rhyupy94v

September 4, 2025

[2]:
```
import torch
import torch.nn as nn

from torchvision import transforms, datasets
from torch.utils.data import DataLoader, random_split

MEAN = [0.5071, 0.4867, 0.4408] #mean of CIFAR-100 of RGB
STD  = [0.2675, 0.2565, 0.2761] #std of CIFAR-100 of RGB

train_tf = transforms.Compose([
    transforms.RandomCrop(32,padding=4), #encounter slight displacement
    transforms.RandomHorizontalFlip(), #Mirror
    transforms.ToTensor(),
    transforms.Normalize(MEAN,STD),
    transforms.RandomErasing(p=0.25, scale=(0.02,0.2), ratio=(0.3,3.3),␣
 ↪value='random'), #Cutout some figure with probablity = 0.25
])

test_tf = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(MEAN, STD),
])

data = datasets.CIFAR100(root='./data', train=True, download=True,␣
 ↪transform=train_tf)

n_val = int(0.1*len(data)) #validation data 10%

train_subset, val_subset = random_split(data,[len(data) - n_val, n_val])

train_base = datasets.CIFAR100(root='./data', train=True, download=True,␣
 ↪transform=train_tf)
val_base = datasets.CIFAR100(root='./data', train=True, download=True,␣
 ↪transform=test_tf)

train_set = torch.utils.data.Subset(train_base, train_subset.indices)
val_set = torch.utils.data.Subset(val_base, val_subset.indices)
```

```
train_loader = DataLoader(train_set, batch_size=128, shuffle=True,␣
 ↪num_workers=4, pin_memory=True)
val_loader = DataLoader(val_set,   batch_size=128, shuffle=False,␣
 ↪num_workers=4, pin_memory=True)
test_loader = DataLoader(datasets.CIFAR100(root='./data', train=False,␣
 ↪download=True, transform=test_tf),
                         batch_size=128, shuffle=False, num_workers=4,␣
 ↪pin_memory=True)
```

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

[3]:
```
import torch
import torch.nn as nn

class BasicBlock(nn.Module):
    def __init__(self, in_c, out_c, stride = 1, downsample = None):
        #in_c : input channels
        #out_c : output channels
        expansion = 1
        super().__init__()
        self.conv1 = nn.Conv2d(in_c,out_c,␣
 ↪kernel_size=3,stride=stride,padding=1,bias=False) #3*3
        self.bn1   = nn.BatchNorm2d(out_c)
        self.relu  = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_c,out_c, kernel_size=3, stride=1,padding=1,␣
 ↪bias=False)
        self.bn2   = nn.BatchNorm2d(out_c)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        if self.downsample is not None:
            identity = self.downsample(identity)
        x = self.relu(x+identity)
        return x


def make_layer_basic(in_c, out_c, blocks, first_stride):
```

```
    layers = []
    down = None
    if first_stride != 1 or in_c != out_c: #downsample or changing channels
        down = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=1, stride=first_stride,␣
 ↪bias=False),
            nn.BatchNorm2d(out_c),
        )
    layers.append(BasicBlock(in_c, out_c, stride=first_stride, downsample=down))
    for _ in range(1, blocks):
        layers.append(BasicBlock(out_c, out_c, stride=1))
    return nn.Sequential(*layers)
```

[4]:
```
import torch
import torch.nn as nn
class ResNetCIFAR(nn.Module):
    def __init__(self, num_classes=100, drop_out = 0.1):
        super().__init__()
        # conv1 3×3,32,s=1,p=1 + BN + ReLU + Dropout
        self.stem = nn.Sequential(
            nn.Conv2d(3,32,kernel_size=3,stride=1,padding=1,bias=False),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.Dropout(drop_out),
        )
        # conv2_x: ×2
        self.layer2 = make_layer_basic(in_c=32,  out_c=32,  blocks=2,␣
 ↪first_stride=1)
        # conv3_x: ×4, stride=2
        self.layer3 = make_layer_basic(in_c=32,  out_c=64,  blocks=4,␣
 ↪first_stride=2)
        # conv4_x: ×4, stride=2
        self.layer4 = make_layer_basic(in_c=64,  out_c=128, blocks=4,␣
 ↪first_stride=2)
        # conv5_x: ×2, stride=2
        self.layer5 = make_layer_basic(in_c=128, out_c=256, blocks=2,␣
 ↪first_stride=2)

        self.pool = nn.AdaptiveMaxPool2d(1)
        self.fc   = nn.Linear(256, num_classes)

    def forward(self, x, return_feats=False):
        feats = {}
        x = self.stem(x);   feats['conv1']   = x
        x = self.layer2(x); feats['conv2_x'] = x
        x = self.layer3(x); feats['conv3_x'] = x
        x = self.layer4(x); feats['conv4_x'] = x
```

```
        x = self.layer5(x); feats['conv5_x'] = x
        x = self.pool(x).flatten(1)          # N×256
        logits = self.fc(x)                  # N×100
        return (logits, feats) if return_feats else logits
```

[5]:
```
def evaluate_top1(model, loader, loss_fun, device):
    model.eval()
    loss_sum, correct, total = 0.0, 0, 0
    for x, y in loader:
        x, y = x.to(device), y.to(device)
        logits = model(x)
        loss = loss_fun(logits, y)
        b = y.size(0)
        loss_sum += loss.item() * b
        correct += (logits.argmax(1) == y).sum().item()
        total += b
    avg_loss = loss_sum / total
    top1_acc = 100.0 * correct / total
    return avg_loss, top1_acc
```

[7]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.optim as optim
from torch.optim.lr_scheduler import CosineAnnealingLR


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = ResNetCIFAR(num_classes=100, drop_out=0.1).to(device)

loss_fun = nn.CrossEntropyLoss()

num_epochs = 200
#lr=0.2
optimizer = optim.SGD(model.parameters(), lr=0.2, momentum=0.9,␣
 ↪weight_decay=5e-4)
scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs, eta_min=1e-5)

accuracy_train_list, accuracy_val_list = [], []
loss_train_list, loss_val_list = [], []

for epoch in range(num_epochs):
    # ---------- train ----------
    model.train()
    running_loss,correct, total = 0.0, 0, 0
```

```
    for x_batch, y_batch in train_loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        logits = model(x_batch)
        loss = loss_fun(logits, y_batch)

        loss.backward()
        optimizer.step()

        b = y_batch.size(0)
        running_loss += loss.item() * b
        pred = logits.argmax(dim=1)
        correct += (pred == y_batch).sum().item()
        total += b

    train_loss = running_loss / total
    train_acc  = 100.0 * correct / total



    # ---------- val ----------
    model.eval()
    running_loss_val,correct_val, total_val = 0.0, 0, 0
    with torch.no_grad():
        for x_val, y_val in val_loader:
            x_val = x_val.to(device)
            y_val = y_val.to(device)
            out = model(x_val)
            loss_t = loss_fun(out, y_val)

            b = y_val.size(0)
            running_loss_val += loss_t.item() * b
            pred_val = out.argmax(dim=1)
            correct_val += (pred_val== y_val).sum().item()
            total_val += b

    val_loss = running_loss_val / total_val
    val_acc  = 100.0 * correct_val / total_val

    loss_train_list.append(train_loss)
    loss_val_list.append(val_loss)
    accuracy_train_list.append(train_acc)
    accuracy_val_list.append(val_acc)

    scheduler.step()
```

```
    print(f"lr = {optimizer.param_groups[0]['lr']:.6f}")
    print(f"Epoch {epoch+1:02d}/{num_epochs} | "
          f"train_loss={train_loss:.4f} acc={train_acc:.2f}%  |  "
          f"val_loss={val_loss:.4f} acc={val_acc:.2f}%")



print("Training Complete")
```

```
lr = 0.199988
Epoch 01/200 | train_loss=4.8278 acc=1.41%  |  val_loss=4.4535 acc=2.44%
lr = 0.199951
Epoch 02/200 | train_loss=4.3766 acc=2.56%  |  val_loss=4.3500 acc=2.84%
lr = 0.199889
Epoch 03/200 | train_loss=4.1931 acc=4.48%  |  val_loss=4.0953 acc=5.26%
lr = 0.199803
Epoch 04/200 | train_loss=4.0233 acc=6.79%  |  val_loss=3.9528 acc=8.54%
lr = 0.199692
Epoch 05/200 | train_loss=3.8679 acc=9.37%  |  val_loss=3.8571 acc=9.26%
lr = 0.199556
Epoch 06/200 | train_loss=3.7290 acc=11.79%  |  val_loss=3.7840 acc=11.28%
lr = 0.199396
Epoch 07/200 | train_loss=3.6421 acc=13.37%  |  val_loss=3.6932 acc=13.58%
lr = 0.199212
Epoch 08/200 | train_loss=3.5256 acc=15.70%  |  val_loss=3.4595 acc=15.84%
lr = 0.199002
Epoch 09/200 | train_loss=3.4116 acc=17.58%  |  val_loss=3.6273 acc=15.56%
lr = 0.198769
Epoch 10/200 | train_loss=3.3193 acc=19.24%  |  val_loss=3.3193 acc=19.32%
lr = 0.198511
Epoch 11/200 | train_loss=3.1927 acc=21.38%  |  val_loss=3.2369 acc=20.08%
lr = 0.198229
Epoch 12/200 | train_loss=3.0616 acc=23.74%  |  val_loss=3.3425 acc=19.66%
lr = 0.197922
Epoch 13/200 | train_loss=2.9558 acc=25.68%  |  val_loss=3.0355 acc=26.10%
lr = 0.197592
Epoch 14/200 | train_loss=2.8553 acc=27.75%  |  val_loss=3.1436 acc=24.20%
lr = 0.197237
Epoch 15/200 | train_loss=2.7622 acc=29.38%  |  val_loss=3.3544 acc=20.90%
lr = 0.196858
Epoch 16/200 | train_loss=2.6816 acc=31.20%  |  val_loss=2.8406 acc=28.12%
lr = 0.196456
Epoch 17/200 | train_loss=2.6083 acc=32.65%  |  val_loss=2.6114 acc=32.48%
lr = 0.196030
Epoch 18/200 | train_loss=2.5363 acc=33.80%  |  val_loss=2.6780 acc=33.64%
lr = 0.195580
Epoch 19/200 | train_loss=2.4667 acc=35.57%  |  val_loss=3.0344 acc=27.44%
```

```
lr = 0.195106
Epoch 20/200 | train_loss=2.4181 acc=36.69%  |  val_loss=2.7413 acc=31.38%
lr = 0.194609
Epoch 21/200 | train_loss=2.3636 acc=37.84%  |  val_loss=2.6301 acc=35.02%
lr = 0.194088
Epoch 22/200 | train_loss=2.3173 acc=38.98%  |  val_loss=2.5602 acc=35.82%
lr = 0.193545
Epoch 23/200 | train_loss=2.2730 acc=39.50%  |  val_loss=2.5405 acc=35.70%
lr = 0.192978
Epoch 24/200 | train_loss=2.2512 acc=40.16%  |  val_loss=2.4368 acc=38.28%
lr = 0.192388
Epoch 25/200 | train_loss=2.2317 acc=40.70%  |  val_loss=2.7946 acc=34.38%
lr = 0.191776
Epoch 26/200 | train_loss=2.1942 acc=41.68%  |  val_loss=2.4706 acc=35.94%
lr = 0.191141
Epoch 27/200 | train_loss=2.1680 acc=42.39%  |  val_loss=2.4935 acc=36.44%
lr = 0.190483
Epoch 28/200 | train_loss=2.1575 acc=42.51%  |  val_loss=2.5884 acc=34.40%
lr = 0.189803
Epoch 29/200 | train_loss=2.1501 acc=42.58%  |  val_loss=2.5946 acc=35.38%
lr = 0.189101
Epoch 30/200 | train_loss=2.1132 acc=42.90%  |  val_loss=2.4345 acc=37.06%
lr = 0.188377
Epoch 31/200 | train_loss=2.1119 acc=43.54%  |  val_loss=2.8352 acc=32.36%
lr = 0.187631
Epoch 32/200 | train_loss=2.0968 acc=43.59%  |  val_loss=2.3125 acc=39.36%
lr = 0.186864
Epoch 33/200 | train_loss=2.0689 acc=44.36%  |  val_loss=2.7709 acc=35.06%
lr = 0.186075
Epoch 34/200 | train_loss=2.0724 acc=44.42%  |  val_loss=2.6000 acc=36.38%
lr = 0.185265
Epoch 35/200 | train_loss=2.0470 acc=44.80%  |  val_loss=2.3914 acc=40.00%
lr = 0.184434
Epoch 36/200 | train_loss=2.0392 acc=44.86%  |  val_loss=2.8542 acc=35.10%
lr = 0.183582
Epoch 37/200 | train_loss=2.0295 acc=45.41%  |  val_loss=2.3531 acc=39.92%
lr = 0.182709
Epoch 38/200 | train_loss=2.0173 acc=45.56%  |  val_loss=2.6870 acc=35.82%
lr = 0.181816
Epoch 39/200 | train_loss=2.0048 acc=45.85%  |  val_loss=2.3790 acc=38.82%
lr = 0.180903
Epoch 40/200 | train_loss=2.0019 acc=45.95%  |  val_loss=2.4396 acc=38.66%
lr = 0.179969
Epoch 41/200 | train_loss=1.9876 acc=46.19%  |  val_loss=2.2157 acc=41.84%
lr = 0.179017
Epoch 42/200 | train_loss=1.9931 acc=46.24%  |  val_loss=2.1408 acc=43.38%
lr = 0.178044
Epoch 43/200 | train_loss=1.9754 acc=46.57%  |  val_loss=2.4054 acc=39.16%
```

```
lr = 0.177052
Epoch 44/200 | train_loss=1.9667 acc=46.85%  |  val_loss=2.5886 acc=36.52%
lr = 0.176042
Epoch 45/200 | train_loss=1.9468 acc=47.27%  |  val_loss=2.3804 acc=39.12%
lr = 0.175012
Epoch 46/200 | train_loss=1.9505 acc=47.13%  |  val_loss=2.5347 acc=38.54%
lr = 0.173964
Epoch 47/200 | train_loss=1.9428 acc=47.39%  |  val_loss=2.1543 acc=43.16%
lr = 0.172898
Epoch 48/200 | train_loss=1.9237 acc=47.83%  |  val_loss=2.5599 acc=36.82%
lr = 0.171814
Epoch 49/200 | train_loss=1.9180 acc=47.86%  |  val_loss=2.1251 acc=43.92%
lr = 0.170712
Epoch 50/200 | train_loss=1.8974 acc=48.31%  |  val_loss=2.2821 acc=40.94%
lr = 0.169593
Epoch 51/200 | train_loss=1.9150 acc=47.99%  |  val_loss=2.0679 acc=44.90%
lr = 0.168456
Epoch 52/200 | train_loss=1.8838 acc=48.47%  |  val_loss=2.2884 acc=40.78%
lr = 0.167303
Epoch 53/200 | train_loss=1.8940 acc=47.97%  |  val_loss=2.3341 acc=40.86%
lr = 0.166133
Epoch 54/200 | train_loss=1.8773 acc=48.79%  |  val_loss=2.2009 acc=42.72%
lr = 0.164947
Epoch 55/200 | train_loss=1.8676 acc=48.83%  |  val_loss=2.1644 acc=43.56%
lr = 0.163744
Epoch 56/200 | train_loss=1.8517 acc=49.28%  |  val_loss=2.2900 acc=41.26%
lr = 0.162526
Epoch 57/200 | train_loss=1.8461 acc=49.51%  |  val_loss=3.0428 acc=33.82%
lr = 0.161293
Epoch 58/200 | train_loss=1.8440 acc=49.49%  |  val_loss=2.3530 acc=41.86%
lr = 0.160044
Epoch 59/200 | train_loss=1.8379 acc=49.81%  |  val_loss=2.2883 acc=41.48%
lr = 0.158781
Epoch 60/200 | train_loss=1.8224 acc=49.94%  |  val_loss=2.5578 acc=36.92%
lr = 0.157503
Epoch 61/200 | train_loss=1.8100 acc=50.12%  |  val_loss=2.2849 acc=41.32%
lr = 0.156211
Epoch 62/200 | train_loss=1.8145 acc=50.45%  |  val_loss=2.4448 acc=39.06%
lr = 0.154905
Epoch 63/200 | train_loss=1.8011 acc=50.50%  |  val_loss=2.4777 acc=39.46%
lr = 0.153585
Epoch 64/200 | train_loss=1.7989 acc=50.56%  |  val_loss=2.1649 acc=44.34%
lr = 0.152252
Epoch 65/200 | train_loss=1.7761 acc=51.04%  |  val_loss=2.1049 acc=44.58%
lr = 0.150907
Epoch 66/200 | train_loss=1.7783 acc=50.99%  |  val_loss=2.2577 acc=42.56%
lr = 0.149548
Epoch 67/200 | train_loss=1.7786 acc=50.84%  |  val_loss=2.4747 acc=38.84%
```

```
lr = 0.148178
Epoch 68/200 | train_loss=1.7657 acc=51.52%  |  val_loss=3.0954 acc=32.20%
lr = 0.146796
Epoch 69/200 | train_loss=1.7512 acc=51.83%  |  val_loss=1.9110 acc=48.82%
lr = 0.145402
Epoch 70/200 | train_loss=1.7337 acc=52.13%  |  val_loss=2.0369 acc=46.30%
lr = 0.143997
Epoch 71/200 | train_loss=1.7449 acc=51.87%  |  val_loss=2.1622 acc=45.34%
lr = 0.142581
Epoch 72/200 | train_loss=1.7292 acc=52.37%  |  val_loss=2.0985 acc=45.32%
lr = 0.141154
Epoch 73/200 | train_loss=1.7138 acc=52.52%  |  val_loss=2.0665 acc=45.52%
lr = 0.139718
Epoch 74/200 | train_loss=1.7073 acc=52.86%  |  val_loss=2.2827 acc=41.88%
lr = 0.138271
Epoch 75/200 | train_loss=1.7065 acc=52.74%  |  val_loss=1.9680 acc=47.80%
lr = 0.136816
Epoch 76/200 | train_loss=1.6894 acc=53.20%  |  val_loss=2.4179 acc=39.40%
lr = 0.135351
Epoch 77/200 | train_loss=1.6830 acc=53.48%  |  val_loss=2.0777 acc=45.62%
lr = 0.133877
Epoch 78/200 | train_loss=1.6838 acc=53.46%  |  val_loss=2.0356 acc=46.34%
lr = 0.132395
Epoch 79/200 | train_loss=1.6748 acc=53.50%  |  val_loss=2.1217 acc=44.68%
lr = 0.130905
Epoch 80/200 | train_loss=1.6577 acc=53.87%  |  val_loss=1.9603 acc=48.50%
lr = 0.129408
Epoch 81/200 | train_loss=1.6573 acc=53.99%  |  val_loss=2.2161 acc=43.88%
lr = 0.127903
Epoch 82/200 | train_loss=1.6522 acc=53.96%  |  val_loss=2.1795 acc=44.76%
lr = 0.126391
Epoch 83/200 | train_loss=1.6282 acc=54.73%  |  val_loss=2.1958 acc=44.50%
lr = 0.124873
Epoch 84/200 | train_loss=1.6170 acc=55.00%  |  val_loss=1.9359 acc=49.04%
lr = 0.123348
Epoch 85/200 | train_loss=1.6199 acc=54.79%  |  val_loss=2.1403 acc=45.56%
lr = 0.121818
Epoch 86/200 | train_loss=1.6081 acc=55.32%  |  val_loss=1.9187 acc=49.14%
lr = 0.120283
Epoch 87/200 | train_loss=1.6105 acc=55.29%  |  val_loss=1.8811 acc=49.96%
lr = 0.118742
Epoch 88/200 | train_loss=1.6008 acc=55.30%  |  val_loss=2.1694 acc=46.78%
lr = 0.117197
Epoch 89/200 | train_loss=1.5700 acc=56.08%  |  val_loss=1.7950 acc=51.36%
lr = 0.115648
Epoch 90/200 | train_loss=1.5754 acc=55.66%  |  val_loss=1.9510 acc=48.56%
lr = 0.114094
Epoch 91/200 | train_loss=1.5601 acc=56.27%  |  val_loss=2.0876 acc=47.04%
```

```
lr = 0.112538
Epoch 92/200 | train_loss=1.5447 acc=56.44%  |  val_loss=1.9881 acc=47.62%
lr = 0.110978
Epoch 93/200 | train_loss=1.5412 acc=56.53%  |  val_loss=2.0299 acc=47.12%
lr = 0.109415
Epoch 94/200 | train_loss=1.5262 acc=57.29%  |  val_loss=1.8269 acc=50.40%
lr = 0.107851
Epoch 95/200 | train_loss=1.5220 acc=57.33%  |  val_loss=1.8473 acc=50.28%
lr = 0.106284
Epoch 96/200 | train_loss=1.5086 acc=57.42%  |  val_loss=1.9863 acc=48.94%
lr = 0.104715
Epoch 97/200 | train_loss=1.4981 acc=57.76%  |  val_loss=1.9569 acc=49.24%
lr = 0.103146
Epoch 98/200 | train_loss=1.4832 acc=58.16%  |  val_loss=1.8246 acc=50.70%
lr = 0.101576
Epoch 99/200 | train_loss=1.4789 acc=58.18%  |  val_loss=1.9169 acc=50.92%
lr = 0.100005
Epoch 100/200 | train_loss=1.4682 acc=58.51%  |  val_loss=1.7219 acc=53.60%
lr = 0.098434
Epoch 101/200 | train_loss=1.4583 acc=58.73%  |  val_loss=1.8680 acc=52.32%
lr = 0.096864
Epoch 102/200 | train_loss=1.4462 acc=58.92%  |  val_loss=1.8010 acc=52.04%
lr = 0.095295
Epoch 103/200 | train_loss=1.4356 acc=59.32%  |  val_loss=1.8025 acc=53.08%
lr = 0.093726
Epoch 104/200 | train_loss=1.4215 acc=59.64%  |  val_loss=1.8475 acc=51.78%
lr = 0.092159
Epoch 105/200 | train_loss=1.4143 acc=59.49%  |  val_loss=2.0018 acc=47.94%
lr = 0.090595
Epoch 106/200 | train_loss=1.3989 acc=60.20%  |  val_loss=1.7155 acc=54.06%
lr = 0.089032
Epoch 107/200 | train_loss=1.3798 acc=60.64%  |  val_loss=1.9255 acc=50.24%
lr = 0.087472
Epoch 108/200 | train_loss=1.3814 acc=60.64%  |  val_loss=1.6574 acc=55.58%
lr = 0.085916
Epoch 109/200 | train_loss=1.3598 acc=61.21%  |  val_loss=1.9598 acc=49.94%
lr = 0.084362
Epoch 110/200 | train_loss=1.3439 acc=61.50%  |  val_loss=1.8197 acc=52.34%
lr = 0.082813
Epoch 111/200 | train_loss=1.3456 acc=61.51%  |  val_loss=1.9047 acc=51.64%
lr = 0.081268
Epoch 112/200 | train_loss=1.3309 acc=61.99%  |  val_loss=1.7362 acc=54.46%
lr = 0.079727
Epoch 113/200 | train_loss=1.3160 acc=62.11%  |  val_loss=1.7543 acc=53.94%
lr = 0.078192
Epoch 114/200 | train_loss=1.3069 acc=62.63%  |  val_loss=1.8285 acc=51.18%
lr = 0.076662
Epoch 115/200 | train_loss=1.2961 acc=63.00%  |  val_loss=1.6763 acc=54.92%
```

```
lr = 0.075137
Epoch 116/200 | train_loss=1.2807 acc=63.10%  |  val_loss=1.7396 acc=54.52%
lr = 0.073619
Epoch 117/200 | train_loss=1.2748 acc=63.11%  |  val_loss=1.9115 acc=51.14%
lr = 0.072107
Epoch 118/200 | train_loss=1.2574 acc=63.45%  |  val_loss=1.7399 acc=54.38%
lr = 0.070602
Epoch 119/200 | train_loss=1.2301 acc=64.25%  |  val_loss=1.5367 acc=57.02%
lr = 0.069105
Epoch 120/200 | train_loss=1.2304 acc=64.46%  |  val_loss=1.5285 acc=57.74%
lr = 0.067615
Epoch 121/200 | train_loss=1.2176 acc=64.81%  |  val_loss=1.5939 acc=57.80%
lr = 0.066133
Epoch 122/200 | train_loss=1.2004 acc=65.15%  |  val_loss=1.5954 acc=57.46%
lr = 0.064659
Epoch 123/200 | train_loss=1.1858 acc=65.50%  |  val_loss=1.6737 acc=56.12%
lr = 0.063194
Epoch 124/200 | train_loss=1.1702 acc=65.91%  |  val_loss=1.7898 acc=53.76%
lr = 0.061739
Epoch 125/200 | train_loss=1.1656 acc=66.10%  |  val_loss=1.6088 acc=55.72%
lr = 0.060292
Epoch 126/200 | train_loss=1.1450 acc=66.75%  |  val_loss=1.6245 acc=56.72%
lr = 0.058856
Epoch 127/200 | train_loss=1.1372 acc=66.81%  |  val_loss=1.6460 acc=56.92%
lr = 0.057429
Epoch 128/200 | train_loss=1.1207 acc=67.37%  |  val_loss=1.5844 acc=58.20%
lr = 0.056013
Epoch 129/200 | train_loss=1.1041 acc=67.79%  |  val_loss=1.6662 acc=56.30%
lr = 0.054608
Epoch 130/200 | train_loss=1.0876 acc=68.11%  |  val_loss=1.5839 acc=58.40%
lr = 0.053214
Epoch 131/200 | train_loss=1.0781 acc=68.23%  |  val_loss=1.5074 acc=60.76%
lr = 0.051832
Epoch 132/200 | train_loss=1.0692 acc=68.49%  |  val_loss=1.6893 acc=55.76%
lr = 0.050462
Epoch 133/200 | train_loss=1.0430 acc=69.39%  |  val_loss=1.4781 acc=60.38%
lr = 0.049103
Epoch 134/200 | train_loss=1.0368 acc=69.51%  |  val_loss=1.5283 acc=58.90%
lr = 0.047758
Epoch 135/200 | train_loss=1.0050 acc=70.37%  |  val_loss=1.4441 acc=61.58%
lr = 0.046425
Epoch 136/200 | train_loss=0.9927 acc=70.56%  |  val_loss=1.4941 acc=59.94%
lr = 0.045105
Epoch 137/200 | train_loss=0.9845 acc=70.62%  |  val_loss=1.5838 acc=58.52%
lr = 0.043799
Epoch 138/200 | train_loss=0.9707 acc=71.12%  |  val_loss=1.4003 acc=61.34%
lr = 0.042507
Epoch 139/200 | train_loss=0.9415 acc=71.82%  |  val_loss=1.4555 acc=61.48%
```

```
lr = 0.041229
Epoch 140/200 | train_loss=0.9453 acc=71.85%  |  val_loss=1.3575 acc=62.66%
lr = 0.039966
Epoch 141/200 | train_loss=0.9204 acc=72.24%  |  val_loss=1.4620 acc=61.68%
lr = 0.038717
Epoch 142/200 | train_loss=0.8987 acc=72.95%  |  val_loss=1.5489 acc=59.80%
lr = 0.037484
Epoch 143/200 | train_loss=0.8826 acc=73.34%  |  val_loss=1.4045 acc=62.70%
lr = 0.036266
Epoch 144/200 | train_loss=0.8665 acc=73.87%  |  val_loss=1.4397 acc=61.70%
lr = 0.035063
Epoch 145/200 | train_loss=0.8570 acc=74.20%  |  val_loss=1.5799 acc=59.48%
lr = 0.033877
Epoch 146/200 | train_loss=0.8337 acc=75.14%  |  val_loss=1.4167 acc=62.64%
lr = 0.032707
Epoch 147/200 | train_loss=0.8169 acc=75.04%  |  val_loss=1.3970 acc=63.08%
lr = 0.031554
Epoch 148/200 | train_loss=0.7921 acc=75.94%  |  val_loss=1.4253 acc=63.16%
lr = 0.030417
Epoch 149/200 | train_loss=0.7781 acc=76.16%  |  val_loss=1.4085 acc=63.98%
lr = 0.029298
Epoch 150/200 | train_loss=0.7572 acc=76.98%  |  val_loss=1.3283 acc=64.68%
lr = 0.028196
Epoch 151/200 | train_loss=0.7342 acc=77.60%  |  val_loss=1.4978 acc=62.00%
lr = 0.027112
Epoch 152/200 | train_loss=0.7160 acc=78.18%  |  val_loss=1.3281 acc=64.54%
lr = 0.026046
Epoch 153/200 | train_loss=0.7035 acc=78.38%  |  val_loss=1.3922 acc=64.40%
lr = 0.024998
Epoch 154/200 | train_loss=0.6814 acc=78.98%  |  val_loss=1.3895 acc=63.94%
lr = 0.023968
Epoch 155/200 | train_loss=0.6529 acc=79.75%  |  val_loss=1.4290 acc=64.18%
lr = 0.022958
Epoch 156/200 | train_loss=0.6407 acc=80.28%  |  val_loss=1.4080 acc=64.24%
lr = 0.021966
Epoch 157/200 | train_loss=0.6111 acc=81.10%  |  val_loss=1.3413 acc=65.92%
lr = 0.020993
Epoch 158/200 | train_loss=0.6068 acc=81.26%  |  val_loss=1.4183 acc=64.24%
lr = 0.020041
Epoch 159/200 | train_loss=0.5781 acc=82.11%  |  val_loss=1.3172 acc=66.58%
lr = 0.019107
Epoch 160/200 | train_loss=0.5580 acc=82.65%  |  val_loss=1.3092 acc=66.88%
lr = 0.018194
Epoch 161/200 | train_loss=0.5225 acc=83.74%  |  val_loss=1.3796 acc=65.42%
lr = 0.017301
Epoch 162/200 | train_loss=0.5110 acc=83.85%  |  val_loss=1.3049 acc=67.24%
lr = 0.016428
Epoch 163/200 | train_loss=0.4889 acc=84.73%  |  val_loss=1.3825 acc=66.24%
```

```
lr = 0.015576
Epoch 164/200 | train_loss=0.4722 acc=85.16%  |  val_loss=1.3326 acc=66.88%
lr = 0.014745
Epoch 165/200 | train_loss=0.4485 acc=86.01%  |  val_loss=1.3684 acc=66.32%
lr = 0.013935
Epoch 166/200 | train_loss=0.4228 acc=86.78%  |  val_loss=1.2805 acc=68.68%
lr = 0.013146
Epoch 167/200 | train_loss=0.4025 acc=87.44%  |  val_loss=1.3122 acc=67.92%
lr = 0.012379
Epoch 168/200 | train_loss=0.3802 acc=88.12%  |  val_loss=1.3211 acc=67.62%
lr = 0.011633
Epoch 169/200 | train_loss=0.3558 acc=88.86%  |  val_loss=1.3222 acc=67.80%
lr = 0.010909
Epoch 170/200 | train_loss=0.3402 acc=89.42%  |  val_loss=1.2752 acc=68.52%
lr = 0.010207
Epoch 171/200 | train_loss=0.3124 acc=90.40%  |  val_loss=1.2999 acc=69.32%
lr = 0.009527
Epoch 172/200 | train_loss=0.2976 acc=90.80%  |  val_loss=1.3030 acc=69.34%
lr = 0.008869
Epoch 173/200 | train_loss=0.2759 acc=91.67%  |  val_loss=1.2371 acc=70.78%
lr = 0.008234
Epoch 174/200 | train_loss=0.2611 acc=92.05%  |  val_loss=1.2411 acc=70.08%
lr = 0.007622
Epoch 175/200 | train_loss=0.2500 acc=92.34%  |  val_loss=1.2858 acc=70.14%
lr = 0.007032
Epoch 176/200 | train_loss=0.2279 acc=93.21%  |  val_loss=1.2381 acc=70.98%
lr = 0.006465
Epoch 177/200 | train_loss=0.2193 acc=93.59%  |  val_loss=1.2171 acc=70.62%
lr = 0.005922
Epoch 178/200 | train_loss=0.2002 acc=94.18%  |  val_loss=1.2174 acc=71.38%
lr = 0.005401
Epoch 179/200 | train_loss=0.1866 acc=94.51%  |  val_loss=1.2112 acc=71.40%
lr = 0.004904
Epoch 180/200 | train_loss=0.1728 acc=95.13%  |  val_loss=1.2238 acc=71.28%
lr = 0.004430
Epoch 181/200 | train_loss=0.1639 acc=95.38%  |  val_loss=1.1890 acc=71.72%
lr = 0.003980
Epoch 182/200 | train_loss=0.1506 acc=95.88%  |  val_loss=1.1939 acc=71.68%
lr = 0.003554
Epoch 183/200 | train_loss=0.1360 acc=96.35%  |  val_loss=1.1701 acc=71.74%
lr = 0.003152
Epoch 184/200 | train_loss=0.1301 acc=96.54%  |  val_loss=1.1715 acc=72.24%
lr = 0.002773
Epoch 185/200 | train_loss=0.1252 acc=96.61%  |  val_loss=1.1565 acc=72.26%
lr = 0.002418
Epoch 186/200 | train_loss=0.1173 acc=96.96%  |  val_loss=1.1663 acc=72.34%
lr = 0.002088
Epoch 187/200 | train_loss=0.1135 acc=97.06%  |  val_loss=1.1598 acc=73.12%
```

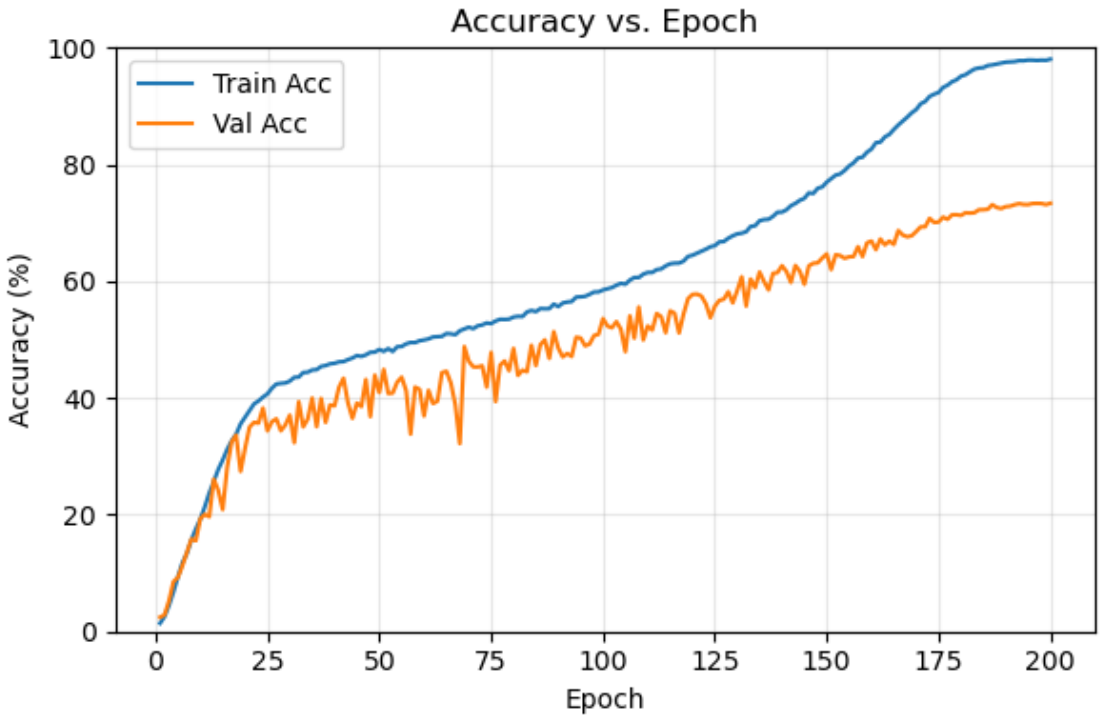
```
lr = 0.001781
Epoch 188/200 | train_loss=0.1082 acc=97.23%  |  val_loss=1.1530 acc=72.64%
lr = 0.001499
Epoch 189/200 | train_loss=0.1035 acc=97.40%  |  val_loss=1.1473 acc=72.46%
lr = 0.001241
Epoch 190/200 | train_loss=0.0999 acc=97.54%  |  val_loss=1.1447 acc=72.78%
lr = 0.001008
Epoch 191/200 | train_loss=0.0988 acc=97.58%  |  val_loss=1.1484 acc=72.86%
lr = 0.000798
Epoch 192/200 | train_loss=0.0968 acc=97.64%  |  val_loss=1.1384 acc=73.12%
lr = 0.000614
Epoch 193/200 | train_loss=0.0925 acc=97.79%  |  val_loss=1.1436 acc=73.30%
lr = 0.000454
Epoch 194/200 | train_loss=0.0916 acc=97.78%  |  val_loss=1.1367 acc=73.16%
lr = 0.000318
Epoch 195/200 | train_loss=0.0871 acc=97.89%  |  val_loss=1.1331 acc=73.14%
lr = 0.000207
Epoch 196/200 | train_loss=0.0883 acc=97.88%  |  val_loss=1.1402 acc=73.32%
lr = 0.000121
Epoch 197/200 | train_loss=0.0895 acc=97.83%  |  val_loss=1.1343 acc=73.34%
lr = 0.000059
Epoch 198/200 | train_loss=0.0879 acc=97.88%  |  val_loss=1.1319 acc=73.32%
lr = 0.000022
Epoch 199/200 | train_loss=0.0872 acc=97.86%  |  val_loss=1.1343 acc=73.14%
lr = 0.000010
Epoch 200/200 | train_loss=0.0855 acc=98.05%  |  val_loss=1.1315 acc=73.36%
Training Complete
```

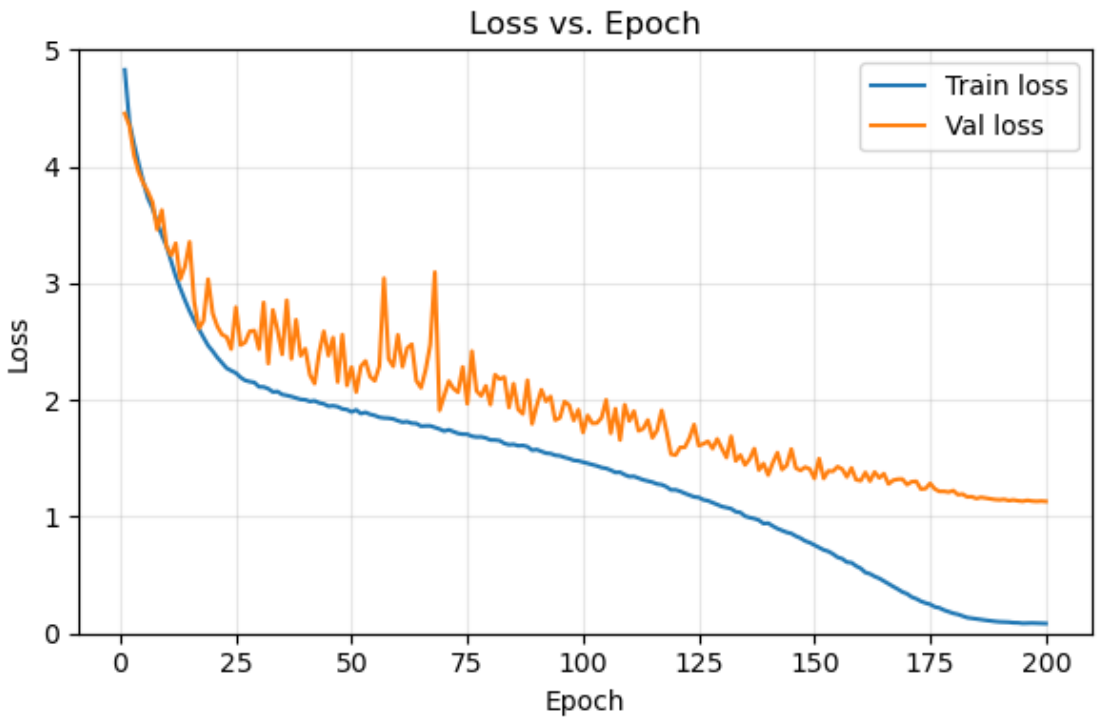
[8]:
```
test_loss, test_top1 = evaluate_top1(model, test_loader, loss_fun, device)
print(f"[TEST] loss={test_loss:.4f}  top1_acc={test_top1:.2f}%")
```

```
[TEST] loss=1.1656  top1_acc=73.07%
```

[11]:
```
#drop_out = 0.1 , lr = 0.2 with scheduler, momentum=0.9, weight_decay=5e-4

import os
import matplotlib.pyplot as plt
import numpy as np

os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

epochs = np.arange(1, len(accuracy_train_list)+1)

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy_train_list, label='Train Acc')
plt.plot(epochs, accuracy_val_list,   label='Val Acc')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy (%)')
plt.title('Accuracy vs. Epoch')
plt.ylim(0, 100)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss_train_list, label='Train loss')
plt.plot(epochs, loss_val_list,   label='Val loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs. Epoch')
plt.ylim(0, 5)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```





[13]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import CosineAnnealingLR

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = ResNetCIFAR(num_classes=100, drop_out=0.1).to(device)
loss_fun = nn.CrossEntropyLoss(label_smoothing=0.1)

#lr=0.01
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9,␣
 ↪weight_decay=5e-4)
#scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs, eta_min=1e-5)

num_epochs = 50

accuracy_train_list2, accuracy_val_list2 = [], []
loss_train_list2, loss_val_list2 = [], []

for epoch in range(num_epochs):
    # ---------- train ----------
    model.train()
```

```
    running_loss,correct, total = 0.0, 0, 0

    for x_batch, y_batch in train_loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        logits = model(x_batch)
        loss = loss_fun(logits, y_batch)

        loss.backward()
        optimizer.step()

        b = y_batch.size(0)
        running_loss += loss.item() * b
        pred = logits.argmax(dim=1)
        correct += (pred == y_batch).sum().item()
        total += b

    train_loss = running_loss / total
    train_acc  = 100.0 * correct / total



    # ---------- val ----------
    model.eval()
    running_loss_val,correct_val, total_val = 0.0, 0, 0
    with torch.no_grad():
        for x_val, y_val in val_loader:
            x_val = x_val.to(device)
            y_val = y_val.to(device)
            out = model(x_val)
            loss_t = loss_fun(out, y_val)

            b = y_val.size(0)
            running_loss_val += loss_t.item() * b
            pred_val = out.argmax(dim=1)
            correct_val += (pred_val== y_val).sum().item()
            total_val += b

    val_loss = running_loss_val / total_val
    val_acc  = 100.0 * correct_val / total_val

    loss_train_list2.append(train_loss)
    loss_val_list2.append(val_loss)
    accuracy_train_list2.append(train_acc)
    accuracy_val_list2.append(val_acc)
```

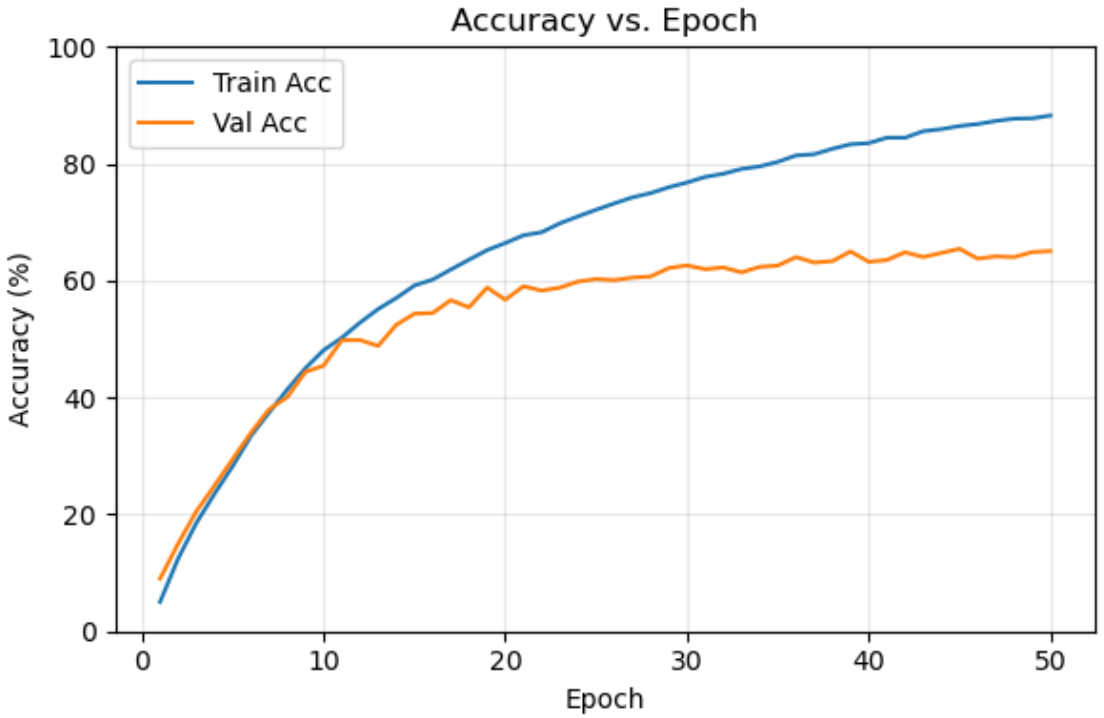
```
    #scheduler.step()
    #print(f"lr = {optimizer.param_groups[0]['lr']:.6f}")

    print(f"Epoch {epoch+1:02d}/{num_epochs} | "
          f"train_loss={train_loss:.4f} acc={train_acc:.2f}%  |  "
          f"val_loss={val_loss:.4f} acc={val_acc:.2f}%")

print("Training Complete")


test_loss, test_top1 = evaluate_top1(model, test_loader, loss_fun, device)
print(f"[TEST] loss={test_loss:.4f}  top1_acc={test_top1:.2f}%")
```

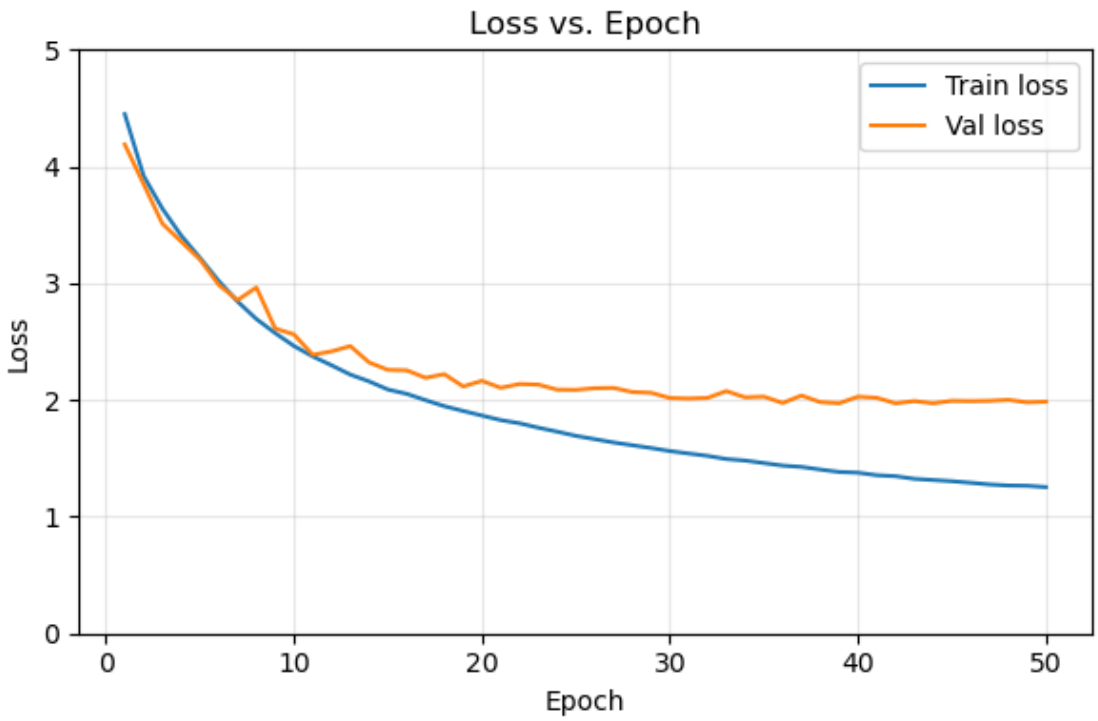
```
Epoch 01/50 | train_loss=4.4485 acc=5.04%  |  val_loss=4.1888 acc=9.02%
Epoch 02/50 | train_loss=3.9203 acc=12.48%  |  val_loss=3.8498 acc=15.04%
Epoch 03/50 | train_loss=3.6397 acc=18.55%  |  val_loss=3.5107 acc=20.54%
Epoch 04/50 | train_loss=3.4086 acc=23.56%  |  val_loss=3.3571 acc=24.94%
Epoch 05/50 | train_loss=3.2173 acc=28.29%  |  val_loss=3.2036 acc=29.48%
Epoch 06/50 | train_loss=3.0188 acc=33.42%  |  val_loss=2.9866 acc=33.98%
Epoch 07/50 | train_loss=2.8440 acc=37.43%  |  val_loss=2.8553 acc=37.96%
Epoch 08/50 | train_loss=2.6938 acc=41.38%  |  val_loss=2.9642 acc=40.08%
Epoch 09/50 | train_loss=2.5733 acc=45.02%  |  val_loss=2.6129 acc=44.38%
Epoch 10/50 | train_loss=2.4626 acc=48.10%  |  val_loss=2.5618 acc=45.46%
Epoch 11/50 | train_loss=2.3733 acc=50.22%  |  val_loss=2.3873 acc=49.80%
Epoch 12/50 | train_loss=2.2973 acc=52.81%  |  val_loss=2.4166 acc=49.84%
Epoch 13/50 | train_loss=2.2175 acc=55.14%  |  val_loss=2.4618 acc=48.84%
Epoch 14/50 | train_loss=2.1592 acc=57.02%  |  val_loss=2.3212 acc=52.44%
Epoch 15/50 | train_loss=2.0897 acc=59.17%  |  val_loss=2.2584 acc=54.36%
Epoch 16/50 | train_loss=2.0527 acc=60.18%  |  val_loss=2.2545 acc=54.46%
Epoch 17/50 | train_loss=1.9985 acc=61.90%  |  val_loss=2.1900 acc=56.66%
Epoch 18/50 | train_loss=1.9471 acc=63.58%  |  val_loss=2.2215 acc=55.44%
Epoch 19/50 | train_loss=1.9062 acc=65.22%  |  val_loss=2.1162 acc=58.82%
Epoch 20/50 | train_loss=1.8668 acc=66.44%  |  val_loss=2.1640 acc=56.74%
Epoch 21/50 | train_loss=1.8277 acc=67.76%  |  val_loss=2.1046 acc=59.04%
Epoch 22/50 | train_loss=1.8006 acc=68.28%  |  val_loss=2.1369 acc=58.30%
Epoch 23/50 | train_loss=1.7628 acc=69.80%  |  val_loss=2.1319 acc=58.80%
Epoch 24/50 | train_loss=1.7290 acc=70.99%  |  val_loss=2.0886 acc=59.84%
Epoch 25/50 | train_loss=1.6928 acc=72.14%  |  val_loss=2.0868 acc=60.28%
Epoch 26/50 | train_loss=1.6647 acc=73.22%  |  val_loss=2.1011 acc=60.08%
Epoch 27/50 | train_loss=1.6363 acc=74.25%  |  val_loss=2.1037 acc=60.52%
Epoch 28/50 | train_loss=1.6125 acc=74.98%  |  val_loss=2.0685 acc=60.72%
Epoch 29/50 | train_loss=1.5888 acc=75.99%  |  val_loss=2.0618 acc=62.16%
Epoch 30/50 | train_loss=1.5626 acc=76.79%  |  val_loss=2.0169 acc=62.60%
Epoch 31/50 | train_loss=1.5416 acc=77.75%  |  val_loss=2.0126 acc=61.96%
Epoch 32/50 | train_loss=1.5215 acc=78.29%  |  val_loss=2.0176 acc=62.26%
Epoch 33/50 | train_loss=1.4944 acc=79.11%  |  val_loss=2.0766 acc=61.42%
```

```
Epoch 34/50 | train_loss=1.4813 acc=79.55%  |  val_loss=2.0226 acc=62.36%
Epoch 35/50 | train_loss=1.4598 acc=80.33%  |  val_loss=2.0286 acc=62.58%
Epoch 36/50 | train_loss=1.4382 acc=81.42%  |  val_loss=1.9748 acc=64.04%
Epoch 37/50 | train_loss=1.4269 acc=81.63%  |  val_loss=2.0381 acc=63.10%
Epoch 38/50 | train_loss=1.4042 acc=82.56%  |  val_loss=1.9819 acc=63.34%
Epoch 39/50 | train_loss=1.3838 acc=83.34%  |  val_loss=1.9724 acc=65.00%
Epoch 40/50 | train_loss=1.3776 acc=83.53%  |  val_loss=2.0281 acc=63.22%
Epoch 41/50 | train_loss=1.3556 acc=84.46%  |  val_loss=2.0191 acc=63.54%
Epoch 42/50 | train_loss=1.3474 acc=84.45%  |  val_loss=1.9717 acc=64.86%
Epoch 43/50 | train_loss=1.3246 acc=85.58%  |  val_loss=1.9908 acc=64.04%
Epoch 44/50 | train_loss=1.3146 acc=85.92%  |  val_loss=1.9723 acc=64.74%
Epoch 45/50 | train_loss=1.3044 acc=86.46%  |  val_loss=1.9929 acc=65.46%
Epoch 46/50 | train_loss=1.2911 acc=86.80%  |  val_loss=1.9904 acc=63.76%
Epoch 47/50 | train_loss=1.2766 acc=87.32%  |  val_loss=1.9929 acc=64.16%
Epoch 48/50 | train_loss=1.2675 acc=87.71%  |  val_loss=2.0014 acc=64.02%
Epoch 49/50 | train_loss=1.2652 acc=87.78%  |  val_loss=1.9804 acc=64.90%
Epoch 50/50 | train_loss=1.2541 acc=88.24%  |  val_loss=1.9856 acc=65.06%
Training Complete
[TEST] loss=2.0144  top1_acc=64.05%
```

[14]:
```python
#drop_out = 0.1 , lr = 0.01, momentum=0.9, weight_decay=5e-4

import os
import matplotlib.pyplot as plt
import numpy as np

os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

epochs = np.arange(1, len(accuracy_train_list2)+1)

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy_train_list2, label='Train Acc')
plt.plot(epochs, accuracy_val_list2,   label='Val Acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Accuracy vs. Epoch')
plt.ylim(0, 100)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss_train_list2, label='Train loss')
plt.plot(epochs, loss_val_list2, label='Val loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.title('Loss vs. Epoch')
plt.ylim(0, 5)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```





[15]:
```
import torch
import torch.nn as nn
import torch.optim as optim

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = ResNetCIFAR(num_classes=100, drop_out=0.1).to(device)
loss_fun = nn.CrossEntropyLoss(label_smoothing=0.1)

#lr=0.001
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9,␣
 ↪weight_decay=5e-4)

num_epochs = 50

accuracy_train_list3, accuracy_val_list3 = [], []
loss_train_list3, loss_val_list3 = [], []

for epoch in range(num_epochs):
    # ---------- train ----------
    model.train()
    running_loss,correct, total = 0.0, 0, 0
```

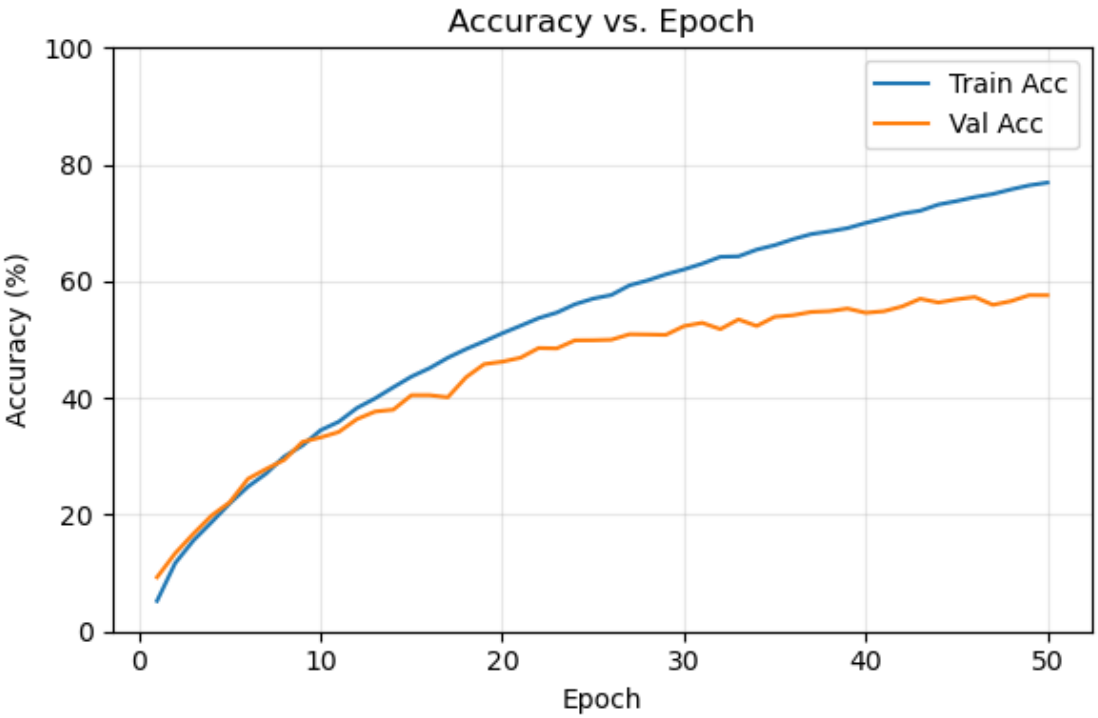
```
    for x_batch, y_batch in train_loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        logits = model(x_batch)
        loss = loss_fun(logits, y_batch)

        loss.backward()
        optimizer.step()

        b = y_batch.size(0)
        running_loss += loss.item() * b
        pred = logits.argmax(dim=1)
        correct += (pred == y_batch).sum().item()
        total += b

    train_loss = running_loss / total
    train_acc  = 100.0 * correct / total



    # ---------- val ----------
    model.eval()
    running_loss_val,correct_val, total_val = 0.0, 0, 0
    with torch.no_grad():
        for x_val, y_val in val_loader:
            x_val = x_val.to(device)
            y_val = y_val.to(device)
            out = model(x_val)
            loss_t = loss_fun(out, y_val)

            b = y_val.size(0)
            running_loss_val += loss_t.item() * b
            pred_val = out.argmax(dim=1)
            correct_val += (pred_val== y_val).sum().item()
            total_val += b

    val_loss = running_loss_val / total_val
    val_acc  = 100.0 * correct_val / total_val

    loss_train_list3.append(train_loss)
    loss_val_list3.append(val_loss)
    accuracy_train_list3.append(train_acc)
    accuracy_val_list3.append(val_acc)

    print(f"Epoch {epoch+1:02d}/{num_epochs} | "
```

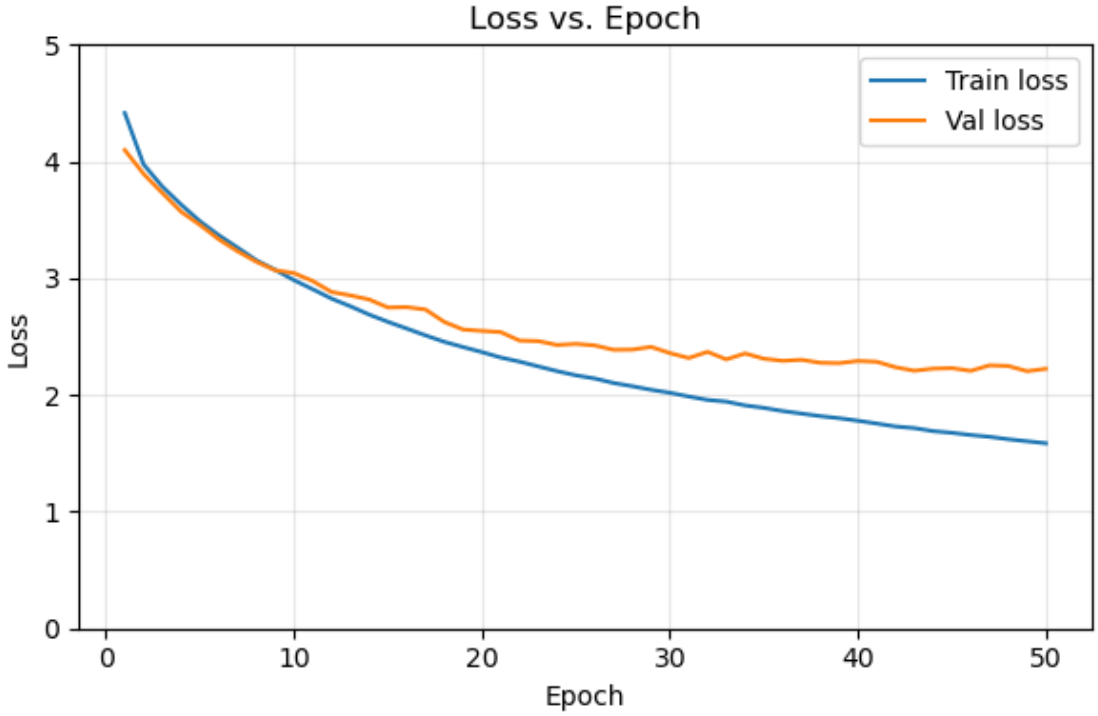
```
          f"train_loss={train_loss:.4f} acc={train_acc:.2f}%  |  "
          f"val_loss={val_loss:.4f} acc={val_acc:.2f}%")

print("Training Complete")

test_loss, test_top1 = evaluate_top1(model, test_loader, loss_fun, device)
print(f"[TEST] loss={test_loss:.4f}  top1_acc={test_top1:.2f}%")
```

```
Epoch 01/50 | train_loss=4.4149 acc=5.25%  |  val_loss=4.0984 acc=9.30%
Epoch 02/50 | train_loss=3.9754 acc=11.72%  |  val_loss=3.8943 acc=13.40%
Epoch 03/50 | train_loss=3.7849 acc=15.59%  |  val_loss=3.7302 acc=16.74%
Epoch 04/50 | train_loss=3.6302 acc=18.72%  |  val_loss=3.5694 acc=19.86%
Epoch 05/50 | train_loss=3.4881 acc=21.94%  |  val_loss=3.4569 acc=22.14%
Epoch 06/50 | train_loss=3.3687 acc=24.81%  |  val_loss=3.3336 acc=26.14%
Epoch 07/50 | train_loss=3.2626 acc=27.06%  |  val_loss=3.2310 acc=27.84%
Epoch 08/50 | train_loss=3.1534 acc=29.96%  |  val_loss=3.1416 acc=29.40%
Epoch 09/50 | train_loss=3.0722 acc=31.86%  |  val_loss=3.0685 acc=32.48%
Epoch 10/50 | train_loss=2.9840 acc=34.48%  |  val_loss=3.0440 acc=33.24%
Epoch 11/50 | train_loss=2.9058 acc=35.94%  |  val_loss=2.9761 acc=34.18%
Epoch 12/50 | train_loss=2.8257 acc=38.29%  |  val_loss=2.8824 acc=36.38%
Epoch 13/50 | train_loss=2.7602 acc=39.93%  |  val_loss=2.8533 acc=37.68%
Epoch 14/50 | train_loss=2.6887 acc=41.84%  |  val_loss=2.8182 acc=38.00%
Epoch 15/50 | train_loss=2.6267 acc=43.67%  |  val_loss=2.7498 acc=40.48%
Epoch 16/50 | train_loss=2.5691 acc=45.12%  |  val_loss=2.7528 acc=40.48%
Epoch 17/50 | train_loss=2.5105 acc=46.90%  |  val_loss=2.7322 acc=40.12%
Epoch 18/50 | train_loss=2.4554 acc=48.38%  |  val_loss=2.6256 acc=43.54%
Epoch 19/50 | train_loss=2.4106 acc=49.70%  |  val_loss=2.5607 acc=45.80%
Epoch 20/50 | train_loss=2.3675 acc=51.09%  |  val_loss=2.5493 acc=46.22%
Epoch 21/50 | train_loss=2.3210 acc=52.37%  |  val_loss=2.5387 acc=46.90%
Epoch 22/50 | train_loss=2.2858 acc=53.69%  |  val_loss=2.4663 acc=48.56%
Epoch 23/50 | train_loss=2.2448 acc=54.63%  |  val_loss=2.4618 acc=48.50%
Epoch 24/50 | train_loss=2.2042 acc=56.07%  |  val_loss=2.4284 acc=49.88%
Epoch 25/50 | train_loss=2.1685 acc=57.02%  |  val_loss=2.4381 acc=49.90%
Epoch 26/50 | train_loss=2.1414 acc=57.65%  |  val_loss=2.4247 acc=49.98%
Epoch 27/50 | train_loss=2.1031 acc=59.30%  |  val_loss=2.3873 acc=50.92%
Epoch 28/50 | train_loss=2.0752 acc=60.15%  |  val_loss=2.3890 acc=50.88%
Epoch 29/50 | train_loss=2.0446 acc=61.18%  |  val_loss=2.4121 acc=50.82%
Epoch 30/50 | train_loss=2.0189 acc=62.03%  |  val_loss=2.3587 acc=52.32%
Epoch 31/50 | train_loss=1.9872 acc=63.02%  |  val_loss=2.3166 acc=52.88%
Epoch 32/50 | train_loss=1.9577 acc=64.18%  |  val_loss=2.3697 acc=51.80%
Epoch 33/50 | train_loss=1.9434 acc=64.26%  |  val_loss=2.3050 acc=53.46%
Epoch 34/50 | train_loss=1.9109 acc=65.42%  |  val_loss=2.3559 acc=52.34%
Epoch 35/50 | train_loss=1.8898 acc=66.19%  |  val_loss=2.3108 acc=53.92%
Epoch 36/50 | train_loss=1.8625 acc=67.21%  |  val_loss=2.2929 acc=54.18%
Epoch 37/50 | train_loss=1.8408 acc=68.08%  |  val_loss=2.3015 acc=54.74%
Epoch 38/50 | train_loss=1.8190 acc=68.55%  |  val_loss=2.2762 acc=54.88%
Epoch 39/50 | train_loss=1.8015 acc=69.10%  |  val_loss=2.2728 acc=55.32%
```

```
Epoch 40/50 | train_loss=1.7798 acc=69.99%  |  val_loss=2.2920 acc=54.60%
Epoch 41/50 | train_loss=1.7555 acc=70.76%  |  val_loss=2.2850 acc=54.86%
Epoch 42/50 | train_loss=1.7305 acc=71.58%  |  val_loss=2.2380 acc=55.68%
Epoch 43/50 | train_loss=1.7167 acc=72.08%  |  val_loss=2.2081 acc=57.00%
Epoch 44/50 | train_loss=1.6904 acc=73.14%  |  val_loss=2.2258 acc=56.36%
Epoch 45/50 | train_loss=1.6762 acc=73.74%  |  val_loss=2.2303 acc=56.90%
Epoch 46/50 | train_loss=1.6572 acc=74.41%  |  val_loss=2.2068 acc=57.30%
Epoch 47/50 | train_loss=1.6421 acc=74.95%  |  val_loss=2.2542 acc=55.96%
Epoch 48/50 | train_loss=1.6209 acc=75.74%  |  val_loss=2.2487 acc=56.60%
Epoch 49/50 | train_loss=1.6039 acc=76.44%  |  val_loss=2.2041 acc=57.62%
Epoch 50/50 | train_loss=1.5867 acc=76.90%  |  val_loss=2.2245 acc=57.60%
Training Complete
[TEST] loss=2.2012  top1_acc=57.82%
```

[ ]:
```python
#drop_out = 0.1 , lr = 0.001, momentum=0.9, weight_decay=5e-4

import os
import matplotlib.pyplot as plt
import numpy as np

os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

epochs = np.arange(1, len(accuracy_train_list3)+1)

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy_train_list3, label='Train Acc')
plt.plot(epochs, accuracy_val_list3,   label='Val Acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Accuracy vs. Epoch')
plt.ylim(0, 100)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss_train_list3, label='Train loss')
plt.plot(epochs, loss_val_list3, label='Val loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs. Epoch')
plt.ylim(0, 5)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```


Accuracy vs. Epoch
Train Acc
Val Acc
Accuracy (%)
Epoch
0
20
40
60
80
100
0
10
20
30
40
50


Loss vs. Epoch
Train loss
Val loss
Loss
Epoch
0
1
2
3
4
5
0
10
20
30
40
50

3. output sizes for each convX layer.

conv1 : (32,32,32)

conv2 : (32,32,32)

conv3 : (64,16,16)

conv4 : (128,8,8)

conv5 : (256,4,4)