INT 305 Lab Report

Xu Zhao
1927631

Part 1

1. Convolution Kernel

In computer vision, a kernel is a small matrix used for blurring, sharpening, embossing, edge detection and more. This is accomplished by doing a convolution between the kernel and an image.

The general expression of a convolution is

$$g(x,y) = w * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx,dy) f(x-dx, y-dy),$$

Where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image, w is the filtered kernel. Every element of the filter kernel is considered by $-a \le dx \le a$ and $-b \le dy \le b$.

In this project, there are two convolutional layers. In the first convolutional layer, there are 8 kernels which transform the pictures from 1 channel into 8 channels. Each kernel is a 3*3 matrix and the stride is 1. In the second convolutional layer, the 8-channel is transformed to 16-channel. The kernels are also 3*3 matrixes and the stride is 1.

2. Loss Function

In machine learning, the loss function is used to evaluate how different between the model prediction and the ground truth. An optimization problem seeks to minimize a loss function.

In this project, multiclass classification with softmax and cross-entropy loss are used. The loss function is

$$L_{CE}(y,t) = -\sum_{k=1}^{K} t_k \log y_k = -t^T(\log y)$$

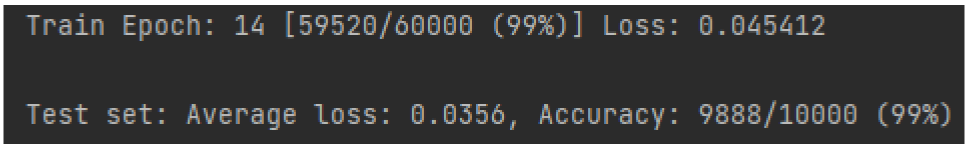
Where y is the prediction, t is the ground truth, k is the k-th class possibility in the output while K is the number of classes.

There are 10 neurons in the output layer which represent the possibility of the prediction to the 1o numbers. The output is the distribution of the model prediction and the target is the ground truth of the dataset.

Part 2

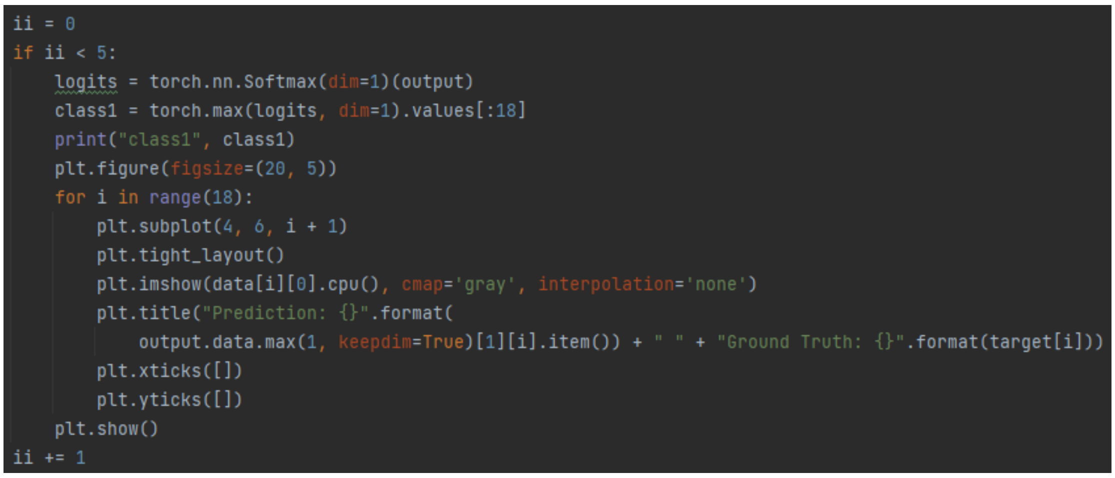
1. Final Accuracy Performance

After training the model, the accuracy of prediction is about 99%, which is very high. In the last epoch, the test result is



```
Train Epoch: 14 [59520/60000 (99%)] Loss: 0.045412

Test set: Average loss: 0.0356, Accuracy: 9888/10000 (99%)
```

2. Some Prediction Results

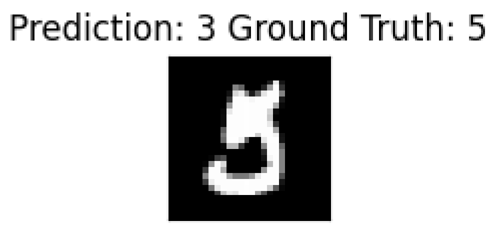


```
ii = 0
if ii < 5:
    logits = torch.nn.Softmax(dim=1)(output)
    class1 = torch.max(logits, dim=1).values[:18]
    print("class1", class1)
    plt.figure(figsize=(20, 5))
    for i in range(18):
        plt.subplot(4, 6, i + 1)
        plt.tight_layout()
        plt.imshow(data[i][0].cpu(), cmap='gray', interpolation='none')
        plt.title("Prediction: {}".format(
            output.data.max(1, keepdim=True)[1][i].item()) + " " + "Ground Truth: {}".format(target[i]))
        plt.xticks([])
        plt.yticks([])
    plt.show()
ii += 1
```

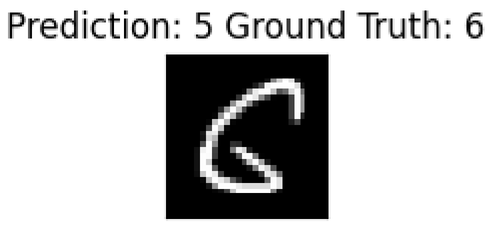
The implementation of drawing pictures and getting confidence value

- Misclassified Images

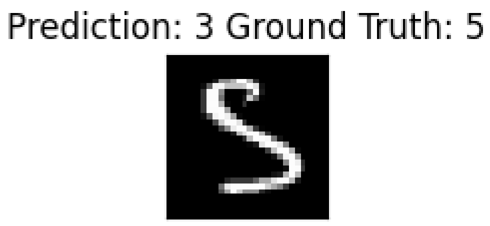
For some vague handwriting, the model cannot predict very well



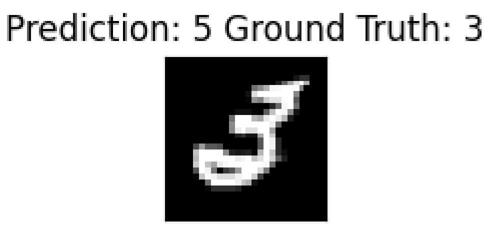
confidence value: 0.8292



confidence value: 0.4512



confidence value: 0.5770

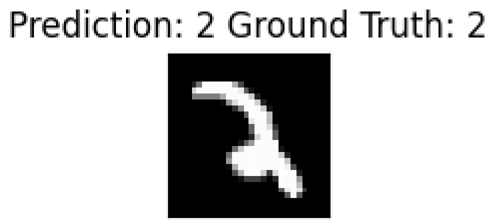


confidence value: 4871

- Well Classified Images

Although these handwriting is not very good, the model can still predict them correctly.

Prediction: 7 Ground Truth: 7
confidence value: 0.9647



confidence value: 0.9998

Prediction: 1 Ground Truth: 1
confidence value: 0.9997

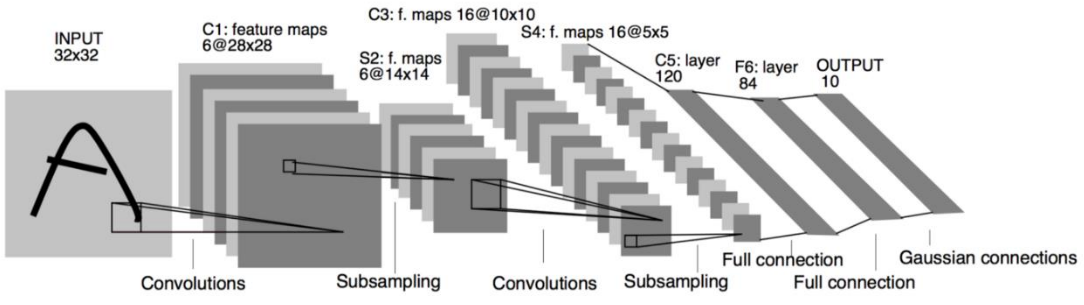
Prediction: 5 Ground Truth: 5
confidence value: 0.9873

Prediction: 0 Ground Truth: 0
confidence value: 0.9992

Prediction: 7 Ground Truth: 7
confidence value: 0.6356

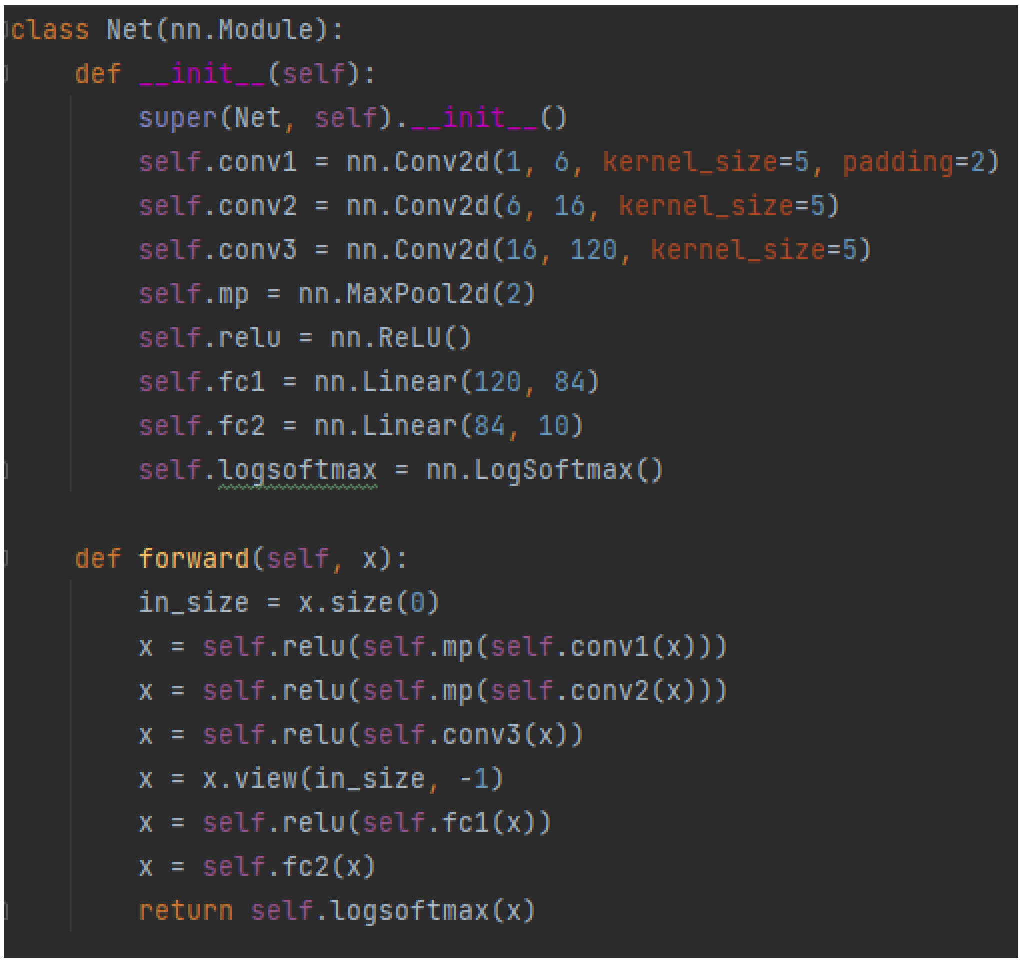
Part 3

For the improvement, I replace the network with LeNet-5 [1], which is introduced to recognize handwriting numbers. It has 3 convolution layers, 2 pooling layers and 2 full connection layers, which is deeper and more complex than the original one. This network has a better performance.



The architecture of LeNet-5

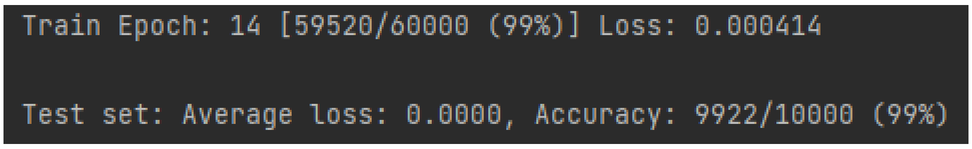
However, this network was introduced very early. Due to the limitations of times, sigmoid is used as the activation function rather than ReLU, Gaussian Connections is used as the last layer of multiclassification rather than softmax. The pooling layers uses average pooling while it is possible that max pooling works better for classification tasks. As a result, these are improved in the LeNet-5 for this project.



```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, padding=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(120, 84)
        self.fc2 = nn.Linear(84, 10)
        self.logsoftmax = nn.LogSoftmax()

    def forward(self, x):
        in_size = x.size(0)
        x = self.relu(self.mp(self.conv1(x)))
        x = self.relu(self.mp(self.conv2(x)))
        x = self.relu(self.conv3(x))
        x = x.view(in_size, -1)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return self.logsoftmax(x)
```

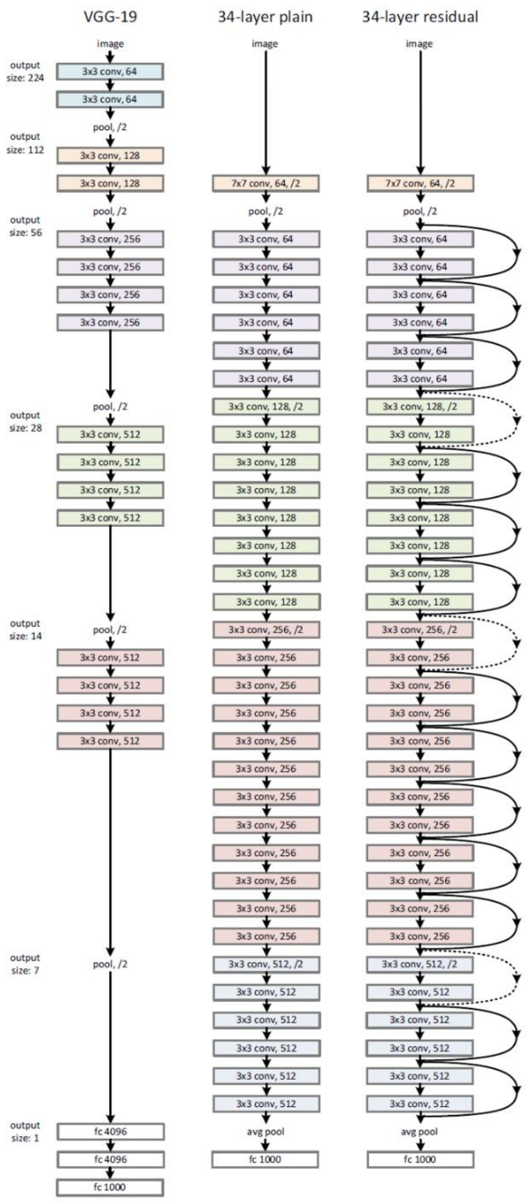
The implementation of LeNet-5

Apply LeNet-5 and keep the hyperparameters the same as the original one. The accuracy of prediction is higher. In the last epoch, the accuracy is 99.22% while the original one is 98.88%, the test result is:

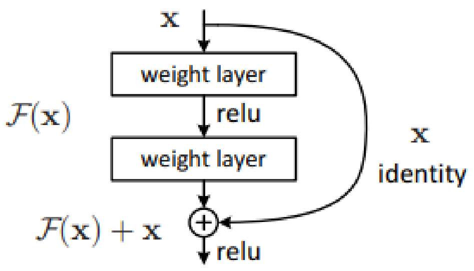


```
Train Epoch: 14 [59520/60000 (99%)] Loss: 0.000414

Test set: Average loss: 0.0000, Accuracy: 9922/10000 (99%)
```

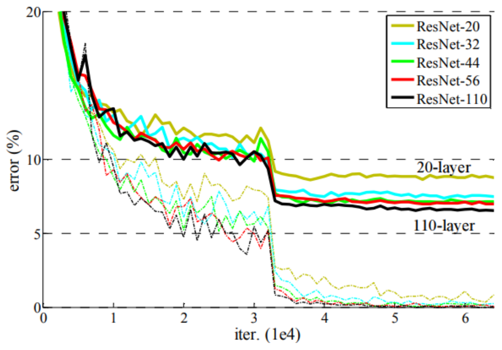
The techniques of residual neural network from Deep Residual Learning for Image Recognition [2] is also tried. It has very deep feedforward neural network with hundreds of layers, much deeper than previous networks. Before Resnet, people think more convolution layers and pooling layers can make the performance of image recognition better. However, there are problems such as gradient vanishing, gradient exploding and degradation in some very deep networks. A residual framework is introduced. It has two main features: 1. Use data preprocessing and Batch Normalization layer in the network to solve the problem of gradient vanishing and gradient exploding. 2. Use residual structure which can weaken the strong connection between each layer to solve the degradation problem.



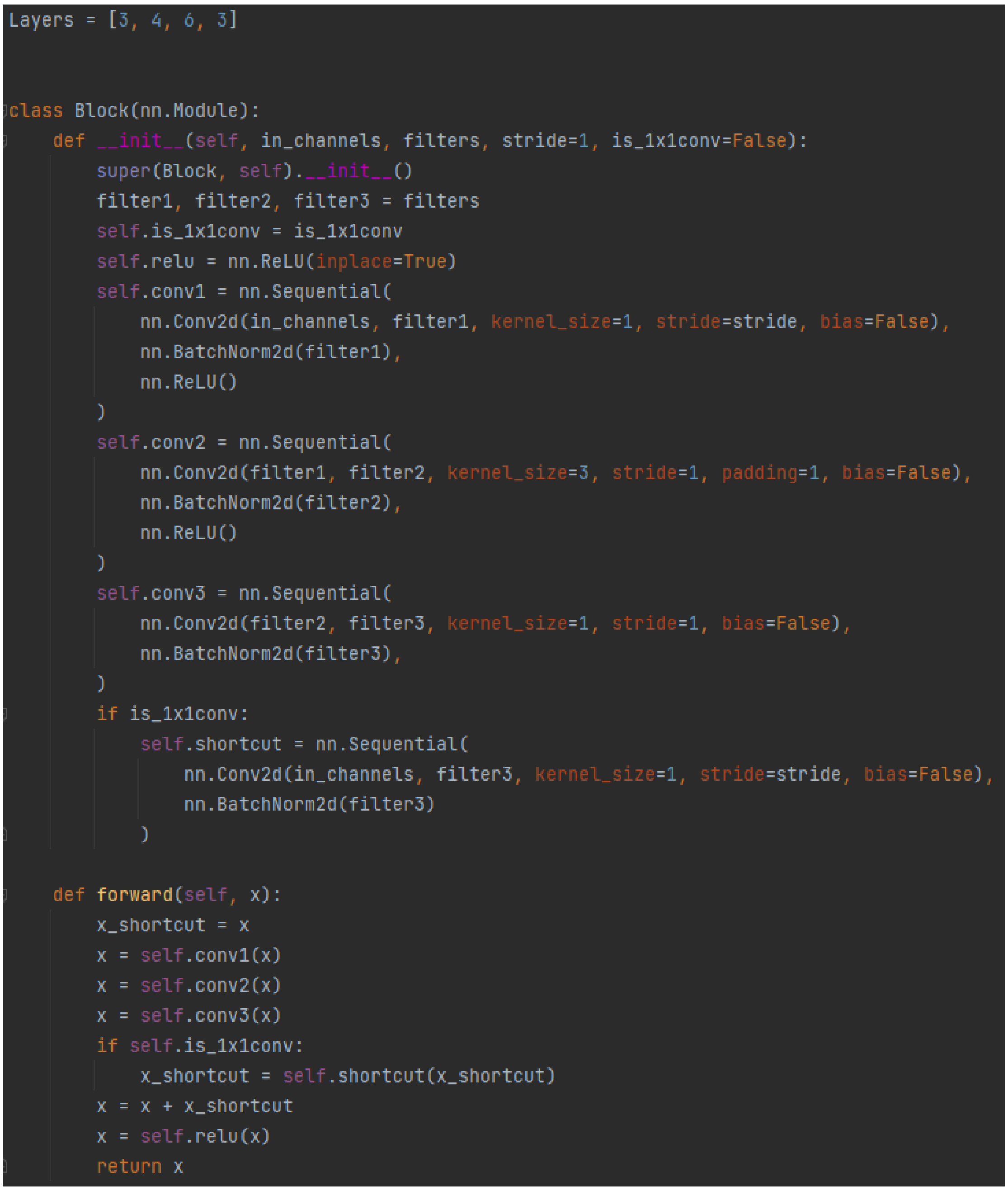
The architecture of ResNet [2]


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Residual learning: a building block [2]

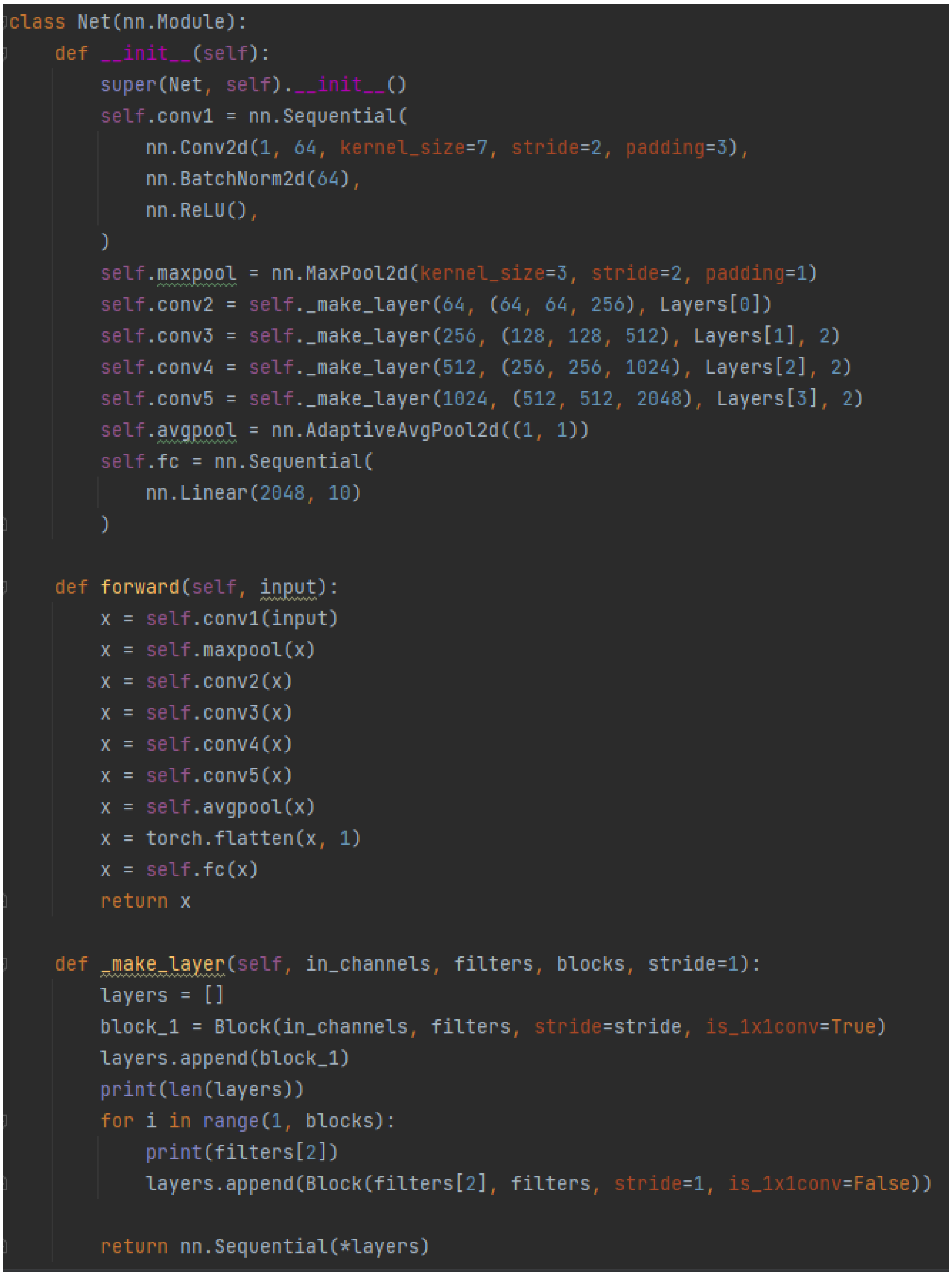


Training on CIFAR-10. Dashed lines denote training error, and bold lines denote testing error. As the network becomes deeper, the performance is better [2].



```python
Layers = [3, 4, 6, 3]


class Block(nn.Module):
    def __init__(self, in_channels, filters, stride=1, is_1x1conv=False):
        super(Block, self).__init__()
        filter1, filter2, filter3 = filters
        self.is_1x1conv = is_1x1conv
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, filter1, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(filter1),
            nn.ReLU()
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(filter1, filter2, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(filter2),
            nn.ReLU()
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(filter2, filter3, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(filter3),
        )
        if is_1x1conv:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, filter3, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(filter3)
            )

    def forward(self, x):
        x_shortcut = x
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        if self.is_1x1conv:
            x_shortcut = self.shortcut(x_shortcut)
        x = x + x_shortcut
        x = self.relu(x)
        return x
```

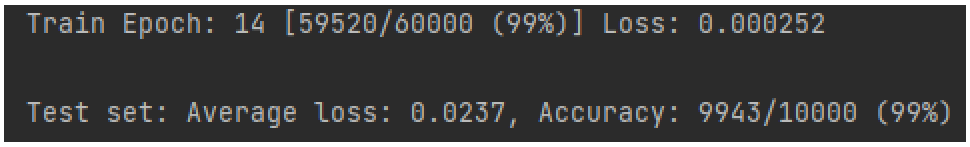
The implementation of ResNet part 1



```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
        )
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.conv2 = self._make_layer(64, (64, 64, 256), Layers[0])
        self.conv3 = self._make_layer(256, (128, 128, 512), Layers[1], 2)
        self.conv4 = self._make_layer(512, (256, 256, 1024), Layers[2], 2)
        self.conv5 = self._make_layer(1024, (512, 512, 2048), Layers[3], 2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Sequential(
            nn.Linear(2048, 10)
        )

    def forward(self, input):
        x = self.conv1(input)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x

    def _make_layer(self, in_channels, filters, blocks, stride=1):
        layers = []
        block_1 = Block(in_channels, filters, stride=stride, is_1x1conv=True)
        layers.append(block_1)
        print(len(layers))
        for i in range(1, blocks):
            print(filters[2])
            layers.append(Block(filters[2], filters, stride=1, is_1x1conv=False))

        return nn.Sequential(*layers)
```

The implementation of ResNet part 2

Apply ResNet and keep the hyperparameters the same as the original one. The accuracy of prediction is higher. In the last epoch, the accuracy is 99.43% while the original one is 98.88%, the test result is:



```
Train Epoch: 14 [59520/60000 (99%)] Loss: 0.000252


Test set: Average loss: 0.0237, Accuracy: 9943/10000 (99%)
```

Reference

[1] Y. LeCun, B. Boser, J. Denker, *et al.*, “Handwriting digit recognition with a back-propagation network, *Advances in neural information processing systems*, 2, 1989.

[2] K. He, X. Zhang, S. Ren, *et al.*, “Deep residual learning for image recognition”, *Proceeding of the IEEE conference on computer vision and pattern recognition,* pp. 770-778, 2016.