COGS181 Final Project

Experiments On Improving Performance Of Convolution Neural Network And Residual Network On Image Recognition

Xiaofei Teng
A16163740
xteng@ucsd.edu

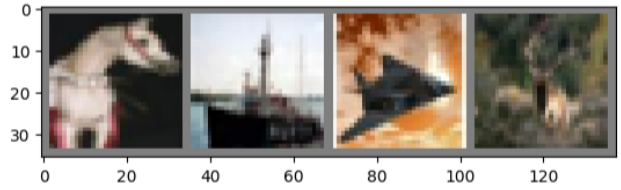
Abstract

This project studied on comparing and contrast the performances of Convolution Neural Network(CNN) and Residual Network(ResNet) on the task of image classification. In practice, I implement AlexNet as a typical form of traditional CNN and ResNet18 as a typical form of ResNet. I also implement my own custom CNN and ResNet and adjust the hyperparameters, such as total epochs and depth of the network, to see how tuning these hyperparameters would influence the performance on the image recognition task. In general, ResNet has a better average performance than CNN and better sensitivity to hyperparameter-tuning, meaning that deepen the network or lengthen the learning cycle has more significant effect on the result.

Introduction

The inspiration comes from the article *Deep Residual Learning for Image Recognition* (He et al., 2015), where the idea of residual learning blocks was integrated into traditional plain CNN to improve performance on the recognition task. In the article, the authors compared ResNet with traditional plain CNN, and also compared ResNet with different depth and layer design. This is also the main design of this project. In this project, I will compare the performance of CNN and ResNet on image classification task, and will also perform hyperparameter-tuning on both networks to explore how and how much I could improve their performances.

The dataset I will be using is the CIFAR-10 dataset, which contains 60,000 sample. Each sample if a 32*32 RGB image, which means that each input image is 3*32*32. The number of classes for this dataset is 10, namely 'plane', 'car', 'bird', 'cat', deer', 'dog', 'frog', 'horse', 'ship', and 'truck'. All the samples are pre-labeled, meaning that this is a supervised machine learning task. The dataset is built-in in the PyTorch library. Examples from the CIFAR-10 dataset are shown as the following:




```
horse  ship plane  deer
```

Other relative works include AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) and the online open-source GitHub repo of ImageNet image classification by tjmoon0104 (tjmoon0104/pytorch-tiny-imagenet: pytorch-tiny-imagenet (github.com)). The former one provides a design of traditional plain CNN that could be used to test on, and the latter provides reference for detail coding on both CNN and ResNet.

ChatGPT is used for debugging in this project.

Method

Pre-process of the dataset

First, I calculated the mean and the standard derivation of the CIFAR-10 dataset and performed the transform accordingly. The following piece of code is for calculating the mean and derivation of the dataset:

```
transform = transforms.ToTensor()
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=len(train_dataset), shuffle=False)

mean = torch.zeros(3)
std = torch.zeros(3)
for images, _ in train_loader:
    num_images = images.size(0)
    images = images.view(num_images, images.size(1), -1)
    mean = images.mean(dim=(0, 2))
    std = images.std(dim=(0, 2))
mean = mean.tolist()
std = std.tolist()

print("Mean of the dataset:", mean)
print("Standard deviation of the dataset:", std)
```

```
Files already downloaded and verified
Mean of the dataset: [0.491400808095932, 0.48215898871421814, 0.44653093814849854]
Standard deviation of the dataset: [0.24703224003314972, 0.24348513782024384, 0.26158785820007324]
```

The dataset contains 4 dimensions, namely N (number of samples), C = 3 (number of channels), H (height), and W (width). The three entries of the mean and standard derivation each represent one dimension of N, H, and W.

Training and evaluation of typical CNN and ResNet

I selected AlexNet and ResNet18 to be the representative CNN and ResNet, trained each model on CIFAR-10 and using livelossplot (a python library that is used to record the live loss and accuracy of a training process) to plot the loss-epoch and accuracy-epoch curve to evaluate the performance of the two models.

Note that both AlexNet and ResNet18 were originally designed to be working on ImageNet, which has a 224*224 input size, thus another transformation to resize the dataset must be performed on CIFAR-10 before feeding it to the models.

```
transform = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)])

train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers=2)

test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=4, shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Exploring the effect of hyperparameter-tuning on custom CNN

For exploring the effect of hyperparameter-tuning on CNN, I designed a custom CNN called myCNN. The class definition of myCNN is as following:

In [55]:
```
class myCNN(nn.Module):
    def __init__(self, num_blocks, input_size, hidden_size, output_size):
        super(myCNN, self).__init__()

        self.conv = nn.Conv2d(input_size, hidden_size, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        blocks = [nn.Sequential(
            nn.Conv2d(hidden_size, hidden_size, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        ) for i in range(num_blocks-1)]
        self.blocks = nn.Sequential(*blocks)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.blocks(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

The network is composed of several “convolution layer blocks”, each of which contains: a convolution layer with kernel_size=3, stride=1, padding=1; a regularization layer using the ReLU function; a pooling layer using maxpool with kernel_size=2 and stride=2. The total number of “convolution layer blocks” are indicated by the hyperparameter num_blocks, which is an integer from 1 to 5. The reason that num_blocks cannot be greater than 5 is because that each “convolution layer block” contains a maxpool layer that would shrink the image size (both height and width) by 0.5, and the original image input is of size 32*32. Thus, after 5 such blocks, the input size become 1*1 and can no longer be fed into another “convolution layer block”. The first block takes 3 as input channel number and output channels equal to the hyperparameter hidden_size, which is set to 32 in my experiments. The rest of the blocks take in hidden_size channels and output the same number of channels.

Using a gridsearch on num_blocks with 10 epochs, I selected the model with the best performance (num_blocks=4), and further explore on how to improve its performance. The experiment was done from two perspectives: one is to lengthen the training procedure, namely increase the total epochs, and the other is to deepen the network. Because there is a maximum of possible blocks, I deepen the network by throwing another convolution layer into the block, thus each block contains two convolution layers. This is the definition of the deepened version of myCNN:

In [87]:
```
class myCNN_improved(nn.Module):
    def __init__(self, num_blocks, input_size, hidden_size, output_size):
        super(myCNN_improved, self).__init__()

        self.conv = nn.Conv2d(input_size, hidden_size, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        blocks = [nn.Sequential(
            nn.Conv2d(hidden_size, hidden_size, kernel_size=3, stride=1, padding=1),
            nn.Conv2d(hidden_size, hidden_size, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        ) for i in range(num_blocks-1)]
        self.blocks = nn.Sequential(*blocks)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.blocks(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

Exploring the effect of hyperparameter-tuning on custom ResNet

For exploring the effect of hyperparameter-tuning on ResNet, I designed a custom ResNet called myResNet. This is done by first defining a residual block, which is composed of two convolution layers (kernel_size=3, stride=1, padding=1) that each followed by a batch normalization layer. A ReLU regularization layer is in between these two convolution layers. A shortcut thar bypass these two convolution layers is designed to pass the identity into later layers. The following code is the definition of the residual block:

In [103]:
```
class ResidualBlock(nn.Module):
    def __init__(self, input_size, output_size, stride=1):
        super(ResidualBlock, self).__init__()

        self.conv1 = nn.Conv2d(input_size, output_size, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(output_size)
        self.relu = nn.ReLU()

        self.conv2 = nn.Conv2d(output_size, output_size, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(output_size)

        self.shortcut = nn.Sequential()
        if stride != 1 or input_size != output_size:
            self.shortcut = nn.Sequential(
                nn.Conv2d(input_size, output_size, kernel_size=1, stride=stride),
                nn.BatchNorm2d(output_size)
            )

    def forward(self, x):
        identity = x

        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))

        identity = self.shortcut(identity)

        out += identity
        out = self.relu(out)

        return out
```

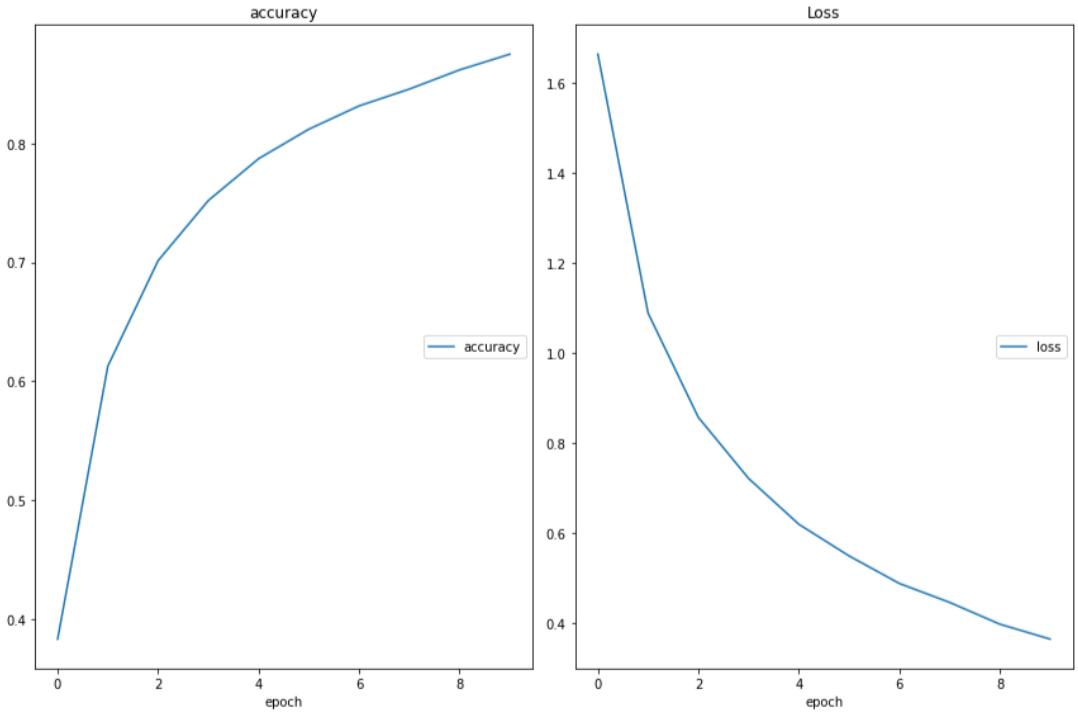
The myResNet network structure is composed of several of the above residual blocks, controlled by a hyperparameter num_blocks. Experiments are done in the same way as on myCNN: a gridsearch on num_blocks is performed to find the best hyperparameter. Two directions goes the exploration on how to improve the performance of myResNet: one is to lengthen the learning procedure, namely increasing the number of epochs, and the other is to deepen the network. As the residual block does not shrink the image size, an unlimited number of residual blocks can be added to the network. Thus, instead of lengthen each residual block, as I did for myCNN, I choose to increase the number of residual blocks to deepen the network. Here is the code of myResNet.

In [104]:
```
class myResNet(nn.Module):
    def __init__(self, num_blocks, input_size, hidden_size, output_size):
        super(myResNet, self).__init__()

        self.conv = nn.Conv2d(input_size, hidden_size, kernel_size=3, stride=1, padding=1)
        self.bn = nn.BatchNorm2d(hidden_size)
        self.relu = nn.ReLU()

        self.blocks = nn.Sequential(
            *[ResidualBlock(hidden_size, hidden_size) for _ in range(num_blocks)]
        )

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        x = self.blocks(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

Result and Discussion

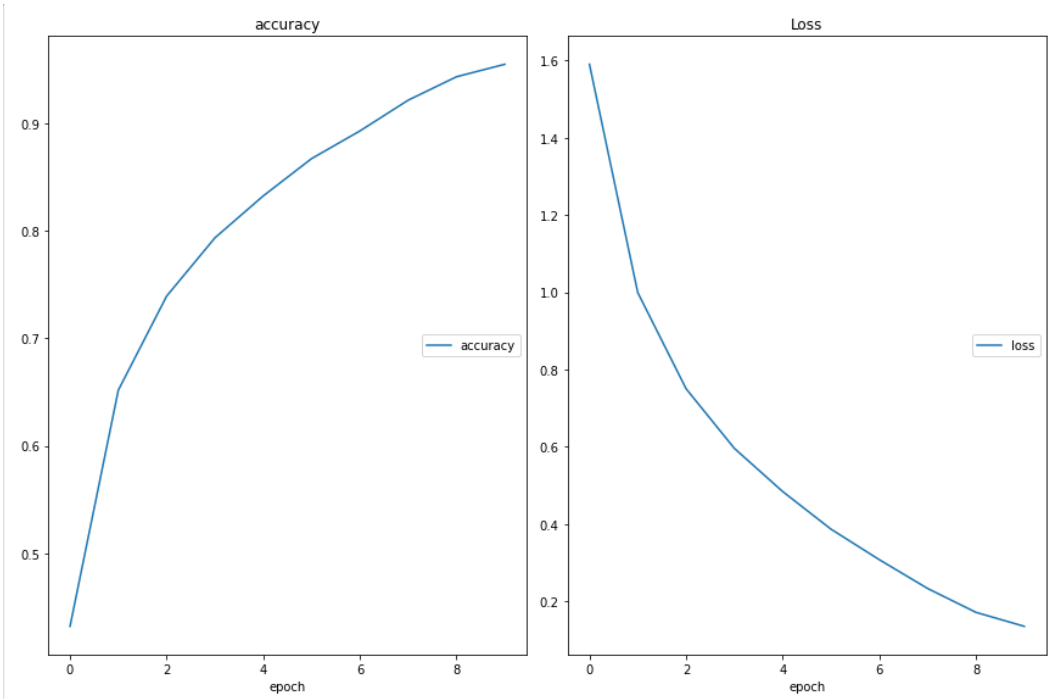
The following graphs are the livelossplot objects for AlexNet and ResNet18.

AlexNet




```
accuracy
        accuracy                (min:    0.383, max:    0.875, cur:    0.875)
Loss
        loss                    (min:    0.364, max:    1.664, cur:    0.364)
```

ResNet




```
accuracy
        accuracy                (min:    0.432, max:    0.955, cur:    0.955)
Loss
        loss                    (min:    0.135, max:    1.590, cur:    0.135)
```

From the graphs we can tell that AlexNet has less accuracy and more loss. It also has less test accuracy than ResNet18. The test accuracy of AlexNet is 78%, while ResNet18 achieves 81%. Generally speaking, from the result of this pre-test, we should expect greater accuracy from ResNet than plain CNN, just as He et al. (2012) have suggested.

For the gridsearch on num_blocks from 1 to 5 for myCNN, the result is as following:

```
[block: 1, epoch: 0] epoch_loss: 2.013 epoch_accuracy: 0.259
[block: 1, epoch: 1] epoch_loss: 1.809 epoch_accuracy: 0.325
[block: 1, epoch: 2] epoch_loss: 1.746 epoch_accuracy: 0.347
[block: 1, epoch: 3] epoch_loss: 1.709 epoch_accuracy: 0.368
[block: 1, epoch: 4] epoch_loss: 1.669 epoch_accuracy: 0.391
[block: 1, epoch: 5] epoch_loss: 1.631 epoch_accuracy: 0.412
[block: 1, epoch: 6] epoch_loss: 1.600 epoch_accuracy: 0.426
[block: 1, epoch: 7] epoch_loss: 1.577 epoch_accuracy: 0.436
[block: 1, epoch: 8] epoch_loss: 1.561 epoch_accuracy: 0.446
[block: 1, epoch: 9] epoch_loss: 1.546 epoch_accuracy: 0.452
[block: 2, epoch: 0] epoch_loss: 1.832 epoch_accuracy: 0.315
[block: 2, epoch: 1] epoch_loss: 1.581 epoch_accuracy: 0.423
[block: 2, epoch: 2] epoch_loss: 1.451 epoch_accuracy: 0.477
[block: 2, epoch: 3] epoch_loss: 1.366 epoch_accuracy: 0.513
[block: 2, epoch: 4] epoch_loss: 1.301 epoch_accuracy: 0.536
[block: 2, epoch: 5] epoch_loss: 1.254 epoch_accuracy: 0.555
[block: 2, epoch: 6] epoch_loss: 1.216 epoch_accuracy: 0.568
[block: 2, epoch: 7] epoch_loss: 1.184 epoch_accuracy: 0.580
[block: 2, epoch: 8] epoch_loss: 1.157 epoch_accuracy: 0.589
[block: 2, epoch: 9] epoch_loss: 1.130 epoch_accuracy: 0.600
[block: 3, epoch: 0] epoch_loss: 1.779 epoch_accuracy: 0.336
[block: 3, epoch: 1] epoch_loss: 1.380 epoch_accuracy: 0.499
[block: 3, epoch: 2] epoch_loss: 1.196 epoch_accuracy: 0.569
[block: 3, epoch: 3] epoch_loss: 1.091 epoch_accuracy: 0.611
[block: 3, epoch: 4] epoch_loss: 1.018 epoch_accuracy: 0.641
[block: 3, epoch: 5] epoch_loss: 0.963 epoch_accuracy: 0.659
[block: 3, epoch: 6] epoch_loss: 0.915 epoch_accuracy: 0.677
[block: 3, epoch: 7] epoch_loss: 0.880 epoch_accuracy: 0.690
[block: 3, epoch: 8] epoch_loss: 0.849 epoch_accuracy: 0.700
[block: 3, epoch: 9] epoch_loss: 0.825 epoch_accuracy: 0.707
[block: 4, epoch: 0] epoch_loss: 1.676 epoch_accuracy: 0.375
[block: 4, epoch: 1] epoch_loss: 1.231 epoch_accuracy: 0.556
[block: 4, epoch: 2] epoch_loss: 1.045 epoch_accuracy: 0.631
[block: 4, epoch: 3] epoch_loss: 0.929 epoch_accuracy: 0.675
[block: 4, epoch: 4] epoch_loss: 0.853 epoch_accuracy: 0.702
[block: 4, epoch: 5] epoch_loss: 0.798 epoch_accuracy: 0.720
[block: 4, epoch: 6] epoch_loss: 0.760 epoch_accuracy: 0.734
[block: 4, epoch: 7] epoch_loss: 0.727 epoch_accuracy: 0.746
[block: 4, epoch: 8] epoch_loss: 0.697 epoch_accuracy: 0.757
[block: 4, epoch: 9] epoch_loss: 0.674 epoch_accuracy: 0.765
[block: 5, epoch: 0] epoch_loss: 1.753 epoch_accuracy: 0.346
[block: 5, epoch: 1] epoch_loss: 1.236 epoch_accuracy: 0.553
[block: 5, epoch: 2] epoch_loss: 1.055 epoch_accuracy: 0.626
[block: 5, epoch: 3] epoch_loss: 0.951 epoch_accuracy: 0.667
[block: 5, epoch: 4] epoch_loss: 0.886 epoch_accuracy: 0.690
[block: 5, epoch: 5] epoch_loss: 0.841 epoch_accuracy: 0.706
[block: 5, epoch: 6] epoch_loss: 0.806 epoch_accuracy: 0.718
[block: 5, epoch: 7] epoch_loss: 0.775 epoch_accuracy: 0.731
[block: 5, epoch: 8] epoch_loss: 0.755 epoch_accuracy: 0.739
[block: 5, epoch: 9] epoch_loss: 0.735 epoch_accuracy: 0.745
```

This suggest the best hyperparameter num_blocks should be 4. It could because that too many pooling layers can over-shrink the image, thus leads to unnecessary loss of the original information.

Different result appears in the case of ResNet, where the best hyperparameter appears to be 5. This probably indicates that there is no significant information loss using residual learning. Also, because the range of the gridsearch is set to be from 1 to 5, it could also indicates that adding more residual blocks beyond num_blocks=5 could lead to a further improvement in its performance. The following is the result of gridsearch for myResNet:

```
[block: 1, epoch: 0] epoch_loss: 1.859 epoch_accuracy: 0.307
[block: 1, epoch: 1] epoch_loss: 1.637 epoch_accuracy: 0.409
[block: 1, epoch: 2] epoch_loss: 1.517 epoch_accuracy: 0.457
[block: 1, epoch: 3] epoch_loss: 1.446 epoch_accuracy: 0.485
[block: 1, epoch: 4] epoch_loss: 1.388 epoch_accuracy: 0.508
[block: 1, epoch: 5] epoch_loss: 1.337 epoch_accuracy: 0.528
[block: 1, epoch: 6] epoch_loss: 1.293 epoch_accuracy: 0.543
[block: 1, epoch: 7] epoch_loss: 1.257 epoch_accuracy: 0.557
[block: 1, epoch: 8] epoch_loss: 1.227 epoch_accuracy: 0.569
[block: 1, epoch: 9] epoch_loss: 1.200 epoch_accuracy: 0.576
[block: 2, epoch: 0] epoch_loss: 1.762 epoch_accuracy: 0.352
[block: 2, epoch: 1] epoch_loss: 1.476 epoch_accuracy: 0.474
[block: 2, epoch: 2] epoch_loss: 1.329 epoch_accuracy: 0.528
[block: 2, epoch: 3] epoch_loss: 1.228 epoch_accuracy: 0.564
[block: 2, epoch: 4] epoch_loss: 1.167 epoch_accuracy: 0.587
[block: 2, epoch: 5] epoch_loss: 1.112 epoch_accuracy: 0.610
[block: 2, epoch: 6] epoch_loss: 1.070 epoch_accuracy: 0.623
[block: 2, epoch: 7] epoch_loss: 1.030 epoch_accuracy: 0.636
[block: 2, epoch: 8] epoch_loss: 0.998 epoch_accuracy: 0.651
[block: 2, epoch: 9] epoch_loss: 0.977 epoch_accuracy: 0.659
[block: 3, epoch: 0] epoch_loss: 1.706 epoch_accuracy: 0.372
[block: 3, epoch: 1] epoch_loss: 1.392 epoch_accuracy: 0.501
[block: 3, epoch: 2] epoch_loss: 1.225 epoch_accuracy: 0.568
[block: 3, epoch: 3] epoch_loss: 1.120 epoch_accuracy: 0.604
[block: 3, epoch: 4] epoch_loss: 1.047 epoch_accuracy: 0.631
[block: 3, epoch: 5] epoch_loss: 0.987 epoch_accuracy: 0.653
[block: 3, epoch: 6] epoch_loss: 0.942 epoch_accuracy: 0.668
[block: 3, epoch: 7] epoch_loss: 0.902 epoch_accuracy: 0.684
[block: 3, epoch: 8] epoch_loss: 0.871 epoch_accuracy: 0.694
[block: 3, epoch: 9] epoch_loss: 0.840 epoch_accuracy: 0.706
[block: 4, epoch: 0] epoch_loss: 1.664 epoch_accuracy: 0.385
[block: 4, epoch: 1] epoch_loss: 1.312 epoch_accuracy: 0.530
[block: 4, epoch: 2] epoch_loss: 1.150 epoch_accuracy: 0.592
[block: 4, epoch: 3] epoch_loss: 1.049 epoch_accuracy: 0.627
[block: 4, epoch: 4] epoch_loss: 0.975 epoch_accuracy: 0.656
[block: 4, epoch: 5] epoch_loss: 0.918 epoch_accuracy: 0.676
[block: 4, epoch: 6] epoch_loss: 0.872 epoch_accuracy: 0.692
[block: 4, epoch: 7] epoch_loss: 0.829 epoch_accuracy: 0.710
[block: 4, epoch: 8] epoch_loss: 0.793 epoch_accuracy: 0.721
[block: 4, epoch: 9] epoch_loss: 0.762 epoch_accuracy: 0.735
[block: 5, epoch: 0] epoch_loss: 1.640 epoch_accuracy: 0.399
[block: 5, epoch: 1] epoch_loss: 1.294 epoch_accuracy: 0.538
[block: 5, epoch: 2] epoch_loss: 1.133 epoch_accuracy: 0.595
[block: 5, epoch: 3] epoch_loss: 1.032 epoch_accuracy: 0.632
[block: 5, epoch: 4] epoch_loss: 0.950 epoch_accuracy: 0.662
[block: 5, epoch: 5] epoch_loss: 0.887 epoch_accuracy: 0.686
[block: 5, epoch: 6] epoch_loss: 0.837 epoch_accuracy: 0.706
[block: 5, epoch: 7] epoch_loss: 0.793 epoch_accuracy: 0.723
[block: 5, epoch: 8] epoch_loss: 0.756 epoch_accuracy: 0.738
[block: 5, epoch: 9] epoch_loss: 0.725 epoch_accuracy: 0.746
```

On improving performance of myCNN

I trained myCNN on both 10 epochs and 20 epochs and record the live training loss and accuracy.

```
[block: 4, epoch: 0] epoch_loss: 1.676 epoch_accuracy: 0.379
[block: 4, epoch: 1] epoch_loss: 1.193 epoch_accuracy: 0.570
[block: 4, epoch: 2] epoch_loss: 1.004 epoch_accuracy: 0.646
[block: 4, epoch: 3] epoch_loss: 0.903 epoch_accuracy: 0.681
[block: 4, epoch: 4] epoch_loss: 0.835 epoch_accuracy: 0.705
[block: 4, epoch: 5] epoch_loss: 0.792 epoch_accuracy: 0.722
[block: 4, epoch: 6] epoch_loss: 0.751 epoch_accuracy: 0.737
[block: 4, epoch: 7] epoch_loss: 0.720 epoch_accuracy: 0.747
[block: 4, epoch: 8] epoch_loss: 0.694 epoch_accuracy: 0.757
[block: 4, epoch: 9] epoch_loss: 0.674 epoch_accuracy: 0.765
[block: 4, epoch: 10] epoch_loss: 0.654 epoch_accuracy: 0.771
[block: 4, epoch: 11] epoch_loss: 0.639 epoch_accuracy: 0.777
[block: 4, epoch: 12] epoch_loss: 0.622 epoch_accuracy: 0.781
[block: 4, epoch: 13] epoch_loss: 0.610 epoch_accuracy: 0.785
[block: 4, epoch: 14] epoch_loss: 0.603 epoch_accuracy: 0.788
[block: 4, epoch: 15] epoch_loss: 0.587 epoch_accuracy: 0.793
[block: 4, epoch: 16] epoch_loss: 0.583 epoch_accuracy: 0.795
[block: 4, epoch: 17] epoch_loss: 0.570 epoch_accuracy: 0.800
[block: 4, epoch: 18] epoch_loss: 0.557 epoch_accuracy: 0.805
[block: 4, epoch: 19] epoch_loss: 0.548 epoch_accuracy: 0.807
```

We can tell that the loss keeps going down and the accuracy keeps going up with more epochs, but it shows some trend of convergence. The test accuracy for the 10epoch model is 71% while it is 72% for the 20epoch model. Although the training accuracy goes up by 4%, the test accuracy only goes up by 1%, and with more epochs, the difference between test accuracy and training accuracy also grows. This is a sign of overfitting to the training dataset. Thus, there is no evidence that lengthen the training procedure will improve the performance of myCNN.

The improved version of myCNN is also trained with 4 blocks on 10 and 20 epochs. The result is very similar to myCNN:

```
[block: 4, epoch: 0] epoch_loss: 1.796 epoch_accuracy: 0.321
[block: 4, epoch: 1] epoch_loss: 1.275 epoch_accuracy: 0.538
[block: 4, epoch: 2] epoch_loss: 1.075 epoch_accuracy: 0.623
[block: 4, epoch: 3] epoch_loss: 0.983 epoch_accuracy: 0.657
[block: 4, epoch: 4] epoch_loss: 0.935 epoch_accuracy: 0.676
[block: 4, epoch: 5] epoch_loss: 0.904 epoch_accuracy: 0.685
[block: 4, epoch: 6] epoch_loss: 0.872 epoch_accuracy: 0.699
[block: 4, epoch: 7] epoch_loss: 0.854 epoch_accuracy: 0.708
[block: 4, epoch: 8] epoch_loss: 0.842 epoch_accuracy: 0.713
[block: 4, epoch: 9] epoch_loss: 0.830 epoch_accuracy: 0.719
[block: 4, epoch: 10] epoch_loss: 0.825 epoch_accuracy: 0.722
[block: 4, epoch: 11] epoch_loss: 0.810 epoch_accuracy: 0.727
[block: 4, epoch: 12] epoch_loss: 0.807 epoch_accuracy: 0.728
[block: 4, epoch: 13] epoch_loss: 0.795 epoch_accuracy: 0.733
[block: 4, epoch: 14] epoch_loss: 0.796 epoch_accuracy: 0.734
[block: 4, epoch: 15] epoch_loss: 0.788 epoch_accuracy: 0.736
[block: 4, epoch: 16] epoch_loss: 0.783 epoch_accuracy: 0.739
[block: 4, epoch: 17] epoch_loss: 0.780 epoch_accuracy: 0.741
[block: 4, epoch: 18] epoch_loss: 0.782 epoch_accuracy: 0.741
[block: 4, epoch: 19] epoch_loss: 0.772 epoch_accuracy: 0.743
```

Both models give a test accuracy of 69%, which is lower than the original version of myCNN. This could because of the fact that no pooling or activation layer is added between the two convolution layers in each block. Overall, there is no evidence that deepen the

network leads to an improvement in learning.

On improving performance for myResNet

I trained myResNet on both 10 epochs and 20 epochs and record the live training loss and accuracy.

```
[block: 5, epoch: 0] epoch_loss: 1.649 epoch_accuracy: 0.396
[block: 5, epoch: 1] epoch_loss: 1.281 epoch_accuracy: 0.541
[block: 5, epoch: 2] epoch_loss: 1.120 epoch_accuracy: 0.598
[block: 5, epoch: 3] epoch_loss: 1.018 epoch_accuracy: 0.638
[block: 5, epoch: 4] epoch_loss: 0.935 epoch_accuracy: 0.670
[block: 5, epoch: 5] epoch_loss: 0.880 epoch_accuracy: 0.689
[block: 5, epoch: 6] epoch_loss: 0.820 epoch_accuracy: 0.711
[block: 5, epoch: 7] epoch_loss: 0.779 epoch_accuracy: 0.727
[block: 5, epoch: 8] epoch_loss: 0.746 epoch_accuracy: 0.740
[block: 5, epoch: 9] epoch_loss: 0.712 epoch_accuracy: 0.754
[block: 5, epoch: 10] epoch_loss: 0.681 epoch_accuracy: 0.765
[block: 5, epoch: 11] epoch_loss: 0.658 epoch_accuracy: 0.772
[block: 5, epoch: 12] epoch_loss: 0.636 epoch_accuracy: 0.779
[block: 5, epoch: 13] epoch_loss: 0.614 epoch_accuracy: 0.788
[block: 5, epoch: 14] epoch_loss: 0.595 epoch_accuracy: 0.795
[block: 5, epoch: 15] epoch_loss: 0.576 epoch_accuracy: 0.799
[block: 5, epoch: 16] epoch_loss: 0.559 epoch_accuracy: 0.806
[block: 5, epoch: 17] epoch_loss: 0.547 epoch_accuracy: 0.810
[block: 5, epoch: 18] epoch_loss: 0.527 epoch_accuracy: 0.816
[block: 5, epoch: 19] epoch_loss: 0.515 epoch_accuracy: 0.821
```

The two models of 10 epochs and 20 epochs gives test accuracy of 73% and 76%, which suggested that lengthen the training procedure is a possible way to improve performance of myResNet.

The improved version of myResNet has its num_block hyperparameter set to 10, and it is also trained with 10 and 20 epochs.

```
[block: 10, epoch: 0] epoch_loss: 1.601 epoch_accuracy: 0.410
[block: 10, epoch: 1] epoch_loss: 1.181 epoch_accuracy: 0.578
[block: 10, epoch: 2] epoch_loss: 1.000 epoch_accuracy: 0.647
[block: 10, epoch: 3] epoch_loss: 0.882 epoch_accuracy: 0.690
[block: 10, epoch: 4] epoch_loss: 0.800 epoch_accuracy: 0.720
[block: 10, epoch: 5] epoch_loss: 0.736 epoch_accuracy: 0.745
[block: 10, epoch: 6] epoch_loss: 0.678 epoch_accuracy: 0.766
[block: 10, epoch: 7] epoch_loss: 0.640 epoch_accuracy: 0.779
[block: 10, epoch: 8] epoch_loss: 0.597 epoch_accuracy: 0.794
[block: 10, epoch: 9] epoch_loss: 0.563 epoch_accuracy: 0.806
[block: 10, epoch: 10] epoch_loss: 0.537 epoch_accuracy: 0.814
[block: 10, epoch: 11] epoch_loss: 0.511 epoch_accuracy: 0.825
[block: 10, epoch: 12] epoch_loss: 0.491 epoch_accuracy: 0.830
[block: 10, epoch: 13] epoch_loss: 0.466 epoch_accuracy: 0.841
[block: 10, epoch: 14] epoch_loss: 0.447 epoch_accuracy: 0.848
[block: 10, epoch: 15] epoch_loss: 0.429 epoch_accuracy: 0.852
[block: 10, epoch: 16] epoch_loss: 0.411 epoch_accuracy: 0.859
[block: 10, epoch: 17] epoch_loss: 0.396 epoch_accuracy: 0.863
[block: 10, epoch: 18] epoch_loss: 0.383 epoch_accuracy: 0.868
[block: 10, epoch: 19] epoch_loss: 0.365 epoch_accuracy: 0.875
```

The two models give test accuracies of 78% and 81%. Same as myResNet with 5 blocks, a significant increase in test accuracy is found with the increasing number of epochs. Also, the 10-block myResNet has significantly better performance than the 5-block myResNet at both 10 and 20 epochs. This suggested that deepen the network is a possible way to improve

performance of myResNet.

Conclusion

As shown in the example of AlexNet and ResNet18, ResNet has an overall better ability at image classification task on CIFAR-10 dataset than plain CNN. Other results have suggested that lengthen learning procedure and deepen the network could help improve the performance of ResNet, but no such evidence was found for CNN.

Limitation

There are significant limitations to this project. Due to the time-consuming training process, I cannot test on other designs of CNN or ResNet, neither could I make more comparisons or further explore the effect on hyperparameter-tuning (for example, to check when does test accuracy goes down for myResNet with respect to number of epochs). Further studies could be done to test my results on more samples, as well as explore other designs of CNN and compare their performance.

Reference

1. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2015). *Deep Residual Learning For Image Recognition*.
2. Tjmoon0104's GitHub repo, pytorch-tiny-imagenet: tjmoon0104/pytorch-tiny-imagenet: pytorch-tiny-imagenet (github.com)
3. ChatGPT: https://chat.openai.com/