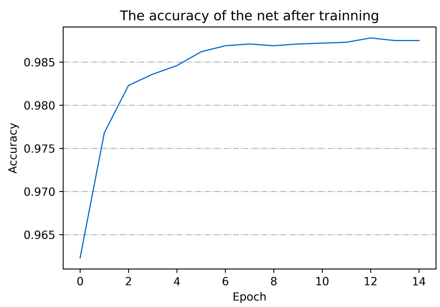
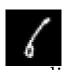


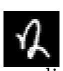
Fig. 3: The accuracy of CNN of each epoch

The final loss of the test set is 0.0412 and Fig. 4 shown some examples that the CNN ppredicted wrong. It is clearly that in Fig. 4(a)-(d), all the confidences are more than 0.9 but the result is wrong. In Fig. 4(a)-(d), all the confidences are just about 0.5.

(a) Target: 4, predict: 9
Confidence: 0.9787



(b) Target: 6, predict: 1
Confidence: 0.9081



(c) Target: 2, predict: 4
Confidence: 0.9028

(d) Target: 8, predict: 7
Confidence: 0.9040

(e) Target: 6, predict: 8
Confidence: 0.6296

(f) Target: 1, predict: 8
Confidence: 0.5342

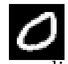
(g) Target: 4, predict: 9
Confidence: 0.5569

(h) Target: 9, predict: 1
Confidence: 0.4238

Fig. 4: Examples of the mis-classified images

Then, Fig. 5 gives some correct examples. We found the confidences are all more than 99% and even is 100%.

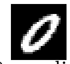
From these false examples, we found the confidence belong to $40-99\%$, which means this network is under fitting but, the confidence for most correct examples is more than 99%. Furthermore, we found that for most of false examples, they are difficult to classify by human-being. Additionally, this problem also happened in our



(a) Target: 0, predict: 0
Confidence: 1.0000

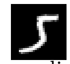
(b) Target: 6, predict: 6
Confidence: 1.0000

(c) Target: 0, predict: 0
Confidence: 1.0000



(d) Target: 0, predict: 0
Confidence: 1.0000

(e) Target: 2, predict: 2
Confidence: 0.9998



(f) Target: 5, predict: 5
Confidence: 0.9967

(g) Target: 8, predict: 8
Confidence: 0.9983

(h) Target: 5, predict: 5
Confidence: 0.9998

Fig. 5: Examples of the mis-classified images

modified ResNet18, so we considered that the best way to solve this problem is expending the data set.

III. Improve the convolutional neural network

In a CNN, a simple but effective way to prevent networks from over fitting is dropout, which is dropout some neural cell randomly in training [5]. Because of this algorithm, the generalization and robustness can be increased. However, in our network, we used the method of batch normalization, which can mitigate the phenomenon of gradient disappearance/explosion [6]. In the end, we modified the structure of ResNet18 to improve this convolutional neural network since it introduces residual and shortcut to solve this problem [7], which is came up with He Kaiming in 2016, *Deep residual learning for image recognition.*

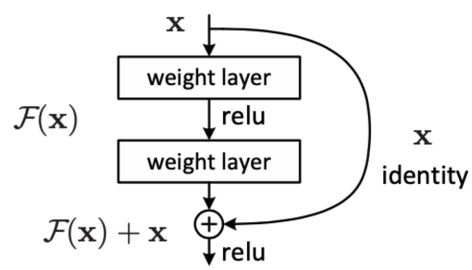


Fig. 6: The building block for residual learning

Firstly, we built a residual block, which is shown in Fig. 6, which can be defined as

$$y = F(x, W_i) + x \tag{13}$$

where x and y are the inout and output vectors of the layers considered. Then, the function $F(x, W_i)$ is the residual mapping to be learned. The shortcut connections in equation (C) does not result in other parameter and computation complexity, which is important and attractive in experiment. But when we apply this equation in calcuation, it is necessary to keep x and F in the equal dimensions. Therefore, if this porblem happen, W_s will be used to match the dimensions:

$$y = F(x, W_i) + W_s x \tag{14}$$

In our work, we changed the input channel into 1 and the output channel into 10, since our input is a gray image not a RGB image, and there are only 10 classifications as output. Furthermore, since handwritten digits recognition is a simple task and we hope to decrease the train time, we reduced the numbers of convolutional kernels in each layer. Then, the structure of our modified ResNet18 is shown in Table. I and the python code is shown in Appendix C.

TABLE I: The structure of moderfied ResNet18 in our assignment

layer name	18-layer
conv1	3×3, 8, stride 1
conv2_x	2×2 max pool, stride 1
	$\begin{bmatrix} 3\times3, & 16 \\ 3\times3, & 16 \end{bmatrix} \times 2$
conv3_x	$\begin{bmatrix} 3\times3, & 32 \\ 3\times3, & 32 \end{bmatrix} \times 2$
conv4_x	$\begin{bmatrix} 3\times3, & 64 \\ 3\times3, & 64 \end{bmatrix} \times 2$
conv5_x	$\begin{bmatrix} 3\times3, & 128 \\ 3\times3, & 128 \end{bmatrix} \times 2$
	average pool, 10-d fc, softmax

Fig. 7 shows the loss curve for this new network. Compared with Fig. 2 which is the loss of the simple CNN, the new one is convergent more quickly and earier.

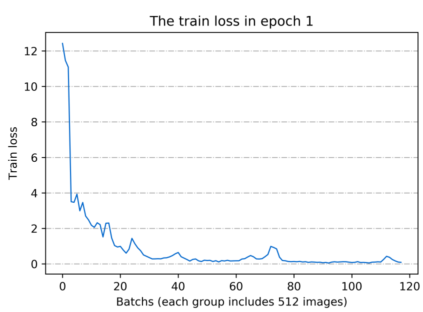


Fig. 7: The loss cruve in epoch 1

After training, the accuracy is 99.38%, higher than the old CNN, but the train time also more than the old one, which is 274. Then, Fig. 8 is the accuracy of each epoch of train. However, since this task is so simple and the data set is small, the advantage of our modified ResNet18, the accuracy is increased.

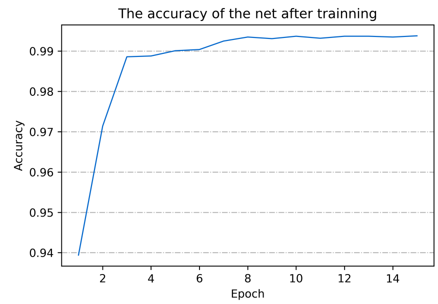


Fig. 8: The accuracy of ResNet18 of each epoch

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[4] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Appendix A
The python code of CNN

This is the model code of the CNN.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, 1)
        self.conv2 = nn.Conv2d(8, 16, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(2304, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        return x
```

Appendix B
The python code of trainning CNN

This is the python code used to train the CNN. In the training, we saved the loss generated by each epoch and the model be trained.

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
import os
import numpy as np
import shutil
from tqdm import trange
import time


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, 1)
        self.conv2 = nn.Conv2d(8, 16, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(2304, 64)
```

```
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        return x

def main(args):
    torch.manual_seed(args.seed)
    device = torch.device("cuda" if (torch.cuda.is_available() and args.use_cuda) else "cpu")

    train_loader, test_loader = dataset(args)

    model = Net().to(device)

    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    File = 'LOSS' + str(args.batch_size)
    fileCreate(File)

    accuracy = np.zeros([args.epochs, 1])

    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        accuracy[epoch-1] = test(model, device, test_loader)
        scheduler.step()
    saveData('accuracy.npy', accuracy, 'accuracy')

    if args.save_model is True:
        fileCreate('best_model')
        torch.save(model.state_dict(), "best_model/mnist_cnn.pt")

    return 0

def parse_args():
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=512, metavar='N', help='input batch size for training (default: 512)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N', help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=15, metavar='N', help='number of epochs to train (default: 15)')
    parser.add_argument('--num-workers', type=int, default=6, metavar='N', help='number of worker of torch to train (default: 10)')
    parser.add_argument('--lr', type=float, default=1, metavar='LR', help='learning rate (default: 1.0)')
```

```
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M', help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--use-cuda', action='store_true', default=True, help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False, help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S', help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N', help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True, help='For Saving the current Model')
    return parser.parse_args()

def dataset(args):
    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if torch.cuda.is_available() and args.use_cuda:
        cuda_kwargs = {'num_workers': args.num_workers,
                       'pin_memory': True,
                       'shuffle': True}
    train_kwargs.update(cuda_kwargs)
    test_kwargs.update(cuda_kwargs)

    transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
    ])

    trainset = datasets.MNIST('./data', train=True, download=True, transform=transform)
    testset = datasets.MNIST('./data', train=False, transform=transform)

    train_loader = torch.utils.data.DataLoader(trainset,**train_kwargs)
    test_loader = torch.utils.data.DataLoader(testset, **test_kwargs)

    return train_loader, test_loader

def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    sum_up_batch_loss = 0
    lossFile = 'LOSS' + str(args.batch_size) + '/loss_epoch' + str(epoch) + '.npy'

    with trange(len(train_loader)) as pbar:
        for batch_idx, ((data, target), i) in enumerate(zip(train_loader, pbar)):
            pbar.set_description(f"epoch{epoch}/{args.epochs}")
            data, target = data.to(device), target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = F.cross_entropy(output, target)
            loss.backward()
            optimizer.step()

            sum_up_batch_loss += loss.cpu().detach().numpy()
            average_loss = sum_up_batch_loss/(batch_idx+1)
            pbar.set_postfix({'loss':'{:.4f}'.format(loss.cpu().detach().numpy()), 'average loss
```

```
                 ’:’{:.4f}’.format(average_loss)})

            saveData(lossFile, loss.cpu().detach().numpy(), ’loss’)

    return 0

def saveData(file, data, item_name):
    if os.path.exists(file) is True:
        dictionary = np.load(file, allow_pickle= True).item()
        data_temp = dictionary[item_name]
        data = np.append(data_temp, data)

    dictionary = {item_name: data}
    np.save(file, dictionary)

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct_num = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.cross_entropy(output, target, reduction=’sum’).item()
            predict = output.argmax(dim=1, keepdim=True)
            correct_num += predict.eq(target.view_as(predict)).sum().item()

    test_loss /= len(test_loader.dataset)
    accuracy = correct_num / len(test_loader.dataset)

    print(’\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.4f}%)\n’.format(test_loss,
        correct_num, len(test_loader.dataset), 100.*accuracy))

    return accuracy

def fileCreate(fileName):
    if os.path.exists(fileName) is True:
        shutil.rmtree(fileName)
        os.makedirs(fileName)
    else:
        os.makedirs(fileName)


if __name__ == ’__main__’:
    args = parse_args()
    main(args)
```

Appendix C
The python code of ResNet18

This is the model code of the ResNet18.

```
class Residual(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Residual, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
```

```
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        if in_channels != out_channels:
            self.conv3 = nn.Conv2d(in_channels, out_channels, 1, 1)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, x):
        y = F.relu(self.bn1(self.conv1(x)))
        y = self.bn2(self.conv2(y))
        if self.conv3:
            x = self.conv3(x)
        x = F.relu(torch.add(x, y))
        return x


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, 1)
        self.bn1 = nn.BatchNorm2d(8)
        self.bk1 = Residual(8, 16)
        self.bk2 = Residual(16, 16)
        self.bk3 = Residual(16, 32)
        self.bk4 = Residual(32, 32)
        self.bk5 = Residual(32, 64)
        self.bk6 = Residual(64, 64)
        self.bk7 = Residual(64, 128)
        self.bk8 = Residual(128, 128)
        self.fc = nn.Linear(36*128, 10)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.max_pool2d(x, 2)
        x = self.bk1(x)
        x = self.bk2(x)
        x = self.bk3(x)
        x = self.bk4(x)
        x = self.bk5(x)
        x = self.bk6(x)
        x = self.bk7(x)
        x = self.bk8(x)
        x = F.avg_pool2d(x, 2)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

Appendix D
The python code of trainning ResNet18

This is the python code used to train ResNet18. In the training, we saved the loss generated by each epoch and the model be trained.

```
from __future__ import print_function
import argparse
```

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
import os
import numpy as np
import shutil
from tqdm import trange
import time


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, 1)
        self.conv2 = nn.Conv2d(8, 16, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(2304, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        return x

def main(args):
    torch.manual_seed(args.seed)
    device = torch.device(”cuda” if (torch.cuda.is_available() and args.use_cuda) else ”cpu”)

    train_loader, test_loader = dataset(args)

    model = Net().to(device)

    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    File = ’LOSS’ + str(args.batch_size)
    fileCreate(File)

    accuracy = np.zeros([args.epochs, 1])

    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        accuracy[epoch-1] = test(model, device, test_loader)
```

```python
        scheduler.step()
    saveData('accuracy.npy', accuracy, 'accuracy')

    if args.save_model is True:
        fileCreate('best_model')
        torch.save(model.state_dict(), "best_model/mnist_cnn.pt")

    return 0

def parse_args():
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=512, metavar='N', help='input batch size for training (default: 512)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N', help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=15, metavar='N', help='number of epochs to train (default: 15)')
    parser.add_argument('--num-workers', type=int, default=6, metavar='N', help='number of worker of torch to train (default: 10)')
    parser.add_argument('--lr', type=float, default=1, metavar='LR', help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M', help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--use-cuda', action='store_true', default=True, help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False, help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S', help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N', help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True, help='For Saving the current Model')
    return parser.parse_args()

def dataset(args):
    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if torch.cuda.is_available() and args.use_cuda:
        cuda_kwargs = {'num_workers': args.num_workers,
                       'pin_memory': True,
                       'shuffle': True}
    train_kwargs.update(cuda_kwargs)
    test_kwargs.update(cuda_kwargs)

    transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
    ])

    trainset = datasets.MNIST('./data', train=True, download=True, transform=transform)
    testset = datasets.MNIST('./data', train=False, transform=transform)

    train_loader = torch.utils.data.DataLoader(trainset,**train_kwargs)
    test_loader = torch.utils.data.DataLoader(testset, **test_kwargs)
```

```python
    return train_loader, test_loader

def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    sum_up_batch_loss = 0
    lossFile = 'LOSS' + str(args.batch_size) + '/loss_epoch' + str(epoch) + '.npy'

    with trange(len(train_loader)) as pbar:
        for batch_idx, ((data, target), i) in enumerate(zip(train_loader, pbar)):
            pbar.set_description(f"epoch{epoch}/{args.epochs}")
            data, target = data.to(device), target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = F.cross_entropy(output, target)
            loss.backward()
            optimizer.step()

            sum_up_batch_loss += loss.cpu().detach().numpy()
            average_loss = sum_up_batch_loss/(batch_idx+1)
            pbar.set_postfix({'loss':'{:.4f}'.format(loss.cpu().detach().numpy()), 'average loss':'{:.4f}'.format(average_loss)})

            saveData(lossFile, loss.cpu().detach().numpy(), 'loss')

    return 0

def saveData(file, data, item_name):
    if os.path.exists(file) is True:
        dictionary = np.load(file, allow_pickle= True).item()
        data_temp = dictionary[item_name]
        data = np.append(data_temp, data)

    dictionary = {item_name: data}
    np.save(file, dictionary)

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct_num = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.cross_entropy(output, target, reduction='sum').item()
            predict = output.argmax(dim=1, keepdim=True)
            correct_num += predict.eq(target.view_as(predict)).sum().item()

    test_loss /= len(test_loader.dataset)
    accuracy = correct_num / len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.4f}%)\n'.format(test_loss, correct_num, len(test_loader.dataset), 100.*accuracy))

    return accuracy
```

```python

def fileCreate(fileName):
    if os.path.exists(fileName) is True:
        shutil.rmtree(fileName)
        os.makedirs(fileName)
    else:
        os.makedirs(fileName)


if __name__ == '__main__':
    args = parse_args()
    main(args)
```