image-classifier

November 14, 2025

[1]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

1 Step 1: Download a dataset and preview images

[2]:
```
# !tar -xvf './cifar100.tar'
```

2 Step 2: Custom Data Loading

[3]:
```
import os
import time
import glob
import torch
import shutil
import torchvision
import torch.nn as nn
from torch.nn import functional as F
import torchvision.transforms as transforms
from PIL import Image
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

config = {
    "data_path": "./cifar100",
    "batch_size": 256,
    "lr": 1e-3,
    "momentum": 0.9,
    "weight_decay": 1e-4,
    "save_root": "./result",
    "epochs": 50,
}

class mydataset(Dataset):
    def __init__(self, data_dir, flag, transform):
        super(mydataset,self).__init__()
        self.root       = data_dir
```

```
        self.label      = flag
        self.transform  = transform

        self.img_dir = os.path.join(self.root, self.label)
        self.img_names  = glob.glob(os.path.join(self.img_dir, '*.jpg'))

        self.tags = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed',␣
↪'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus',␣
↪'butterfly', 'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair',␣
↪'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch', 'crab', 'crocodile',␣
↪'cup', 'dinosaur', 'dolphin', 'elephant', 'flatfish', 'forest', 'fox',␣
↪'girl', 'hamster', 'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower',␣
↪'leopard', 'lion', 'lizard', 'lobster', 'man', 'maple_tree', 'motorcycle',␣
↪'mountain', 'mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter',␣
↪'palm_tree', 'pear', 'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy',␣
↪'porcupine', 'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',␣
↪'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',␣
↪'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',␣
↪'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',␣
↪'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman',␣
↪'worm']

    def RGB2Gradient(self, img:torch.tensor):
        """
        Converts an RGB image tensor to its gradient magnitude using Sobel␣
↪operator.
        The output is replicated to 3 channels to match the input dimensions of␣
↪the other branch.
        """

        # Define Sobel kernels
        sobel_kernel_x = torch.tensor([[-1., 0., 1.],
                                       [-2., 0., 2.],
                                       [-1., 0., 1.]], dtype=torch.float32).
↪reshape((1, 1, 3, 3))
        sobel_kernel_y = torch.tensor([[-1., -2., -1.],
                                       [ 0.,  0.,  0.],
                                       [ 1.,  2.,  1.]], dtype=torch.float32).
↪reshape((1, 1, 3, 3))

        sobel_kernel_x = sobel_kernel_x.to(img.device)
        sobel_kernel_y = sobel_kernel_y.to(img.device)

        # Convert to grayscale: [3, H, W] -> Output [1, H, W]
        # gray_img = img[0, :, :] * 0.2989 + img[1, :, :] * 0.5870 + img[2, :, :
↪] * 0.1140
```

```
        # gray_img = gray_img.unsqueeze(0)
        gray_img = transforms.Grayscale(num_output_channels=1)(img)

        # Add batch dimension [1, H, W] -> [1, 1, H, W]
        gray_img_batch = gray_img.unsqueeze(0)

        # [YOU NEED TO FILL] Apply Sobel filters
        # Use F.conv2d, gray_img_batch, and the Sobel kernels (sobel_kernel_x,␣
↪sobel_kernel_y)
        # to calculate the gradients in the x and y directions.
        grad_x = F.conv2d(gray_img_batch, sobel_kernel_x, padding=1)
        grad_y = F.conv2d(gray_img_batch, sobel_kernel_y, padding=1)

        # [YOU NEED TO FILL] Calculate gradient magnitude ---
        # Calculate the magnitude (G = sqrt(Gx^2 + Gy^2)) from grad_x and␣
↪grad_y.
        # The result should be stored in a variable named 'magnitude'.
        magnitude = torch.sqrt(grad_x ** 2 + grad_y ** 2)

        # Normalize magnitude
        mag_min = magnitude.min()
        mag_max = magnitude.max()
        epsilon = 1e-6 # Avoid division by zero
        normalized_magnitude = (magnitude - mag_min) / (mag_max - mag_min +␣
↪epsilon)

        # Replicate to 3 channels [1, 1, H, W] -> [1, 3, H, W]
        normalized_magnitude_3channel = normalized_magnitude.repeat(1, 3, 1, 1)

        # Remove batch dimension [1, 3, H, W] -> [3, H, W]
        return normalized_magnitude_3channel.squeeze(0)

    def __getitem__(self,idx):
        img_name = self.img_names[idx]
        img = Image.open(os.path.join(img_name)).convert('RGB')

        # original image
        img = self.transform(img)

        # gradient image
        grad_img = self.RGB2Gradient(img)

        for i in range(len(self.tags)):
            if self.tags[i] in img_name:
                tag = i
                break
```

```
        return img, grad_img, tag

    def __len__(self):
        return len(self.img_names)


cifar100_mean = [0.5071, 0.4865, 0.4409]
cifar100_std = [0.2673, 0.2564, 0.2762]
transform_train = transforms.Compose(
        [transforms.Resize([64, 64]),
         transforms.RandomHorizontalFlip(),    # 50%
         transforms.RandomCrop(64, padding=4), #     4       64x64
         transforms.ToTensor(),
         transforms.Normalize(mean=cifar100_mean, std=cifar100_std)])
transform_test = transforms.Compose(
    [transforms.Resize([64, 64]),
     transforms.ToTensor(),
     transforms.Normalize(mean=cifar100_mean, std=cifar100_std)])

train_dataset = mydataset(data_dir=config['data_path'], flag= "train",␣
 ↪transform=transform_train)
test_dataset  = mydataset(data_dir=config['data_path'], flag= "test",␣
 ↪transform=transform_test)

# define data loader
train_loader = DataLoader(
        train_dataset,
        batch_size=config['batch_size'], shuffle=True, num_workers=0,␣
 ↪pin_memory=True, drop_last=False)

test_loader = DataLoader(
        test_dataset,
        batch_size=config['batch_size'], shuffle=True, num_workers=0,␣
 ↪pin_memory=True, drop_last=False)
```

3 Step 3: Configure the Neural Network

[4]:
```
import torch.nn as nn
from torch.nn import functional as F
from torch.nn import Sequential as Seq

class CommonBlock(nn.Module):
    def __init__(self, in_channel, out_channel, stride):        #  Block
        super(CommonBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size=3,␣
 ↪stride=stride, padding=1, bias=False)
```

```
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3,␣
↪stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)

    def forward(self, x):
        identity = x                                                    #␣
↪ Block shortcut

        x = F.relu(self.bn1(self.conv1(x)), inplace=True)               #
        x = self.bn2(self.conv2(x))                                     #     relu

        x += identity                                                   #
        return F.relu(x, inplace=True)                                  #

class SpecialBlock(nn.Module):                                          #␣
↪ Block
    def __init__(self, in_channel, out_channel, stride):               #␣
↪   stride    shortcut   stride
        super(SpecialBlock, self).__init__()
        self.change_channel = nn.Sequential(                            #␣
↪       change_channel
            nn.Conv2d(in_channel, out_channel, kernel_size=1, stride=stride[0],␣
↪padding=0, bias=False),
            nn.BatchNorm2d(out_channel)
        )
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size=3,␣
↪stride=stride[0], padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3,␣
↪stride=stride[1], padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)

    def forward(self, x):
        identity = self.change_channel(x)                               #␣
↪ change_channel

        x = F.relu(self.bn1(self.conv1(x)), inplace=True)
        x = self.bn2(self.conv2(x))                                     #

        x += identity
        return F.relu(x, inplace=True)                                  #

class SEBlock(nn.Module):
    """
    Squeeze-and-Excitation (SE) Block   2D    (B, C, H, W)
```

```
    """
    def __init__(self, in_channels, reduction_ratio=16):
        super(SEBlock, self).__init__()

        hidden_dim = max(16, in_channels // reduction_ratio)

        # Squeeze (Global Average Pool)
        self.avg_pool = nn.AdaptiveAvgPool2d(1) #   [B, C, 1, 1]

        # Excitation (FC layers)
        self.fc = nn.Sequential(
            nn.Linear(in_channels, hidden_dim, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(hidden_dim, in_channels, bias=False),
            nn.Sigmoid()
        )
        self.flatten = nn.Flatten(start_dim=1)

    def forward(self, x):
        # x shape: [B, C, H, W]
        b, c, _, _ = x.size()

        # Squeeze
        y = self.avg_pool(x) # [B, C, 1, 1]
        y = self.flatten(y)  # [B, C]

        # Excitation
        weights = self.fc(y) # [B, C]

        # Rescale: [B, C] -> [B, C, 1, 1]
        weights = weights.view(b, c, 1, 1)

        #
        return x * weights

class ConvNet(nn.Module):
    def __init__(self, classes_num):
        super(ConvNet, self).__init__()

        # --- RGB   ---
        self.prepare_rgb = nn.Sequential(           #   == [batch, 64, 56, 56]
            nn.Conv2d(3, 64, 7, 2, 3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, 2, 1)
        )
```

```
        self.layer1_rgb = nn.Sequential(            #␣
↪layer1      channel 64  CommonBlock
            CommonBlock(64, 64, 1),
            CommonBlock(64, 64, 1)
        )
        self.layer2_rgb = nn.Sequential(            #␣
↪layer234     channel   SpecialBlock  CommonBlock
            SpecialBlock(64, 128, [2, 1]),
            CommonBlock(128, 128, 1)
        )
        self.layer3_rgb = nn.Sequential(
            SpecialBlock(128, 256, [2, 1]),
            CommonBlock(256, 256, 1)
        )
        self.layer4_rgb = nn.Sequential(
            SpecialBlock(256, 512, [2, 1]),
            CommonBlock(512, 512, 1)
        )

        # ---    ---
        self.prepare_grad = nn.Sequential(          #   == [batch, 64, 56, 56]
            nn.Conv2d(3, 64, 7, 2, 3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, 2, 1)
        )
        self.layer1_grad = nn.Sequential(           #␣
↪layer1      channel 64  CommonBlock
            CommonBlock(64, 64, 1),
            CommonBlock(64, 64, 1)
        )
        self.layer2_grad = nn.Sequential(           #␣
↪layer234     channel   SpecialBlock  CommonBlock
            SpecialBlock(64, 128, [2, 1]),
            CommonBlock(128, 128, 1)
        )
        self.layer3_grad = nn.Sequential(
            SpecialBlock(128, 256, [2, 1]),
            CommonBlock(256, 256, 1)
        )
        self.layer4_grad = nn.Sequential(
            SpecialBlock(256, 512, [2, 1]),
            CommonBlock(512, 512, 1)
        )

        # ---    Layer 4    ---
        # Layer 3   (256) + Layer 3   (256) = 512
```

```
        #
        self.attention_mid = SEBlock(in_channels=512) #      2D

        #    Layer 4,    512
        self.shared_layer4 = nn.Sequential(
            SpecialBlock(512, 512, [2, 1]), #   512
            CommonBlock(512, 512, 1)
        )

        #
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.flatten = nn.Flatten(start_dim=1)

        # (  Q3 T-SNE   )
        #      fc_fused    "  "
        self.fc_fused = nn.Linear(512, 512) #    T-SNE
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.5)
        self.classifier = nn.Linear(512, classes_num) #

        # self.head = Seq(nn.AdaptiveAvgPool2d(1),
        #           nn.Flatten(start_dim=1),
        #           nn.Linear(512, classes_num))

    def forward(self, x_rgb, x_grad):
        # x = self.prepare(x)         #

        # x = self.layer1(x)          #
        # x = self.layer2(x)
        # x = self.layer3(x)
        # x = self.layer4(x)

        # x = self.head(x)

        # return x

        # RGB    ( L3)
        x_rgb = self.prepare_rgb(x_rgb)
        x_rgb = self.layer1_rgb(x_rgb)
        x_rgb = self.layer2_rgb(x_rgb)
        x_rgb = self.layer3_rgb(x_rgb) # Shape: [B, 256, H, W]

        #     ( L3)
        x_grad = self.prepare_grad(x_grad)
        x_grad = self.layer1_grad(x_grad)
        x_grad = self.layer2_grad(x_grad)
        x_grad = self.layer3_grad(x_grad) # Shape: [B, 256, H, W]
```

```
        # ---     ---
        #    (  T-SNE)
        x_fused = torch.cat((x_rgb, x_grad), dim=1) # Shape: [B, 512, H, W]

        #      Layer 4    2D
        x_fused_attended = self.attention_mid(x_fused)

        #    Layer 4
        x_shared = self.shared_layer4(x_fused_attended) # Shape: [B, 512, H',␣
 ↪W']


        #
        x_pool = self.pool(x_shared) # Shape: [B, 512, 1, 1]
        x_flat = self.flatten(x_pool) # Shape: [B, 512]

        #    (  T-SNE)
        x_processed = self.relu(self.fc_fused(x_flat)) # [B, 512]

        x_processed = self.dropout(x_processed)
        x_out = self.classifier(x_processed)

        return x_out

# config = {
#     "lr": 1e-3,
#     "momentum": 0.9,
#     "weight_decay": 1e-4,
# }

device = torch.device("cuda:0" if torch.cuda.is_available() else "mps" if torch.
 ↪backends.mps.is_available() else "cpu")
print("Using device:", device)
net = ConvNet(classes_num=100).to(device)
criterion = torch.nn.CrossEntropyLoss(label_smoothing=0.1).to(device)
# optimizer = torch.optim.SGD(net.parameters(), lr=config["lr"],␣
 ↪momentum=config["momentum"], weight_decay=config["weight_decay"])
# AdamW
optimizer = torch.optim.AdamW(net.parameters(), lr=config["lr"],␣
 ↪weight_decay=config["weight_decay"])
#
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,␣
 ↪T_max=config["epochs"]) # T_max = num_epochs
```

```
Using device: cuda:0
```

4 Step 4: Train the network and save model

[5]:
```
import time

class AverageMeter(object):
    def __init__(self):
      self.reset()

    def reset(self):
      self.val   = 0
      self.avg   = 0
      self.sum   = 0
      self.count = 0

    def update(self, val, n=1):
      self.val   = val
      self.sum   += val * n
      self.count += n
      self.avg   = self.sum / self.count

def accuracy(output, target, topk=(1,1)):
    """Computes the precision@k for the specified values of k"""
    maxk = max(topk)
    batch_size = target.size(0)

    _, pred = output.topk(maxk, 1, True, True)
    pred    = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))

    res = []
    for k in topk:
      correct_k = correct[:k].view(-1).float().sum(0)
      res.append(correct_k.mul_(100.0 / batch_size))
    return res[0]

def train(train_loader, net, optimizer, criterion, epoch):
    batch_time = AverageMeter()
    data_time  = AverageMeter()
    top1       = AverageMeter()

    LOSS = AverageMeter()

    net.train()

    end = time.time()
    for i, (img_rgb, img_grad, target) in enumerate(train_loader, start=1):
        data_time.update(time.time() - end)
```

```
        img_rgb = img_rgb.to(device)
        img_grad = img_grad.to(device)
        target = target.to(device)

        out = net(img_rgb, img_grad)

        loss = criterion(out, target)

        prec1 = accuracy(out, target) # prec1:list
        top1.update(prec1.item(), img_rgb.size(0))

        LOSS.update(loss.item(), img_rgb.size(0))

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        batch_time.update(time.time() - end)
        end = time.time()

        if i % 50 == 1:
            log_str = ('Epoch[{0}]:[{1:03}/{2:03}] '
                       'Time:{batch_time.val:.4f} '
                       'Data:{data_time.val:.4f}  '
                       'loss:{loss.val:.4f}({loss.avg:.4f})  '
                       'prec@1:{top1.val:.2f}({top1.avg:.2f})  '.format(
                       epoch, i, len(train_loader), batch_time=batch_time,␣
 ↪data_time=data_time,
                       loss=LOSS,
                       top1=top1))
            print(log_str)

    return LOSS.avg

def save_checkpoint(state, is_best, save_root, epoch):
    if not os.path.exists(save_root):
        os.makedirs(save_root)
    save_path = os.path.join(save_root, 'epoch_{}.pth.tar'.format(str(epoch)))
    torch.save(state, save_path)
    if is_best:
        best_save_path = os.path.join(save_root, 'model_best.pth.tar')
        shutil.copyfile(save_path, best_save_path)

# config = {
#     "save_root": "./result",
#     "epochs": 50,
```

```
# }

best_top1 = 0
test_top1 = 0
for epoch in range(1, config["epochs"]+1):
    # train one epoch
    epoch_start_time = time.time()
    train_loss = train(train_loader, net, optimizer, criterion, epoch)

    if 'scheduler' in locals(): #   scheduler
        scheduler.step()

    # evaluate on testing set
    # test_top1 = test(test_loader, net, criterion)

    epoch_duration = time.time() - epoch_start_time
    print('Epoch time: {}s'.format(int(epoch_duration)))

    # save model
    is_best = False
    # if test_top1 > best_top1:
    #     best_top1 = test_top1
    #     is_best = True
    print('Saving models......')
    save_checkpoint({
        'epoch': epoch,
        'net': net.state_dict(),
        'prec@1': test_top1,
    }, is_best, config["save_root"], epoch)
```

```
Epoch[1]:[001/196] Time:1.2327 Data:0.3740  loss:4.6564(4.6564)
prec@1:0.78(0.78)
Epoch[1]:[051/196] Time:0.3038 Data:0.2805  loss:4.1418(4.2954)
prec@1:9.38(8.39)
Epoch[1]:[101/196] Time:0.3051 Data:0.2821  loss:3.9532(4.1729)
prec@1:10.16(9.16)
Epoch[1]:[151/196] Time:0.3229 Data:0.2953  loss:3.8513(4.0840)
prec@1:11.33(9.95)
Epoch time: 62s
Saving models…
Epoch[2]:[001/196] Time:0.3514 Data:0.3264  loss:3.7240(3.7240)
prec@1:10.16(10.16)
Epoch[2]:[051/196] Time:0.3104 Data:0.2825  loss:3.6642(3.6846)
prec@1:15.23(14.28)
Epoch[2]:[101/196] Time:0.3094 Data:0.2887  loss:3.6081(3.6333)
prec@1:16.80(15.67)
Epoch[2]:[151/196] Time:0.3176 Data:0.2922  loss:3.4094(3.5937)
prec@1:20.70(16.82)
```

```
Epoch time: 62s
Saving models…
Epoch[3]:[001/196] Time:0.3117 Data:0.2879  loss:3.4748(3.4748)
prec@1:20.31(20.31)
Epoch[3]:[051/196] Time:0.3074 Data:0.2844  loss:3.2916(3.3558)
prec@1:26.17(22.78)
Epoch[3]:[101/196] Time:0.3070 Data:0.2836  loss:3.2680(3.3302)
prec@1:18.75(23.21)
Epoch[3]:[151/196] Time:0.3112 Data:0.2884  loss:3.1217(3.2903)
prec@1:27.73(24.41)
Epoch time: 61s
Saving models…
Epoch[4]:[001/196] Time:0.3414 Data:0.3184  loss:3.1680(3.1680)
prec@1:22.66(22.66)
Epoch[4]:[051/196] Time:0.3510 Data:0.3249  loss:3.0998(3.1067)
prec@1:34.38(29.08)
Epoch[4]:[101/196] Time:0.3487 Data:0.3253  loss:3.0128(3.0857)
prec@1:30.47(29.57)
Epoch[4]:[151/196] Time:0.3033 Data:0.2801  loss:2.9852(3.0671)
prec@1:29.69(30.10)
Epoch time: 61s
Saving models…
Epoch[5]:[001/196] Time:0.3063 Data:0.2829  loss:2.7824(2.7824)
prec@1:34.77(34.77)
Epoch[5]:[051/196] Time:0.3080 Data:0.2846  loss:2.9262(2.9004)
prec@1:32.03(35.17)
Epoch[5]:[101/196] Time:0.3049 Data:0.2816  loss:2.9157(2.9052)
prec@1:33.98(34.87)
Epoch[5]:[151/196] Time:0.3017 Data:0.2791  loss:2.8733(2.8941)
prec@1:38.67(35.43)
Epoch time: 60s
Saving models…
Epoch[6]:[001/196] Time:0.3060 Data:0.2829  loss:2.7555(2.7555)
prec@1:41.41(41.41)
Epoch[6]:[051/196] Time:0.3176 Data:0.2947  loss:2.6304(2.7718)
prec@1:43.36(38.27)
Epoch[6]:[101/196] Time:0.3049 Data:0.2808  loss:2.6115(2.7557)
prec@1:41.80(38.99)
Epoch[6]:[151/196] Time:0.3070 Data:0.2840  loss:2.7285(2.7475)
prec@1:41.80(39.35)
Epoch time: 61s
Saving models…
Epoch[7]:[001/196] Time:0.3081 Data:0.2846  loss:2.7407(2.7407)
prec@1:35.94(35.94)
Epoch[7]:[051/196] Time:0.3046 Data:0.2817  loss:2.5925(2.6459)
prec@1:40.62(42.37)
Epoch[7]:[101/196] Time:0.3102 Data:0.2876  loss:2.7256(2.6474)
prec@1:40.23(42.28)
```

```
Epoch[7]:[151/196] Time:0.3056 Data:0.2823  loss:2.5043(2.6370)
prec@1:48.83(42.71)
Epoch time: 60s
Saving models…
Epoch[8]:[001/196] Time:0.3110 Data:0.2878  loss:2.6124(2.6124)
prec@1:42.97(42.97)
Epoch[8]:[051/196] Time:0.3109 Data:0.2832  loss:2.4832(2.5113)
prec@1:49.61(46.61)
Epoch[8]:[101/196] Time:0.3016 Data:0.2773  loss:2.5418(2.5145)
prec@1:46.48(46.38)
Epoch[8]:[151/196] Time:0.3059 Data:0.2832  loss:2.5497(2.5141)
prec@1:46.09(46.55)
Epoch time: 61s
Saving models…
Epoch[9]:[001/196] Time:0.3088 Data:0.2854  loss:2.2424(2.2424)
prec@1:58.59(58.59)
Epoch[9]:[051/196] Time:0.3098 Data:0.2867  loss:2.3300(2.4051)
prec@1:48.05(50.02)
Epoch[9]:[101/196] Time:0.3134 Data:0.2901  loss:2.3846(2.4097)
prec@1:50.39(49.91)
Epoch[9]:[151/196] Time:0.3083 Data:0.2833  loss:2.5010(2.4095)
prec@1:48.44(49.97)
Epoch time: 60s
Saving models…
Epoch[10]:[001/196] Time:0.3109 Data:0.2873  loss:2.3190(2.3190)
prec@1:54.69(54.69)
Epoch[10]:[051/196] Time:0.3113 Data:0.2882  loss:2.2252(2.3129)
prec@1:53.91(53.14)
Epoch[10]:[101/196] Time:0.3046 Data:0.2821  loss:2.4425(2.3019)
prec@1:46.88(53.15)
Epoch[10]:[151/196] Time:0.3101 Data:0.2874  loss:2.3194(2.2990)
prec@1:53.91(53.37)
Epoch time: 60s
Saving models…
Epoch[11]:[001/196] Time:0.3115 Data:0.2883  loss:2.1174(2.1174)
prec@1:57.81(57.81)
Epoch[11]:[051/196] Time:0.3097 Data:0.2873  loss:2.1909(2.1997)
prec@1:53.91(56.35)
Epoch[11]:[101/196] Time:0.3073 Data:0.2842  loss:2.2412(2.1994)
prec@1:51.17(56.37)
Epoch[11]:[151/196] Time:0.3043 Data:0.2818  loss:2.1086(2.1993)
prec@1:60.55(56.45)
Epoch time: 60s
Saving models…
Epoch[12]:[001/196] Time:0.3097 Data:0.2863  loss:2.1067(2.1067)
prec@1:61.72(61.72)
Epoch[12]:[051/196] Time:0.3101 Data:0.2876  loss:2.1718(2.0931)
prec@1:58.59(60.19)
```

```
Epoch[12]:[101/196] Time:0.3085 Data:0.2858  loss:2.1203(2.0972)
prec@1:56.25(59.86)
Epoch[12]:[151/196] Time:0.3257 Data:0.3006  loss:2.1336(2.1039)
prec@1:61.33(59.66)
Epoch time: 60s
Saving models…
Epoch[13]:[001/196] Time:0.3053 Data:0.2822  loss:2.0793(2.0793)
prec@1:62.11(62.11)
Epoch[13]:[051/196] Time:0.3079 Data:0.2853  loss:2.0285(1.9933)
prec@1:63.28(63.20)
Epoch[13]:[101/196] Time:0.3108 Data:0.2879  loss:1.9929(1.9991)
prec@1:62.89(62.74)
Epoch[13]:[151/196] Time:0.3097 Data:0.2845  loss:2.0003(2.0046)
prec@1:64.45(62.71)
Epoch time: 61s
Saving models…
Epoch[14]:[001/196] Time:0.3316 Data:0.3091  loss:1.8306(1.8306)
prec@1:66.41(66.41)
Epoch[14]:[051/196] Time:0.3277 Data:0.3021  loss:1.9522(1.8997)
prec@1:64.06(65.63)
Epoch[14]:[101/196] Time:0.3124 Data:0.2894  loss:1.8891(1.9118)
prec@1:63.67(65.47)
Epoch[14]:[151/196] Time:0.3084 Data:0.2852  loss:1.9223(1.9226)
prec@1:66.80(65.08)
Epoch time: 62s
Saving models…
Epoch[15]:[001/196] Time:0.3206 Data:0.2972  loss:1.9300(1.9300)
prec@1:67.19(67.19)
Epoch[15]:[051/196] Time:0.3092 Data:0.2867  loss:1.8662(1.8180)
prec@1:71.09(69.08)
Epoch[15]:[101/196] Time:0.3100 Data:0.2870  loss:1.8843(1.8339)
prec@1:64.84(68.36)
Epoch[15]:[151/196] Time:0.3086 Data:0.2817  loss:1.7728(1.8333)
prec@1:69.92(68.32)
Epoch time: 61s
Saving models…
Epoch[16]:[001/196] Time:0.3183 Data:0.2939  loss:1.7016(1.7016)
prec@1:71.88(71.88)
Epoch[16]:[051/196] Time:0.3004 Data:0.2804  loss:1.9527(1.7339)
prec@1:67.97(71.97)
Epoch[16]:[101/196] Time:0.3018 Data:0.2790  loss:1.6576(1.7335)
prec@1:72.27(71.72)
Epoch[16]:[151/196] Time:0.3063 Data:0.2832  loss:1.8103(1.7341)
prec@1:67.97(71.71)
Epoch time: 59s
Saving models…
Epoch[17]:[001/196] Time:0.3056 Data:0.2824  loss:1.7197(1.7197)
prec@1:73.44(73.44)
```

```
Epoch[17]:[051/196] Time:0.2993 Data:0.2766  loss:1.5505(1.6252)
prec@1:77.73(75.83)
Epoch[17]:[101/196] Time:0.3007 Data:0.2783  loss:1.5952(1.6328)
prec@1:78.91(75.36)
Epoch[17]:[151/196] Time:0.2994 Data:0.2770  loss:1.7587(1.6329)
prec@1:66.80(75.34)
Epoch time: 59s
Saving models…
Epoch[18]:[001/196] Time:0.3460 Data:0.3228  loss:1.5654(1.5654)
prec@1:78.91(78.91)
Epoch[18]:[051/196] Time:0.3439 Data:0.3209  loss:1.6656(1.5477)
prec@1:76.17(78.31)
Epoch[18]:[101/196] Time:0.2988 Data:0.2762  loss:1.5646(1.5488)
prec@1:78.12(78.21)
Epoch[18]:[151/196] Time:0.3047 Data:0.2818  loss:1.5762(1.5566)
prec@1:75.39(77.88)
Epoch time: 59s
Saving models…
Epoch[19]:[001/196] Time:0.3024 Data:0.2793  loss:1.3723(1.3723)
prec@1:83.59(83.59)
Epoch[19]:[051/196] Time:0.2995 Data:0.2749  loss:1.3780(1.4551)
prec@1:81.64(81.23)
Epoch[19]:[101/196] Time:0.3059 Data:0.2812  loss:1.4044(1.4578)
prec@1:82.81(81.22)
Epoch[19]:[151/196] Time:0.2972 Data:0.2745  loss:1.4348(1.4653)
prec@1:79.30(81.03)
Epoch time: 58s
Saving models…
Epoch[20]:[001/196] Time:0.3070 Data:0.2837  loss:1.3693(1.3693)
prec@1:83.59(83.59)
Epoch[20]:[051/196] Time:0.3000 Data:0.2774  loss:1.3121(1.3756)
prec@1:83.98(84.24)
Epoch[20]:[101/196] Time:0.2959 Data:0.2733  loss:1.4133(1.3827)
prec@1:82.42(83.95)
Epoch[20]:[151/196] Time:0.3390 Data:0.3161  loss:1.3941(1.3922)
prec@1:83.59(83.66)
Epoch time: 61s
Saving models…
Epoch[21]:[001/196] Time:0.3198 Data:0.2973  loss:1.3584(1.3584)
prec@1:84.38(84.38)
Epoch[21]:[051/196] Time:0.3262 Data:0.3038  loss:1.2825(1.3194)
prec@1:87.50(86.59)
Epoch[21]:[101/196] Time:0.3069 Data:0.2839  loss:1.3741(1.3152)
prec@1:82.42(86.53)
Epoch[21]:[151/196] Time:0.3036 Data:0.2811  loss:1.4032(1.3189)
prec@1:84.38(86.30)
Epoch time: 60s
Saving models…
```

```
Epoch[22]:[001/196] Time:0.3035 Data:0.2804  loss:1.2210(1.2210)
prec@1:90.62(90.62)
Epoch[22]:[051/196] Time:0.2995 Data:0.2766  loss:1.3014(1.2627)
prec@1:87.50(88.20)
Epoch[22]:[101/196] Time:0.3067 Data:0.2838  loss:1.2661(1.2578)
prec@1:87.89(88.54)
Epoch[22]:[151/196] Time:0.2992 Data:0.2767  loss:1.2450(1.2568)
prec@1:89.45(88.48)
Epoch time: 59s
Saving models…
Epoch[23]:[001/196] Time:0.3048 Data:0.2815  loss:1.2166(1.2166)
prec@1:91.02(91.02)
Epoch[23]:[051/196] Time:0.3006 Data:0.2782  loss:1.2112(1.1990)
prec@1:90.23(90.92)
Epoch[23]:[101/196] Time:0.3032 Data:0.2809  loss:1.1407(1.1928)
prec@1:92.19(90.84)
Epoch[23]:[151/196] Time:0.3100 Data:0.2868  loss:1.1756(1.1986)
prec@1:92.19(90.63)
Epoch time: 59s
Saving models…
Epoch[24]:[001/196] Time:0.3048 Data:0.2817  loss:1.1549(1.1549)
prec@1:92.58(92.58)
Epoch[24]:[051/196] Time:0.3049 Data:0.2824  loss:1.1377(1.1494)
prec@1:93.36(92.29)
Epoch[24]:[101/196] Time:0.3001 Data:0.2775  loss:1.1504(1.1479)
prec@1:92.97(92.38)
Epoch[24]:[151/196] Time:0.3031 Data:0.2805  loss:1.2052(1.1498)
prec@1:90.62(92.19)
Epoch time: 59s
Saving models…
Epoch[25]:[001/196] Time:0.3049 Data:0.2816  loss:1.0683(1.0683)
prec@1:95.70(95.70)
Epoch[25]:[051/196] Time:0.3198 Data:0.2973  loss:1.0945(1.1056)
prec@1:94.53(93.76)
Epoch[25]:[101/196] Time:0.3154 Data:0.2931  loss:1.1278(1.1029)
prec@1:92.97(93.86)
Epoch[25]:[151/196] Time:0.3187 Data:0.2942  loss:1.0931(1.1070)
prec@1:92.58(93.70)
Epoch time: 62s
Saving models…
Epoch[26]:[001/196] Time:0.3267 Data:0.3044  loss:1.0728(1.0728)
prec@1:95.31(95.31)
Epoch[26]:[051/196] Time:0.3207 Data:0.2985  loss:1.1119(1.0668)
prec@1:92.97(95.26)
Epoch[26]:[101/196] Time:0.3094 Data:0.2858  loss:1.0693(1.0695)
prec@1:94.92(95.03)
Epoch[26]:[151/196] Time:0.3115 Data:0.2884  loss:1.1102(1.0684)
prec@1:92.58(95.13)
```

```
Epoch time: 63s
Saving models…
Epoch[27]:[001/196] Time:0.3564 Data:0.3328  loss:1.0453(1.0453)
prec@1:95.31(95.31)
Epoch[27]:[051/196] Time:0.3146 Data:0.2910  loss:1.0079(1.0373)
prec@1:98.05(96.21)
Epoch[27]:[101/196] Time:0.3064 Data:0.2832  loss:1.0224(1.0352)
prec@1:96.88(96.27)
Epoch[27]:[151/196] Time:0.3183 Data:0.2958  loss:1.0084(1.0388)
prec@1:96.88(96.20)
Epoch time: 62s
Saving models…
Epoch[28]:[001/196] Time:0.3414 Data:0.3189  loss:1.0132(1.0132)
prec@1:97.27(97.27)
Epoch[28]:[051/196] Time:0.3167 Data:0.2920  loss:1.0017(1.0121)
prec@1:97.66(97.10)
Epoch[28]:[101/196] Time:0.3175 Data:0.2946  loss:1.0041(1.0117)
prec@1:97.27(96.98)
Epoch[28]:[151/196] Time:0.3169 Data:0.2945  loss:1.0113(1.0127)
prec@1:97.66(96.86)
Epoch time: 62s
Saving models…
Epoch[29]:[001/196] Time:0.3199 Data:0.2970  loss:1.0198(1.0198)
prec@1:98.05(98.05)
Epoch[29]:[051/196] Time:0.3135 Data:0.2910  loss:0.9600(0.9866)
prec@1:98.83(97.93)
Epoch[29]:[101/196] Time:0.3257 Data:0.3034  loss:1.0011(0.9896)
prec@1:97.66(97.74)
Epoch[29]:[151/196] Time:0.3221 Data:0.3000  loss:1.0235(0.9892)
prec@1:96.09(97.76)
Epoch time: 63s
Saving models…
Epoch[30]:[001/196] Time:0.3281 Data:0.3059  loss:0.9501(0.9501)
prec@1:98.05(98.05)
Epoch[30]:[051/196] Time:0.3177 Data:0.2938  loss:0.9637(0.9763)
prec@1:98.44(98.12)
Epoch[30]:[101/196] Time:0.3170 Data:0.2944  loss:0.9772(0.9747)
prec@1:97.27(98.18)
Epoch[30]:[151/196] Time:0.3277 Data:0.3051  loss:0.9906(0.9756)
prec@1:98.44(98.06)
Epoch time: 64s
Saving models…
Epoch[31]:[001/196] Time:0.3216 Data:0.2995  loss:0.9670(0.9670)
prec@1:98.44(98.44)
Epoch[31]:[051/196] Time:0.3537 Data:0.3317  loss:0.9732(0.9604)
prec@1:98.05(98.44)
Epoch[31]:[101/196] Time:0.3167 Data:0.2929  loss:0.9290(0.9587)
prec@1:99.61(98.51)
```

```
Epoch[31]:[151/196] Time:0.3143 Data:0.2914  loss:0.9696(0.9592)
prec@1:98.05(98.52)
Epoch time: 65s
Saving models…
Epoch[32]:[001/196] Time:0.3496 Data:0.3274  loss:0.9542(0.9542)
prec@1:98.44(98.44)
Epoch[32]:[051/196] Time:0.3200 Data:0.2981  loss:0.9394(0.9453)
prec@1:98.83(98.70)
Epoch[32]:[101/196] Time:0.3136 Data:0.2917  loss:0.9191(0.9432)
prec@1:99.22(98.75)
Epoch[32]:[151/196] Time:0.3058 Data:0.2838  loss:0.9264(0.9451)
prec@1:99.61(98.76)
Epoch time: 63s
Saving models…
Epoch[33]:[001/196] Time:0.3268 Data:0.3042  loss:0.9329(0.9329)
prec@1:98.83(98.83)
Epoch[33]:[051/196] Time:0.3142 Data:0.2922  loss:0.9299(0.9378)
prec@1:98.44(98.95)
Epoch[33]:[101/196] Time:0.3451 Data:0.3232  loss:0.9216(0.9353)
prec@1:99.61(99.06)
Epoch[33]:[151/196] Time:0.3443 Data:0.3224  loss:0.9391(0.9356)
prec@1:98.05(99.05)
Epoch time: 63s
Saving models…
Epoch[34]:[001/196] Time:0.3087 Data:0.2867  loss:0.9340(0.9340)
prec@1:99.22(99.22)
Epoch[34]:[051/196] Time:0.3036 Data:0.2816  loss:0.9103(0.9275)
prec@1:98.83(99.19)
Epoch[34]:[101/196] Time:0.3151 Data:0.2912  loss:0.9289(0.9271)
prec@1:99.22(99.16)
Epoch[34]:[151/196] Time:0.3234 Data:0.3013  loss:0.9308(0.9269)
prec@1:99.22(99.16)
Epoch time: 63s
Saving models…
Epoch[35]:[001/196] Time:0.3176 Data:0.2956  loss:0.9086(0.9086)
prec@1:100.00(100.00)
Epoch[35]:[051/196] Time:0.3161 Data:0.2941  loss:0.9145(0.9216)
prec@1:99.61(99.17)
Epoch[35]:[101/196] Time:0.3133 Data:0.2912  loss:0.9159(0.9202)
prec@1:99.61(99.26)
Epoch[35]:[151/196] Time:0.3445 Data:0.3226  loss:0.9412(0.9187)
prec@1:97.66(99.30)
Epoch time: 62s
Saving models…
Epoch[36]:[001/196] Time:0.3151 Data:0.2931  loss:0.8959(0.8959)
prec@1:100.00(100.00)
Epoch[36]:[051/196] Time:0.3453 Data:0.3234  loss:0.9137(0.9121)
prec@1:99.61(99.48)
```

```
Epoch[36]:[101/196] Time:0.3082 Data:0.2843  loss:0.9140(0.9111)
prec@1:99.61(99.46)
Epoch[36]:[151/196] Time:0.3161 Data:0.2941  loss:0.9365(0.9108)
prec@1:98.44(99.44)
Epoch time: 61s
Saving models…
Epoch[37]:[001/196] Time:0.3088 Data:0.2868  loss:0.9000(0.9000)
prec@1:99.61(99.61)
Epoch[37]:[051/196] Time:0.3091 Data:0.2872  loss:0.9047(0.9039)
prec@1:99.22(99.68)
Epoch[37]:[101/196] Time:0.3089 Data:0.2839  loss:0.8939(0.9058)
prec@1:100.00(99.59)
Epoch[37]:[151/196] Time:0.3071 Data:0.2820  loss:0.8915(0.9065)
prec@1:99.61(99.56)
Epoch time: 60s
Saving models…
Epoch[38]:[001/196] Time:0.3783 Data:0.3537  loss:0.9061(0.9061)
prec@1:99.22(99.22)
Epoch[38]:[051/196] Time:0.3100 Data:0.2880  loss:0.9107(0.9010)
prec@1:100.00(99.63)
Epoch[38]:[101/196] Time:0.3064 Data:0.2821  loss:0.9051(0.9016)
prec@1:99.22(99.66)
Epoch[38]:[151/196] Time:0.3082 Data:0.2862  loss:0.9080(0.9015)
prec@1:99.61(99.66)
Epoch time: 60s
Saving models…
Epoch[39]:[001/196] Time:0.3373 Data:0.3150  loss:0.9056(0.9056)
prec@1:100.00(100.00)
Epoch[39]:[051/196] Time:0.3087 Data:0.2868  loss:0.9116(0.8980)
prec@1:99.22(99.67)
Epoch[39]:[101/196] Time:0.3111 Data:0.2891  loss:0.8843(0.8981)
prec@1:99.22(99.66)
Epoch[39]:[151/196] Time:0.3191 Data:0.2971  loss:0.8887(0.8978)
prec@1:100.00(99.64)
Epoch time: 60s
Saving models…
Epoch[40]:[001/196] Time:0.3078 Data:0.2858  loss:0.8916(0.8916)
prec@1:100.00(100.00)
Epoch[40]:[051/196] Time:0.3058 Data:0.2807  loss:0.8806(0.8934)
prec@1:100.00(99.75)
Epoch[40]:[101/196] Time:0.3059 Data:0.2840  loss:0.8959(0.8931)
prec@1:100.00(99.73)
Epoch[40]:[151/196] Time:0.3087 Data:0.2867  loss:0.9043(0.8930)
prec@1:99.61(99.74)
Epoch time: 61s
Saving models…
Epoch[41]:[001/196] Time:0.3261 Data:0.3038  loss:0.8821(0.8821)
prec@1:100.00(100.00)
```

```
Epoch[41]:[051/196] Time:0.3117 Data:0.2897  loss:0.9036(0.8916)
prec@1:99.61(99.72)
Epoch[41]:[101/196] Time:0.3131 Data:0.2912  loss:0.8756(0.8913)
prec@1:100.00(99.74)
Epoch[41]:[151/196] Time:0.3194 Data:0.2975  loss:0.8739(0.8906)
prec@1:99.61(99.75)
Epoch time: 61s
Saving models…
Epoch[42]:[001/196] Time:0.3172 Data:0.2952  loss:0.8898(0.8898)
prec@1:99.61(99.61)
Epoch[42]:[051/196] Time:0.3128 Data:0.2905  loss:0.8971(0.8875)
prec@1:99.61(99.80)
Epoch[42]:[101/196] Time:0.3130 Data:0.2911  loss:0.8787(0.8858)
prec@1:100.00(99.80)
Epoch[42]:[151/196] Time:0.3184 Data:0.2965  loss:0.8792(0.8864)
prec@1:100.00(99.81)
Epoch time: 61s
Saving models…
Epoch[43]:[001/196] Time:0.3131 Data:0.2910  loss:0.8827(0.8827)
prec@1:100.00(100.00)
Epoch[43]:[051/196] Time:0.3164 Data:0.2944  loss:0.8660(0.8839)
prec@1:100.00(99.91)
Epoch[43]:[101/196] Time:0.3139 Data:0.2918  loss:0.8820(0.8847)
prec@1:100.00(99.88)
Epoch[43]:[151/196] Time:0.3086 Data:0.2866  loss:0.8797(0.8844)
prec@1:100.00(99.85)
Epoch time: 60s
Saving models…
Epoch[44]:[001/196] Time:0.3116 Data:0.2895  loss:0.8898(0.8898)
prec@1:99.61(99.61)
Epoch[44]:[051/196] Time:0.3069 Data:0.2850  loss:0.8784(0.8834)
prec@1:100.00(99.83)
Epoch[44]:[101/196] Time:0.3132 Data:0.2891  loss:0.8747(0.8828)
prec@1:100.00(99.85)
Epoch[44]:[151/196] Time:0.3078 Data:0.2856  loss:0.8882(0.8826)
prec@1:99.61(99.86)
Epoch time: 61s
Saving models…
Epoch[45]:[001/196] Time:0.3141 Data:0.2897  loss:0.8732(0.8732)
prec@1:100.00(100.00)
Epoch[45]:[051/196] Time:0.3430 Data:0.3209  loss:0.8739(0.8820)
prec@1:100.00(99.81)
Epoch[45]:[101/196] Time:0.3442 Data:0.3221  loss:0.8791(0.8821)
prec@1:100.00(99.82)
Epoch[45]:[151/196] Time:0.3431 Data:0.3210  loss:0.8912(0.8817)
prec@1:99.61(99.83)
Epoch time: 63s
Saving models…
```

```
Epoch[46]:[001/196] Time:0.3089 Data:0.2869  loss:0.8727(0.8727)
prec@1:100.00(100.00)
Epoch[46]:[051/196] Time:0.3070 Data:0.2850  loss:0.8761(0.8789)
prec@1:100.00(99.89)
Epoch[46]:[101/196] Time:0.3458 Data:0.3239  loss:0.8727(0.8790)
prec@1:100.00(99.88)
Epoch[46]:[151/196] Time:0.3086 Data:0.2865  loss:0.8852(0.8792)
prec@1:99.61(99.87)
Epoch time: 61s
Saving models…
Epoch[47]:[001/196] Time:0.3322 Data:0.3101  loss:0.8803(0.8803)
prec@1:100.00(100.00)
Epoch[47]:[051/196] Time:0.3152 Data:0.2929  loss:0.8829(0.8811)
prec@1:100.00(99.83)
Epoch[47]:[101/196] Time:0.3136 Data:0.2895  loss:0.8717(0.8799)
prec@1:100.00(99.86)
Epoch[47]:[151/196] Time:0.3102 Data:0.2877  loss:0.8790(0.8800)
prec@1:100.00(99.86)
Epoch time: 61s
Saving models…
Epoch[48]:[001/196] Time:0.3188 Data:0.2965  loss:0.8820(0.8820)
prec@1:100.00(100.00)
Epoch[48]:[051/196] Time:0.3091 Data:0.2872  loss:0.8743(0.8796)
prec@1:100.00(99.87)
Epoch[48]:[101/196] Time:0.3143 Data:0.2924  loss:0.8697(0.8789)
prec@1:100.00(99.90)
Epoch[48]:[151/196] Time:0.3308 Data:0.3089  loss:0.8819(0.8789)
prec@1:100.00(99.88)
Epoch time: 61s
Saving models…
Epoch[49]:[001/196] Time:0.3083 Data:0.2863  loss:0.8769(0.8769)
prec@1:100.00(100.00)
Epoch[49]:[051/196] Time:0.3131 Data:0.2912  loss:0.8826(0.8783)
prec@1:100.00(99.86)
Epoch[49]:[101/196] Time:0.3229 Data:0.3002  loss:0.8719(0.8785)
prec@1:100.00(99.86)
Epoch[49]:[151/196] Time:0.3061 Data:0.2842  loss:0.8658(0.8784)
prec@1:100.00(99.87)
Epoch time: 61s
Saving models…
Epoch[50]:[001/196] Time:0.3531 Data:0.3311  loss:0.8816(0.8816)
prec@1:100.00(100.00)
Epoch[50]:[051/196] Time:0.3225 Data:0.2984  loss:0.8794(0.8777)
prec@1:100.00(99.92)
Epoch[50]:[101/196] Time:0.3095 Data:0.2874  loss:0.8811(0.8795)
prec@1:100.00(99.87)
Epoch[50]:[151/196] Time:0.3101 Data:0.2882  loss:0.8687(0.8786)
prec@1:100.00(99.88)
```

```
Epoch time: 61s
Saving models…
```

5 Step 5: Test on single image

[6]:
```
img = Image.open("./cifar100/test/apple_9904.jpg")
# img = transform_test(img).unsqueeze(0).to(device)
img_rgb = transform_test(img)
img_grad = test_dataset.RGB2Gradient(img_rgb)

img_rgb = img_rgb.unsqueeze(0).to(device)
img_grad = img_grad.unsqueeze(0).to(device)

out = net(img_rgb, img_grad)
predicted_classes = torch.argmax(out, dim=1)
tags = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee',␣
 ↪'beetle', 'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly',␣
 ↪'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee',␣
 ↪'clock', 'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup',␣
 ↪'dinosaur', 'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl',␣
 ↪'hamster', 'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard',␣
 ↪'lion', 'lizard', 'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain',␣
 ↪'mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree',␣
 ↪'pear', 'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',␣
 ↪'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose', 'sea',␣
 ↪'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake', 'spider',␣
 ↪'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table', 'tank',␣
 ↪'telephone', 'television', 'tiger', 'tractor', 'train', 'trout', 'tulip',␣
 ↪'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman', 'worm']

print(tags[predicted_classes[0]])
```

```
train
```

6 Step 6: Evaluate model accuracy

[7]:
```
def test(test_loader, net, criterion):
    losses = AverageMeter()
    top1   = AverageMeter()

    net.eval()

    for i, (img_rgb, img_grad, target) in enumerate(test_loader, start=1):
        img_rgb = img_rgb.to(device)
        img_grad = img_grad.to(device)
        target = target.to(device)
```

```
        with torch.no_grad():
            out = net(img_rgb, img_grad)
            loss = criterion(out, target)

        prec1 = accuracy(out, target) # prec1:list
        losses.update(loss.item(), img_rgb.size(0))
        top1.update(prec1.item(), img_rgb.size(0))

    f_l = [losses.avg, top1.avg]
    print('----------------------------------test classification␣
 ↪result----------------------------------')
    print('Loss: {:.4f}, Prec@1: {:.2f}%'.format(*f_l))

    return top1.avg

test_top1 = test(test_loader, net, criterion)
```

```
----------------------------------test classification
result----------------------------------
Loss: 2.6205, Prec@1: 54.23%
```

7 Step 7: T-SNE Visualization

Use hooks in PyTorch to extract feature representations from the intermediate layers of the model for the test set "testloader", and visualize them using the T-SNE method. The specific requirements are as follows:

Visualize the features before and after the dual-branch feature fusion. If there are multiple fusions, you may choose specific layers for visualization.

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
import torch.nn.functional as F

# 1.
net.eval()
device = torch.device("cuda:0" if torch.cuda.is_available() else "mps" if torch.
 ↪backends.mps.is_available() else "cpu")

# 2.
features = {
    'rgb_branch': [],
    'grad_branch': [],
    'after_fusion': []
}
```

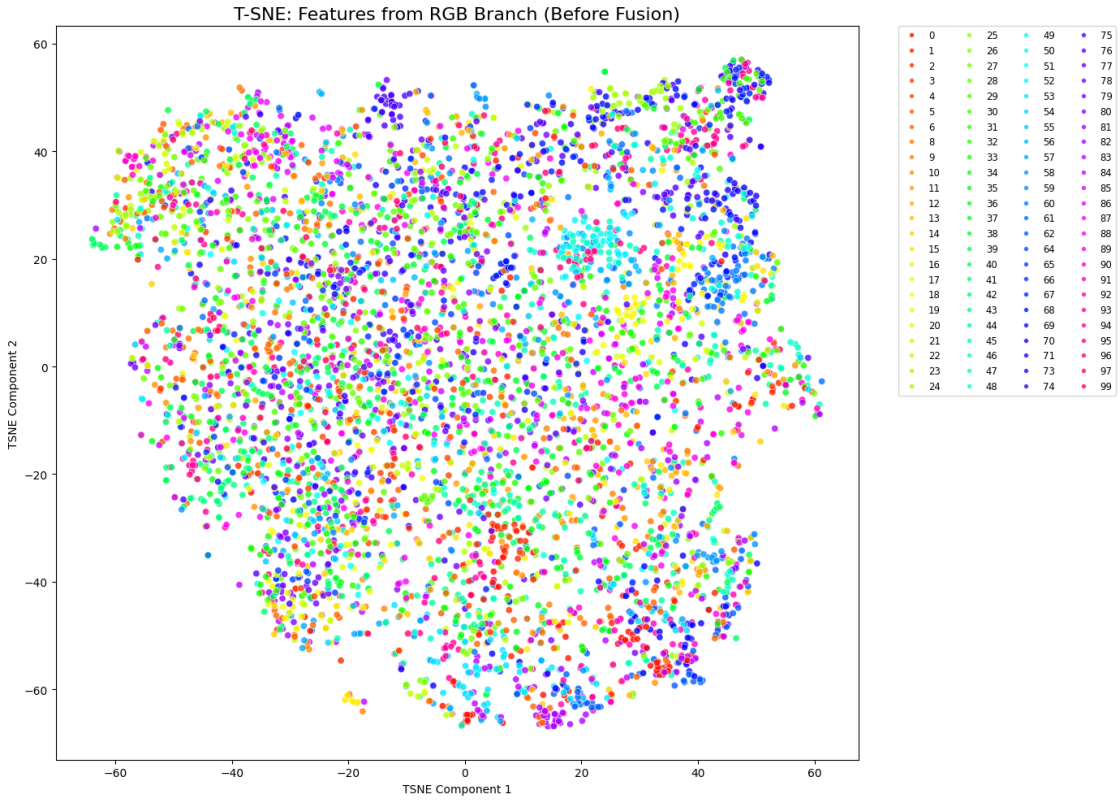
```
all_labels = []

# 3.   PyTorch Hooks
def pool_and_flatten(module_output):
    # 'output'    [B, C, H, W]
    #       [B, C]
    pooled = F.adaptive_avg_pool2d(module_output, (1, 1))
    flattened = torch.flatten(pooled, 1)
    return flattened.detach().cpu().numpy()

def hook_rgb(module, input, output):
    # "Before" - RGB   ( layer3_rgb  ,   )
    # output shape: [B, 256, H, W]
    features['rgb_branch'].append(pool_and_flatten(output))

def hook_grad(module, input, output):
    # "Before" - Gradient   ( layer3_grad  ,   )
    # output shape: [B, 256, H, W]
    features['grad_branch'].append(pool_and_flatten(output))

def hook_after(module, input, output):
    # "After" -    ( fc_fused  )
    # output shape: [B, 512]
    features['after_fusion'].append(output.detach().cpu().numpy())

# 4.   hooks
#   1:   'layer3_rgb' ,
hook_handle_rgb = net.layer3_rgb.register_forward_hook(hook_rgb)
#   2:   'layer3_grad' ,
hook_handle_grad = net.layer3_grad.register_forward_hook(hook_grad)
#   3:   'fc_fused' ,     ( )
hook_handle_after = net.fc_fused.register_forward_hook(hook_after)

# 5.   test_loader
print(f"Extracting features from test_loader...")
with torch.no_grad():
    for i, (img_rgb, img_grad, target) in enumerate(test_loader):
        img_rgb = img_rgb.to(device)
        img_grad = img_grad.to(device)
        target = target.to(device)

        #       hooks
        _ = net(img_rgb, img_grad)

        #
        all_labels.append(target.cpu().numpy())
```

```
        #          batch (  20-30  )
        if i > 20:
             print(f"Extracted features from {i+1} batches.")
             break

# 6.   hooks
hook_handle_rgb.remove()
hook_handle_grad.remove()
hook_handle_after.remove()
print("Feature extraction complete. Hooks removed.")

# 7.
features_rgb = np.concatenate(features['rgb_branch'], axis=0)
features_grad = np.concatenate(features['grad_branch'], axis=0)
features_after = np.concatenate(features['after_fusion'], axis=0)
labels_arr = np.concatenate(all_labels, axis=0)

print(f"Total features extracted: {features_rgb.shape[0]}")
#     features_rgb   features_grad   [N, 256]
# features_after   [N, 512]
# T-SNE

# 8. ( )   T-SNE          N
#        20   batch
tsne_features_rgb = features_rgb
tsne_features_grad = features_grad
tsne_features_after = features_after
tsne_labels = labels_arr

# 9.   T-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42, n_jobs=-1)

print("Running T-SNE for 'RGB Branch (Before)' features...")
tsne_results_rgb = tsne.fit_transform(tsne_features_rgb)

print("Running T-SNE for 'Gradient Branch (Before)' features...")
tsne_results_grad = tsne.fit_transform(tsne_features_grad)

print("Running T-SNE for 'After Fusion' features...")
tsne_results_after = tsne.fit_transform(tsne_features_after)

print("T-SNE computation complete.")

# 10.
def plot_tsne(tsne_results, labels, title):
    plt.figure(figsize=(14, 10))
    sns.scatterplot(
```

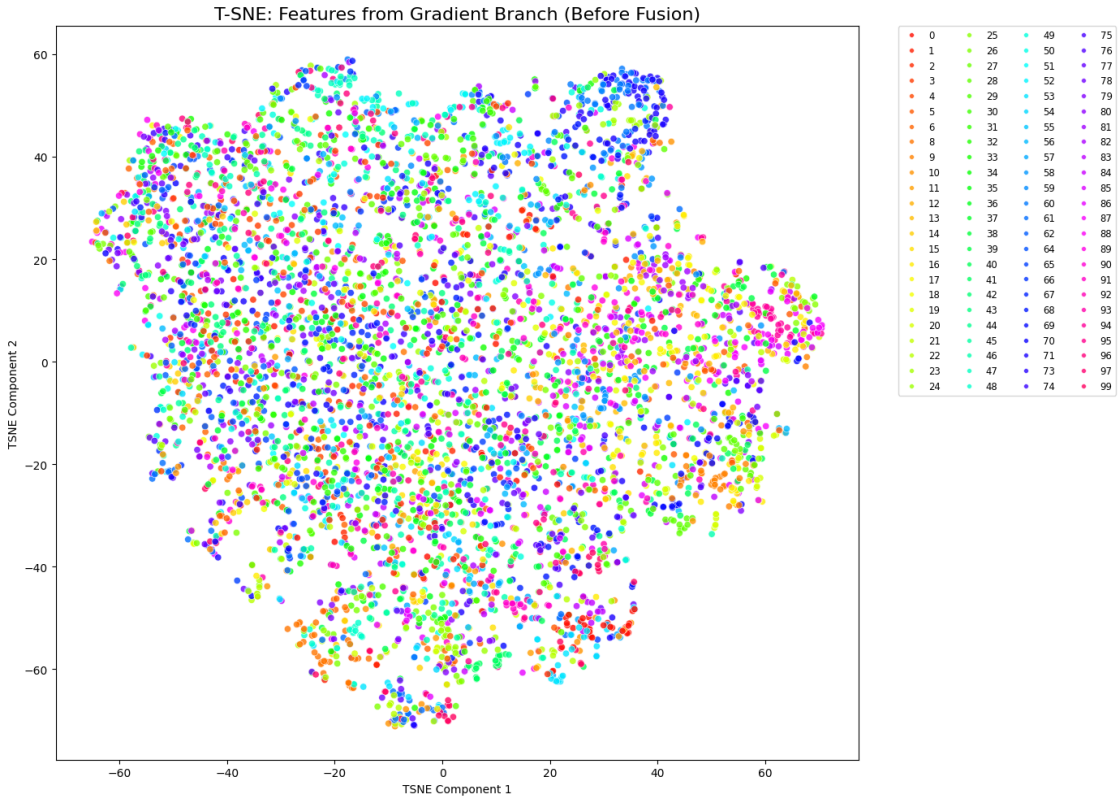
```
        x=tsne_results[:, 0],
        y=tsne_results[:, 1],
        hue=labels,
        palette=sns.color_palette("hsv", n_colors=100), # CIFAR-100
        legend="full",
        alpha=0.8
    )
    plt.title(title, fontsize=16)
    plt.xlabel("TSNE Component 1")
    plt.ylabel("TSNE Component 2")
    #
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., markerscale=0.
 ↪7, fontsize='small', ncol=4)
    plt.tight_layout()
    #      notebook
    # plt.savefig(title.replace(" ", "_") + ".png")
    plt.show() #   .ipynb  , plt.show()

# 11.    T-SNE
plot_tsne(tsne_results_rgb, tsne_labels, "T-SNE: Features from RGB Branch␣
 ↪(Before Fusion)")
plot_tsne(tsne_results_grad, tsne_labels, "T-SNE: Features from Gradient Branch␣
 ↪(Before Fusion)")
plot_tsne(tsne_results_after, tsne_labels, "T-SNE: Features After Fusion (at␣
 ↪fc_fused output)")
```

```
Extracting features from test_loader…
Extracted features from 22 batches.
Feature extraction complete. Hooks removed.
Total features extracted: 5632
Running T-SNE for 'RGB Branch (Before)' features…
Running T-SNE for 'Gradient Branch (Before)' features…
Running T-SNE for 'After Fusion' features…
T-SNE computation complete.
```

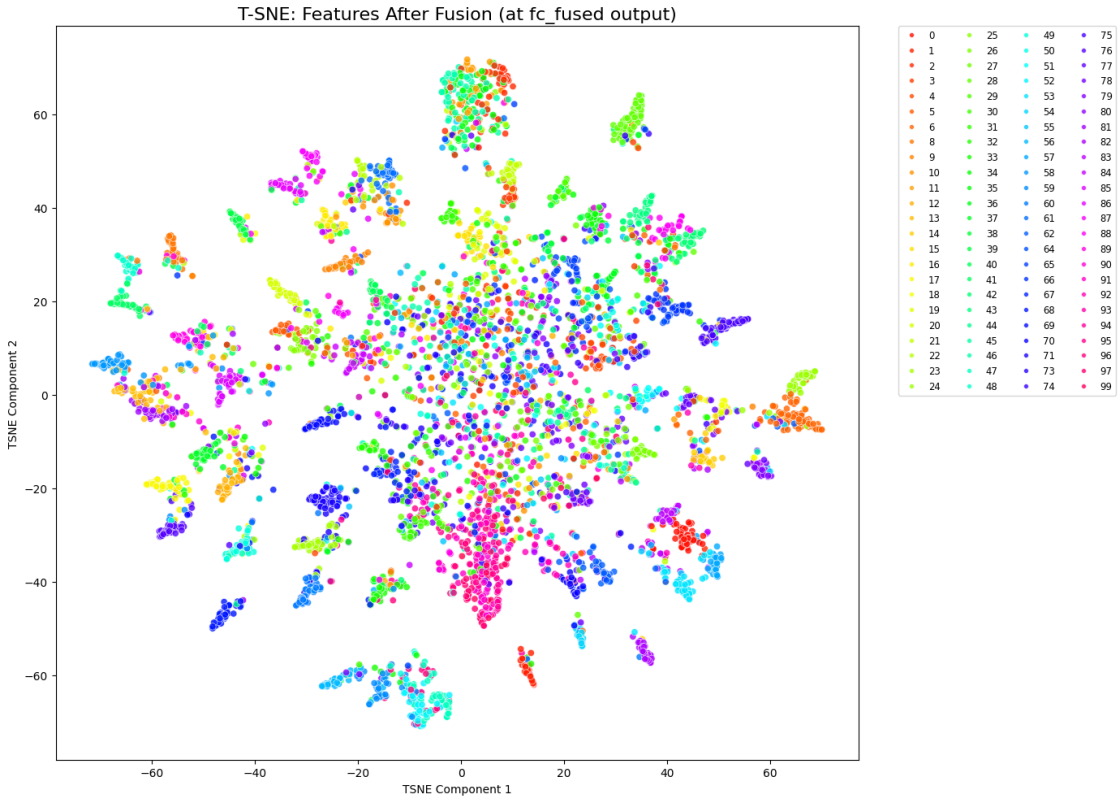
```
/tmp/ipykernel_2259411/977395255.py:110: UserWarning: The palette list has more
values (100) than needed (96), which may not be intended.
  sns.scatterplot(
```




```
/tmp/ipykernel_2259411/977395255.py:110: UserWarning: The palette list has more
values (100) than needed (96), which may not be intended.
  sns.scatterplot(
```




```
/tmp/ipykernel_2259411/977395255.py:110: UserWarning: The palette list has more
values (100) than needed (96), which may not be intended.
  sns.scatterplot(
```


T-SNE: Features After Fusion (at fc_fused output)
TSNE Component 1
TSNE Component 2
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
0 1 2 3 4 5 6 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58 59 60 61 62 64 65 66 67 68 69 70 71 73 74
75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 99