Youshaa Murhij, BMSTU, Coding report1

Understanding the original dataset:

The original one batch data is (10000 x 3072) matrix expressed in numpy array. The number of columns, (10000), indicates the number of sample data. As stated in the CIFAR-10/CIFAR-100 dataset, the row vector, (3072) represents an color image of 32x32 pixels.

Regenerating the results of the paper for CIFAR10 Dataset:

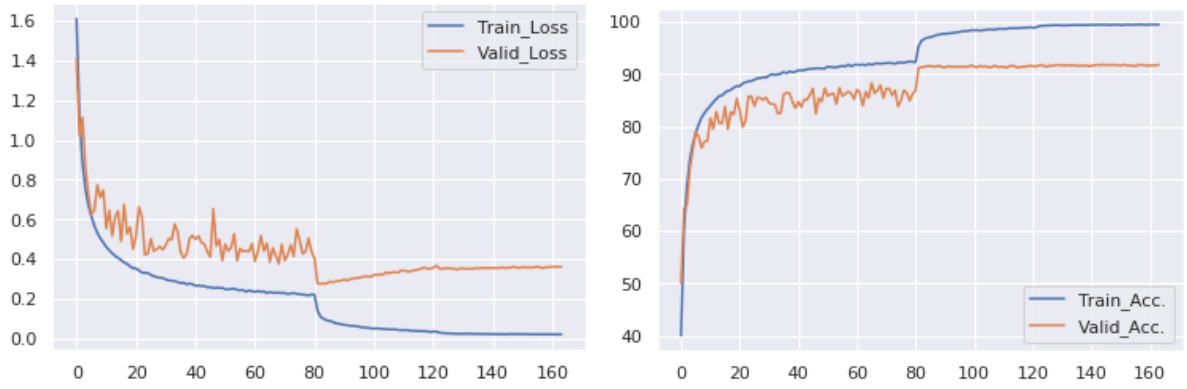
RESNET 20: 164 Epochs

Epoch: [164 | 164] LR: 0.001000

Processing (391/391) Data: 0.015s | Batch: 0.069s | Total: 0:00:27 | ETA: 0:00:01 | Loss: 0.0209 | top1: 99.3900 | top5: 99.9980

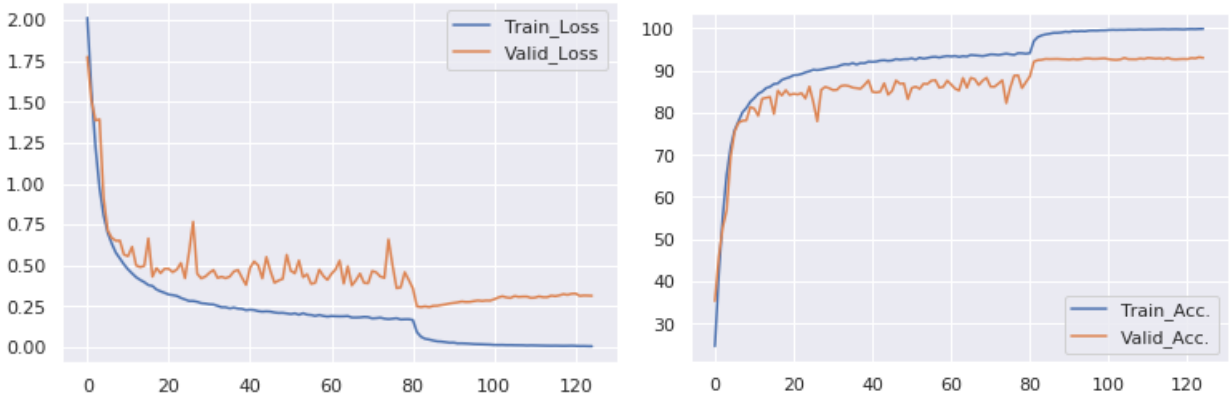
Processing (100/100) Data: 0.010s | Batch: 0.032s | Total: 0:00:03 | ETA: 0:00:01 | Loss: 0.3605 | top1: 91.7600 | top5: 99.7000

Best acc: tensor (91.8100, device='cuda:0')



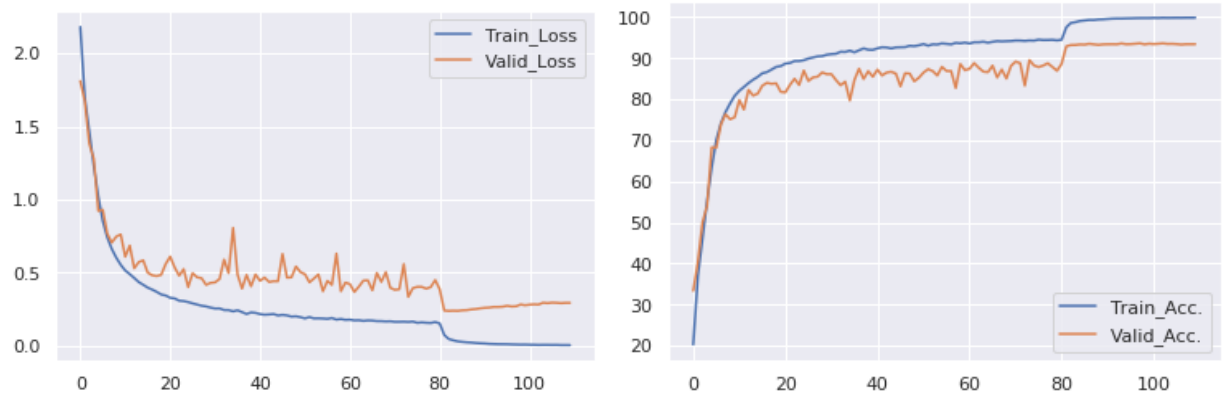
RESNET 56: 125 Epochs

Epoch: [125 | 125] LR: 0.001000
Processing (391/391) Data: 0.064s | Batch: 0.161s | Total: 0:01:03 | ETA: 0:00:01 | Loss: 0.0061 | top1: 99.8740 | top5: 100.0000
Processing (100/100) Data: 0.012s | Batch: 0.046s | Total: 0:00:04 | ETA: 0:00:01 | Loss: 0.3145 | top1: 93.0100 | top5: 99.7900
Best acc: tensor (93.1600, device='cuda:0')



RESNET 110: 110 Epochs

Epoch: [110 | 110] LR: 0.010000
Processing (391/391) Data: 0.005s | Batch: 0.314s | Total: 0:02:02 | ETA: 0:00:01 | Loss: 0.0059 | top1: 99.8540 | top5: 100.0000
Processing (100/100) Data: 0.031s | Batch: 0.083s | Total: 0:00:08 | ETA: 0:00:01 | Loss: 0.2947 | top1: 93.4000 | top5: 99.8300
Best acc: tensor (93.6100, device='cuda:0')



While Training my model I noticed that we may get different results when training our models with different random seed.

For CIFAR10:

In case of 20 layers and setting the number of epochs equal to 164 as in the base-code and without changing other hyper parameters, I got the same accuracy in both training and testing as the one indicated in the paper.

In case of 56 layers, the training takes more time the previous one and for a number of epochs equal to 125, the best accuracy I got was 93.210% for the testing data and 99.5760% for the training data

In the case of 110 layers, it took certainly more and more time to converge and for 110 epochs without changing anything else I got 99.6040% accuracy for training data and 93.6100% for the testing data

So, I can conclude that this deep net needs a sufficient number of epochs to converge and the results that I got are good enough and almost similar to the ones in the original paper and the small difference is due to the different number of epochs.

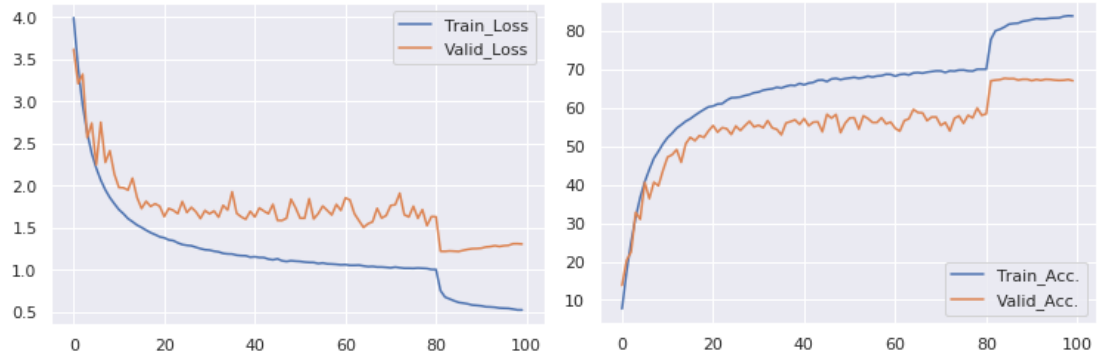
The most important notice that we should consider is that the needs about 80 epochs to converge and after that the weights, accuracy and the performance become stable and change slowly in case of additional epochs

Regenerating the results of the paper for CIFAR100 DataSet:

RESNET 20: 100 Epochs

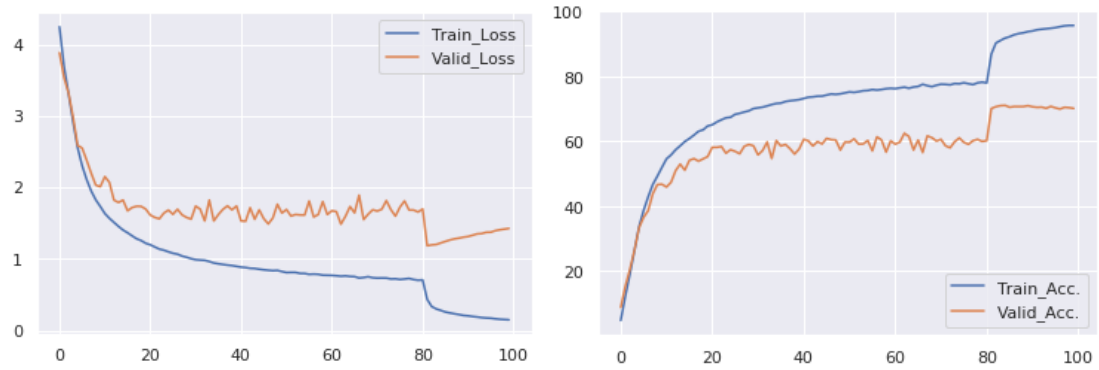
Epoch: [100 | 100] LR: 0.010000
Processing (391/391) Data: 0.023s | Batch: 0.063s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.5211 | top1: 83.8820 | top5: 97.8800

Processing (100/100) Data: 0.009s | Batch: 0.030s | Total: 0:00:03 | ETA: 0:00:01 | Loss: 1.3038 | top1: 67.0400 | top5: 90.3900
Best acc: tensor (67.6700, device='cuda:0')



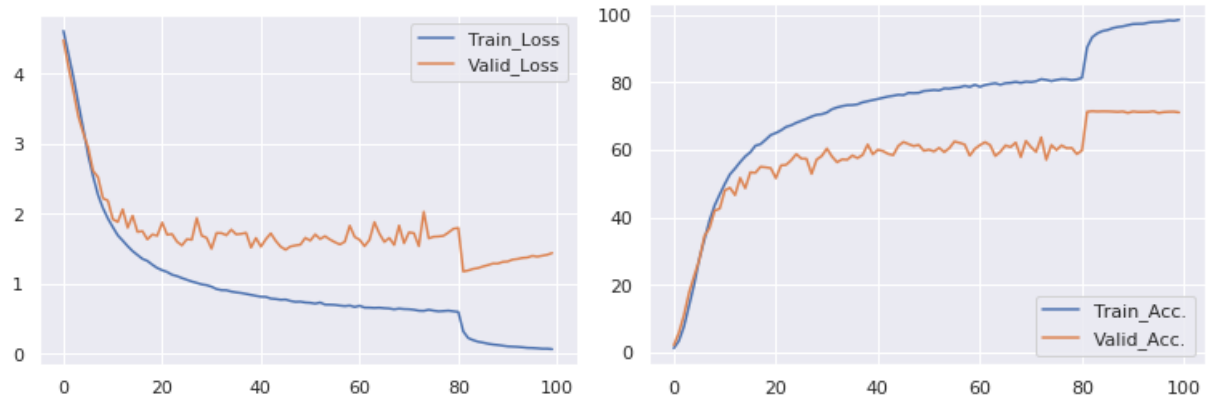
RESNET 56: 100 Epochs

Epoch: [100 | 100] LR: 0.010000
Processing (391/391) Data: 0.077s | Batch: 0.162s | Total: 0:01:03 | ETA: 0:00:01 | Loss: 0.1473 | top1: 95.6540 | top5: 99.8180
Processing (100/100) Data: 0.017s | Batch: 0.042s | Total: 0:00:04 | ETA: 0:00:01 | Loss: 1.4247 | top1: 70.0800 | top5: 91.2400
Best acc: tensor (71., device='cuda:0')



RESNET 110: 100 Epochs

Epoch: [100 | 100] LR: 0.010000
Processing (391/391) Data: 0.005s | Batch: 0.305s | Total: 0:01:59 | ETA: 0:00:01 | Loss: 0.0671 | top1: 98.4240 | top5: 99.9700
Processing (100/100) Data: 0.030s | Batch: 0.083s | Total: 0:00:08 | ETA: 0:00:01 | Loss: 1.4395 | top1: 71.0600 | top5: 91.7900
Best acc: tensor (71.4200, device='cuda:0')

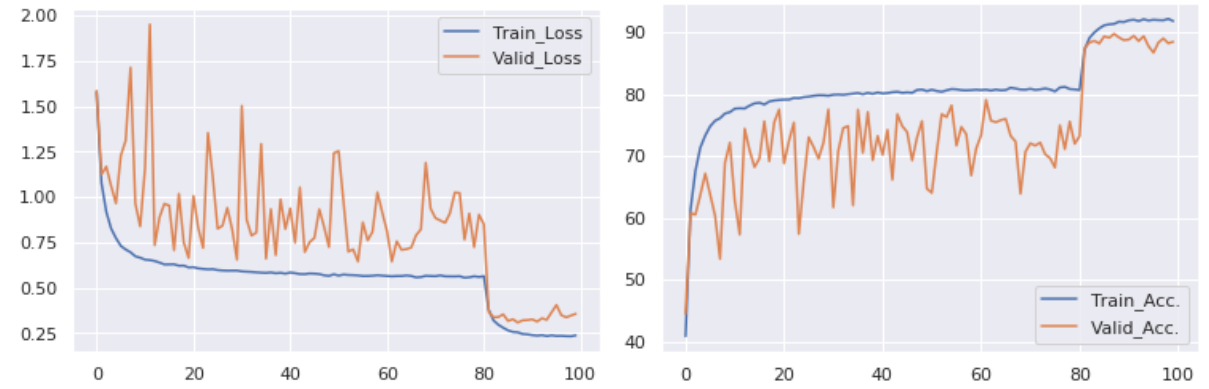


For CIFAR100 we get similar results to the ones we got on CIFAR10 expect for that the gap between the Training acc. And the Validation became bigger and this happens because we increased the number of classes to 100 which leads to a bigger error margin.

Optional task 1:

Learning Rate: 10 * Initial Learning Rate (1.e-4)

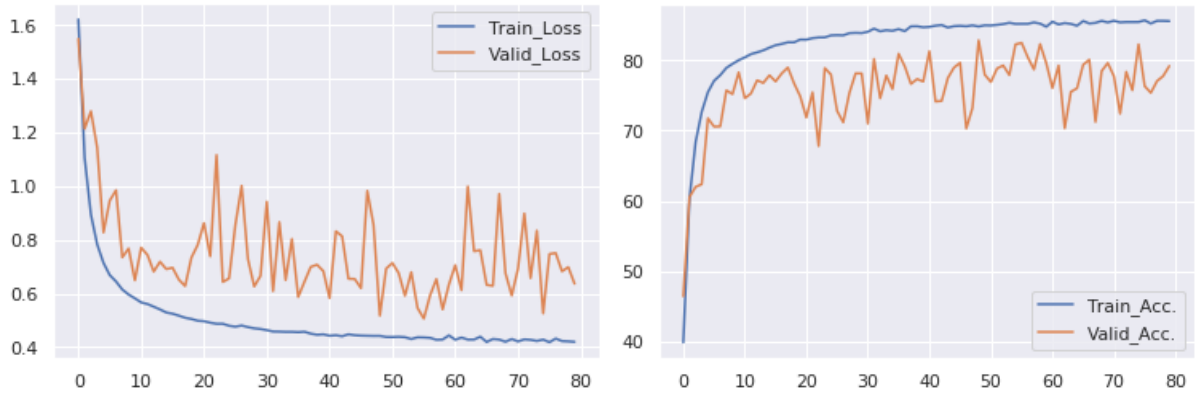
Epoch: [100 | 100] LR: 0.010000
Processing (391/391) Data: 0.020s | Batch: 0.062s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.2369 | top1: 91.8020 | top5: 99.8460
Processing (100/100) Data: 0.010s | Batch: 0.028s | Total: 0:00:02 | ETA: 0:00:01 | Loss: 0.3560 | top1: 88.4500 | top5: 99.5700
Best acc: tensor (89.7200, device='cuda:0')



As we can see from the figures above, increasing the learning rate by 10 times will lead to increase the validation error which means we need enough epochs to fix this issue. And as a result, we got a lower validation accuracy 89.7200.

Learning Rate: 5 * Initial Learning Rate (1.e-4)

Epoch: [80 | 80] LR: 0.100000
Processing (391/391) Data: 0.018s | Batch: 0.063s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.4200 | top1: 85.5440 | top5: 99.4540
Processing (100/100) Data: 0.009s | Batch: 0.028s | Total: 0:00:02 | ETA: 0:00:01 | Loss: 0.6364 | top1: 79.1600 | top5: 98.9100
Best acc: tensor (82.8200, device='cuda:0')



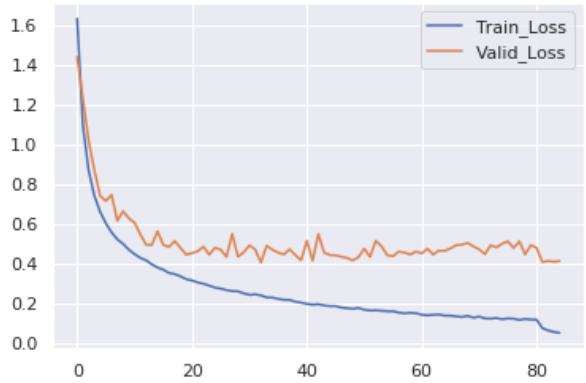
Learning Rate: 2 * Initial Learning Rate (1.e-4)

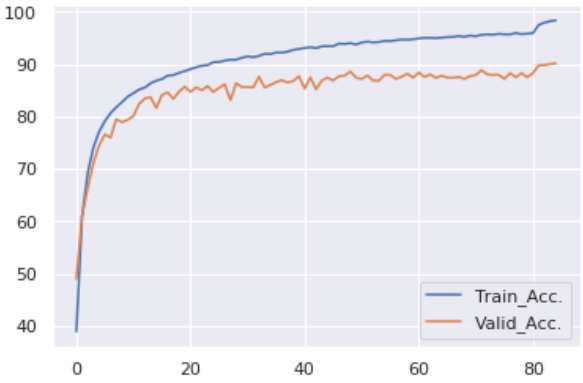
Epoch: [80 | 80] LR: 0.100000
Processing (391/391) Data: 0.019s | Batch: 0.063s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.2966 | top1: 89.7520 | top5: 99.7340
Processing (100/100) Data: 0.009s | Batch: 0.029s | Total: 0:00:02 | ETA: 0:00:01 | Loss: 0.5221 | top1: 83.3100 | top5: 99.2800
Best acc: tensor (86.7800, device='cuda:0')

Learning Rate: 1/10 * Initial Learning Rate (1.e-4)

Epoch: [85 | 85] LR: 0.010000
Processing (391/391) Data: 0.020s | Batch: 0.063s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.0488 | top1: 98.3680 | top5: 99.9980
Processing (100/100) Data: 0.009s | Batch: 0.030s | Total: 0:00:02 | ETA: 0:00:01 | Loss: 0.4103 | top1: 90.1200 | top5: 99.7200
Best acc: tensor (90.1200, device='cuda:0')

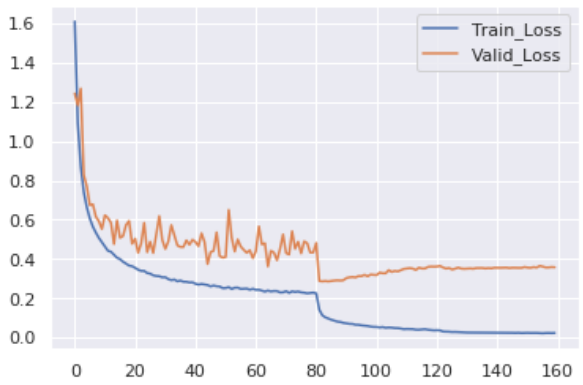
As seen in the figures below, decreasing the learning rate led to stable learning rate but the gap between training and a validation LOSS stays the same of worth.

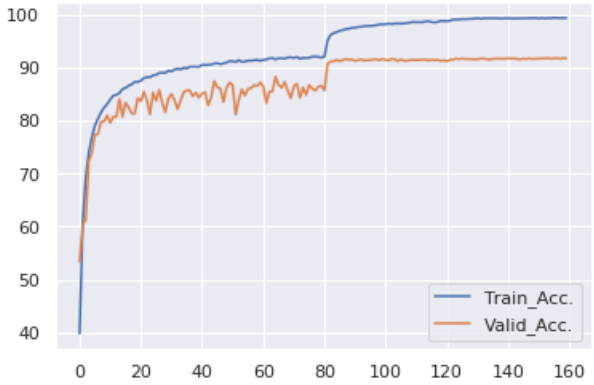




Increasing the number of the epochs to 160:

Epoch: [160 | 160] LR: 0.001000
Processing (391/391) Data: 0.021s | Batch: 0.063s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.0228 | top1: 99.3800 | top5: 99.9980
Processing (100/100) Data: 0.008s | Batch: 0.029s | Total: 0:00:02 | ETA: 0:00:01 | Loss: 0.3576 | top1: 91.7600 | top5: 99.8000
Best acc: tensor (91.8200, device='cuda:0')





Increasing the number of epochs to a high value will lead to overfitting problem as it is obvious from the graph that the accuracy is not improving over more and more epochs

Decreasing the number of the epochs to 40:

Epoch: [40 | 40] LR: 0.100000
Processing (391/391) Data: 0.023s | Batch: 0.062s | Total: 0:00:24 | ETA: 0:00:01 | Loss: 0.2716 | top1: 90.4940 | top5: 99.8020
Processing (100/100) Data: 0.009s | Batch: 0.028s | Total: 0:00:02 | ETA: 0:00:01 | Loss: 0.4902 | top1: 85.0600 | top5: 99.2800
Best acc: tensor (86.0700, device='cuda:0')

Meanwhile, decreasing the number of epochs (in our case, 40) led to a bigger error range and a drop in the accuracy to 85%

ReadME DOC - Youshaa Murhij - Coding Homework 1

Required Task 1

First I imported required libraries for printing and displaying results as Figures

```
%matplotlib inline
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

Here I installed the lastest version of Pytorsh Platform

```
!pip install -q torch==1.3.0 torchvision
```

```
%cd /content/drive/My\ Drive/
```

```
import torch
print(torch.__version__)
```

```
1.3.0+cu100
```

After that, we need to download the base line code for this paper from github

```
!git clone --recursive https://github.com/chenhuims/pytorch-classification.git
```

```
%cd /content/drive/My Drive/pytorch-classification/
```

Here I called the train function on resnet and set the depth to 20 with 164 epochs and saved the re

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --depth 20 --epochs 164 --schedule 81 122 --gamma 0.1 --wd 1
```

Required Libraries to display the graphs in a good way

```
import seaborn as sns
sns.set()
```

Displaying the Loss and the Accuracy

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10/resnet-20/
```

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10/resnet-20/
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

Displaying the accuracy figure:

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

Here I called the train function on resnet and set the depth to 56 with 125 epochs and saved the re

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --depth 56 --epochs 125 --schedule 81 122 --gamma 0.1 --wd 1
```

```
Displaying the Loss and the Accuracy
```

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10/resnet-56/
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

Here I called the train function on CIFAR10 and set the depth to 110 with 110 epochs and saved th

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --depth 110 --epochs 110 --schedule 81 122 --gamma 0.1 --wd
```

Here I called the train function on cifar100 and set the depth to 20 with 100 epochs and saved the

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar100 --depth 20 --epochs 100 --schedule 81 122
```

Here I called the train function on cifar100 and set the depth to 56 with 100 epochs and saved the

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar100 --depth 56 --epochs 100 --schedule 81 122
```

Displaying the Loss and the Accuracy

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar100/resnet-56/
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

Here I called the train function on cifar100 and set the depth to 110 with 100 epochs and saved th

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar100 --depth 110 --epochs 100 --schedule 81 12
```

Optional Task 1

First, we multiply the base learning rate by 10 and check for changes in the results

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar10 --depth 20 --epochs 100 --schedule 81 122
```

Displaying the Loss and the Accuracy

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10_optional/resnet-2
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

then, we multiply the base learning rate by 5 and check for changes in the results

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar10 --depth 20 --epochs 80 --schedule 81 122 -
```

Displaying the Loss and the Accuracy

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10_optional/resnet-2
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

```
1
```

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

we multiply the base learning rate by 2

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar10 --depth 20 --epochs 80 --schedule 81 122 -
```

we multiply the base learning rate by 1/10

```
1
```

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar10 --depth 20 --epochs 85 --schedule 81 122 -
```

Displaying the Loss and the Accuracy

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10_optional/resnet-2
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

we double the number of training epochs

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar10 --depth 20 --epochs 160 --schedule 81 122
```

we half the number of training epochs

```
%cd /content/drive/My Drive/pytorch-classification/
!python cifar.py -a resnet --dataset cifar10 --depth 20 --epochs 40 --schedule 81 122 -
```

Displaying the Loss and the Accuracy

```
%cd /content/drive/My Drive/pytorch-classification/checkpoint/cifar10_optional/resnet-2
file = pd.read_csv('log.txt',delimiter='  ')
plt.plot(file['Train_Loss'])
plt.plot(file['Valid_Loss'])
plt.legend(['Train_Loss','Valid_Loss'],loc='upper right');
plt.show()
```

```
plt.plot(file['Train_Acc.'])
plt.plot(file['Valid_Acc.'])
plt.legend(['Train_Acc.','Valid_Acc.'],loc='lower right');
plt.show()
```

Training script for CIFAR-10/100

```


from __future__ import print_function

import argparse
import os
import shutil
import time
import random

import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data as data
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import models.cifar as models

from utils import Bar, Logger, AverageMeter, accuracy, mkdir_p, savefig


model_names = sorted(name for name in models.__dict__
    if name.islower() and not name.startswith("__")
    and callable(models.__dict__[name]))

parser = argparse.ArgumentParser(description='PyTorch CIFAR10/100 Training')
```

```
# Datasets
parser.add_argument('-d', '--dataset', default='cifar10', type=str)
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
# Optimization options
parser.add_argument('--epochs', default=300, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('--train-batch', default=128, type=int, metavar='N',
                    help='train batchsize')
parser.add_argument('--test-batch', default=100, type=int, metavar='N',
                    help='test batchsize')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate')
parser.add_argument('--drop', '--dropout', default=0, type=float,
                    metavar='Dropout', help='Dropout ratio')
parser.add_argument('--schedule', type=int, nargs='+', default=[150, 225],
                        help='Decrease learning rate at these epochs.')
parser.add_argument('--gamma', type=float, default=0.1, help='LR is multiplied by gamma
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--weight-decay', '--wd', default=5e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)')
# Checkpoints
parser.add_argument('-c', '--checkpoint', default='checkpoint', type=str, metavar='PATH
                    help='path to save checkpoint (default: checkpoint)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
# Architecture
parser.add_argument('--arch', '-a', metavar='ARCH', default='resnet20',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('--depth', type=int, default=29, help='Model depth.')
parser.add_argument('--block-name', type=str, default='BasicBlock',
                    help='the building block for Resnet and Preresnet: BasicBlock, Bott
parser.add_argument('--cardinality', type=int, default=8, help='Model cardinality (grou
parser.add_argument('--widen-factor', type=int, default=4, help='Widen factor. 4 -> 64,
parser.add_argument('--growthRate', type=int, default=12, help='Growth rate for DenseNe
parser.add_argument('--compressionRate', type=int, default=2, help='Compression Rate (t
# Miscs
parser.add_argument('--manualSeed', type=int, help='manual seed')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
#Device options
parser.add_argument('--gpu-id', default='0', type=str,
                    help='id(s) for CUDA_VISIBLE_DEVICES')

args = parser.parse_args()
state = {k: v for k, v in args._get_kwargs()}

# Validate dataset
assert args.dataset == 'cifar10' or args.dataset == 'cifar100', 'Dataset can only be ci
```

```
84
85 # Use CUDA
86 os.environ['CUDA_VISIBLE_DEVICES'] = args.gpu_id
87 use_cuda = torch.cuda.is_available()
88
89 # Random seed
90 if args.manualSeed is None:
91     args.manualSeed = random.randint(1, 10000)
92 random.seed(args.manualSeed)
93 torch.manual_seed(args.manualSeed)
94 if use_cuda:
95     torch.cuda.manual_seed_all(args.manualSeed)
96
97 best_acc = 0  # best test accuracy
98
99 def main():
100     global best_acc
101     start_epoch = args.start_epoch  # start from epoch 0 or last checkpoint epoch
102
103     if not os.path.isdir(args.checkpoint):
104         mkdir_p(args.checkpoint)
105
106
107
108     # Data
109     print('==> Preparing dataset %s' % args.dataset)
110     transform_train = transforms.Compose([
111         transforms.RandomCrop(32, padding=4),
112         transforms.RandomHorizontalFlip(),
113         transforms.ToTensor(),
114         transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
115     ])
116
117     transform_test = transforms.Compose([
118         transforms.ToTensor(),
119         transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
120     ])
121     if args.dataset == 'cifar10':
122         dataloader = datasets.CIFAR10
123         num_classes = 10
124     else:
125         dataloader = datasets.CIFAR100
126         num_classes = 100
127
128
129     trainset = dataloader(root='./data', train=True, download=True, transform=transform
130     trainloader = data.DataLoader(trainset, batch_size=args.train_batch, shuffle=True,
131
132     testset = dataloader(root='./data', train=False, download=False, transform=transfor
133     testloader = data.DataLoader(testset, batch_size=args.test_batch, shuffle=False, nu
134
135     # Model
136     print("==> creating model '{}'".format(args.arch))
137     if args.arch.startswith('resnext'):
138         model = models.__dict__[args.arch](
```

```
                    cardinality=args.cardinality,
                    num_classes=num_classes,
                    depth=args.depth,
                    widen_factor=args.widen_factor,
                    dropRate=args.drop,
                )
    elif args.arch.startswith('densenet'):
        model = models.__dict__[args.arch](
                    num_classes=num_classes,
                    depth=args.depth,
                    growthRate=args.growthRate,
                    compressionRate=args.compressionRate,
                    dropRate=args.drop,
                )
    elif args.arch.startswith('wrn'):
        model = models.__dict__[args.arch](
                    num_classes=num_classes,
                    depth=args.depth,
                    widen_factor=args.widen_factor,
                    dropRate=args.drop,
                )
    elif args.arch.endswith('resnet'):
        model = models.__dict__[args.arch](
                    num_classes=num_classes,
                    depth=args.depth,
                    block_name=args.block_name,
                )
    else:
        model = models.__dict__[args.arch](num_classes=num_classes)

    model = torch.nn.DataParallel(model).cuda()
    cudnn.benchmark = True
    print('    Total params: %.2fM' % (sum(p.numel() for p in model.parameters())/10000
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum, weigh

    # Resume
    title = 'cifar-10-' + args.arch
    if args.resume:
        # Load checkpoint.
        print('==> Resuming from checkpoint..')
        assert os.path.isfile(args.resume), 'Error: no checkpoint directory found!'
        args.checkpoint = os.path.dirname(args.resume)
        checkpoint = torch.load(args.resume)
        best_acc = checkpoint['best_acc']
        start_epoch = checkpoint['epoch']
        model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer'])
        logger = Logger(os.path.join(args.checkpoint, 'log.txt'), title=title, resume=T
    else:
        logger = Logger(os.path.join(args.checkpoint, 'log.txt'), title=title)
        logger.set_names(['Learning_Rate', 'Train_Loss', 'Valid_Loss', 'Train_Acc.', 'V


    if args.evaluate:
```

```
        print('\nEvaluation only')
        test_loss, test_acc = test(testloader, model, criterion, start_epoch, use_cuda)
        print(' Test Loss:  %.8f, Test Acc:  %.2f' % (test_loss, test_acc))
        return

    # Train and val
    for epoch in range(start_epoch, args.epochs):
        adjust_learning_rate(optimizer, epoch)

        print('\nEpoch: [%d | %d] LR: %f' % (epoch + 1, args.epochs, state['lr']))

        train_loss, train_acc = train(trainloader, model, criterion, optimizer, epoch,
        test_loss, test_acc = test(testloader, model, criterion, epoch, use_cuda)

        # append logger file
        logger.append([state['lr'], train_loss, test_loss, train_acc, test_acc])

        # save model
        is_best = test_acc > best_acc
        best_acc = max(test_acc, best_acc)
        save_checkpoint({
                'epoch': epoch + 1,
                'state_dict': model.state_dict(),
                'acc': test_acc,
                'best_acc': best_acc,
                'optimizer' : optimizer.state_dict(),
            }, is_best, checkpoint=args.checkpoint)

    logger.close()
    logger.plot()
    savefig(os.path.join(args.checkpoint, 'log.eps'))

    print('Best acc:')
    print(best_acc)

def train(trainloader, model, criterion, optimizer, epoch, use_cuda):
    # switch to train mode
    model.train()

    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()
    top5 = AverageMeter()
    end = time.time()

    bar = Bar('Processing', max=len(trainloader))
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        # measure data loading time
        data_time.update(time.time() - end)

        if use_cuda:
            inputs, targets = inputs.cuda(), targets.cuda(async=True)
        inputs, targets = torch.autograd.Variable(inputs), torch.autograd.Variable(targ

```

```
        # compute output
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # measure accuracy and record loss
        prec1, prec5 = accuracy(outputs.data, targets.data, topk=(1, 5))
        if float(torch.__version__[:3]) < 0.5:
            losses.update(loss.data[0], inputs.size(0))
            top1.update(prec1[0], inputs.size(0))
            top5.update(prec5[0], inputs.size(0))
        else:
            losses.update(loss.data, inputs.size(0))
            top1.update(prec1, inputs.size(0))
            top5.update(prec5, inputs.size(0))

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()

        # plot progress
        bar.suffix  = '({batch}/{size}) Data: {data:.3f}s | Batch: {bt:.3f}s | Total: {
                    batch=batch_idx + 1,
                    size=len(trainloader),
                    data=data_time.avg,
                    bt=batch_time.avg,
                    total=bar.elapsed_td,
                    eta=bar.eta_td,
                    loss=losses.avg,
                    top1=top1.avg,
                    top5=top5.avg,
                    )
        bar.next()
    bar.finish()
    return (losses.avg, top1.avg)

def test(testloader, model, criterion, epoch, use_cuda):
    global best_acc

    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()
    top5 = AverageMeter()

    # switch to evaluate mode
    model.eval()

    end = time.time()
    bar = Bar('Processing', max=len(testloader))
    for batch_idx, (inputs, targets) in enumerate(testloader):
```

```
        # measure data loading time
        data_time.update(time.time() - end)

        if use_cuda:
            inputs, targets = inputs.cuda(), targets.cuda()
        inputs, targets = torch.autograd.Variable(inputs, volatile=True), torch.autogra

        # compute output
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # measure accuracy and record loss
        prec1, prec5 = accuracy(outputs.data, targets.data, topk=(1, 5))
        if float(torch.__version__[:3]) < 0.5:
            losses.update(loss.data[0], inputs.size(0))
            top1.update(prec1[0], inputs.size(0))
            top5.update(prec5[0], inputs.size(0))
        else:
            losses.update(loss.data, inputs.size(0))
            top1.update(prec1, inputs.size(0))
            top5.update(prec5, inputs.size(0))

        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()

        # plot progress
        bar.suffix  = '({batch}/{size}) Data: {data:.3f}s | Batch: {bt:.3f}s | Total: {
                    batch=batch_idx + 1,
                    size=len(testloader),
                    data=data_time.avg,
                    bt=batch_time.avg,
                    total=bar.elapsed_td,
                    eta=bar.eta_td,
                    loss=losses.avg,
                    top1=top1.avg,
                    top5=top5.avg,
                    )
        bar.next()
    bar.finish()
    return (losses.avg, top1.avg)

def save_checkpoint(state, is_best, checkpoint='checkpoint', filename='checkpoint.pth.t
    filepath = os.path.join(checkpoint, filename)
    torch.save(state, filepath)
    if is_best:
        shutil.copyfile(filepath, os.path.join(checkpoint, 'model_best.pth.tar'))

def adjust_learning_rate(optimizer, epoch):
    global state
    if epoch in args.schedule:
        state['lr'] *= args.gamma
        for param_group in optimizer.param_groups:
            param_group['lr'] = state['lr']

if __name__ == '__main__':
```

```
359 if __name__ == "__main__":
360     main()
```