MobileNetV2 — 垃圾分类

3210104591 秦雨扬

实验内容

深度学习计算中，从头开始训练一个实用的模型通常非常耗时，需要大量计算能力。常用的数据如OpenImage、ImageNet、VOC、COCO等公开大型数据集，规模达到几十万甚至超过上百万张。网络和开源社区上通常会提供这些数据集上预训练好的模型。大部分细分领域任务在训练网络模型时，如果不使用预训练模型而从头开始训练网络，不仅耗时，且模型容易陷入局部极小值和过拟合。因此大部分任务都会选择预训练模型，在其上做微调（也称为Fine-Tune）。

本实验以MobileNetV2+垃圾分类数据集为例，主要介绍如在使用c在CPU/GPU平台上进行Fine-Tune。

垃圾分类信息：

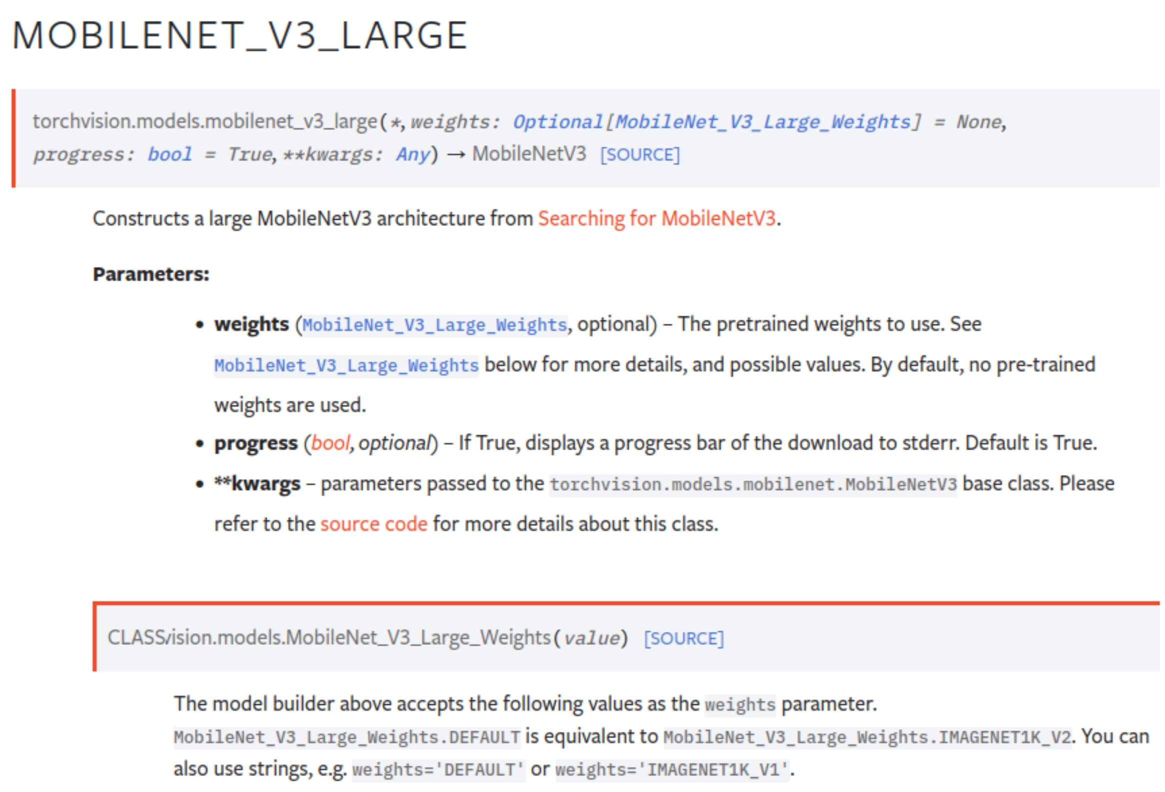
```
{
        '干垃圾': ['贝壳', '打火机', '旧镜子', '扫把', '陶瓷碗', '牙刷', '一次性筷子',
'脏污衣服'],
        '可回收物': ['报纸', '玻璃制品', '篮球', '塑料瓶', '硬纸板', '玻璃瓶', '金属制
品', '帽子', '易拉罐', '纸张'],
        '湿垃圾': ['菜叶', '橙皮', '蛋壳', '香蕉皮'],
        '有害垃圾': ['电池', '药片胶囊', '荧光灯', '油漆桶']
    }
    ['贝壳', '打火机', '旧镜子', '扫把', '陶瓷碗', '牙刷', '一次性筷子', '脏污衣服',
    '报纸', '玻璃制品', '篮球', '塑料瓶', '硬纸板', '玻璃瓶', '金属制品', '帽子', '易拉
罐', '纸张',
    '菜叶', '橙皮', '蛋壳', '香蕉皮',
    '电池', '药片胶囊', '荧光灯', '油漆桶']
    ['Seashell', 'Lighter', 'Old Mirror', 'Broom', 'Ceramic Bowl', 'Toothbrush',
'Disposable Chopsticks', 'Dirty Cloth',
    'Newspaper', 'Glassware', 'Basketball', 'Plastic Bottle', 'Cardboard', 'Glass
Bottle', 'Metalware', 'Hats', 'Cans', 'Paper',
    'Vegetable Leaf', 'Orange Peel', 'Eggshell', 'Banana Peel',
    'Battery', 'Tablet capsules', 'Fluorescent lamp', 'Paint bucket']
```

脚本、预训练模型的 Checkpoint 和数据集组织为如下形式：

```
├── main.ipynb # 入口Jupyter Notebook文件
│
├── src_mindspore
│   ├── dataset.py
│   ├── mobilenetv2.py
│   └── mobilenetv2-200_1067_gpu_cpu.ckpt
│
├── results/mobilenetv2.mindir # 待生成的MindSpore0.5.0模型文件
│
├── train_main.py # 将 main.ipynb Notebook 训练模型代码转化为py文件
└── datasets/5fbdf571c06d3433df85ac65-momodel/garbage_26x100/ # 数据集
    ├── train/
    ├── val/
    └── label.txt
```

Pytorch 实践

首先自然而言地想要用pytorch，但是莫名奇妙发现其精度非常低。然后就自然而然地想，既然V2不行，那V3怎么样？结果发现不分上下。如果backbone不允许训练，只训练head的话，训练200个epoch精度才在25~30%之间，这显然说明数据处理不对。然后第一次学会去看model card, pytorch官网如下



MOBILENET_V3_LARGE

torchvision.models.mobilenet_v3_large(*, *weights: Optional[MobileNet_V3_Large_Weights] = None*, *progress: bool = True*, ***kwargs: Any*) → MobileNetV3 [SOURCE]

Constructs a large MobileNetV3 architecture from Searching for MobileNetV3.

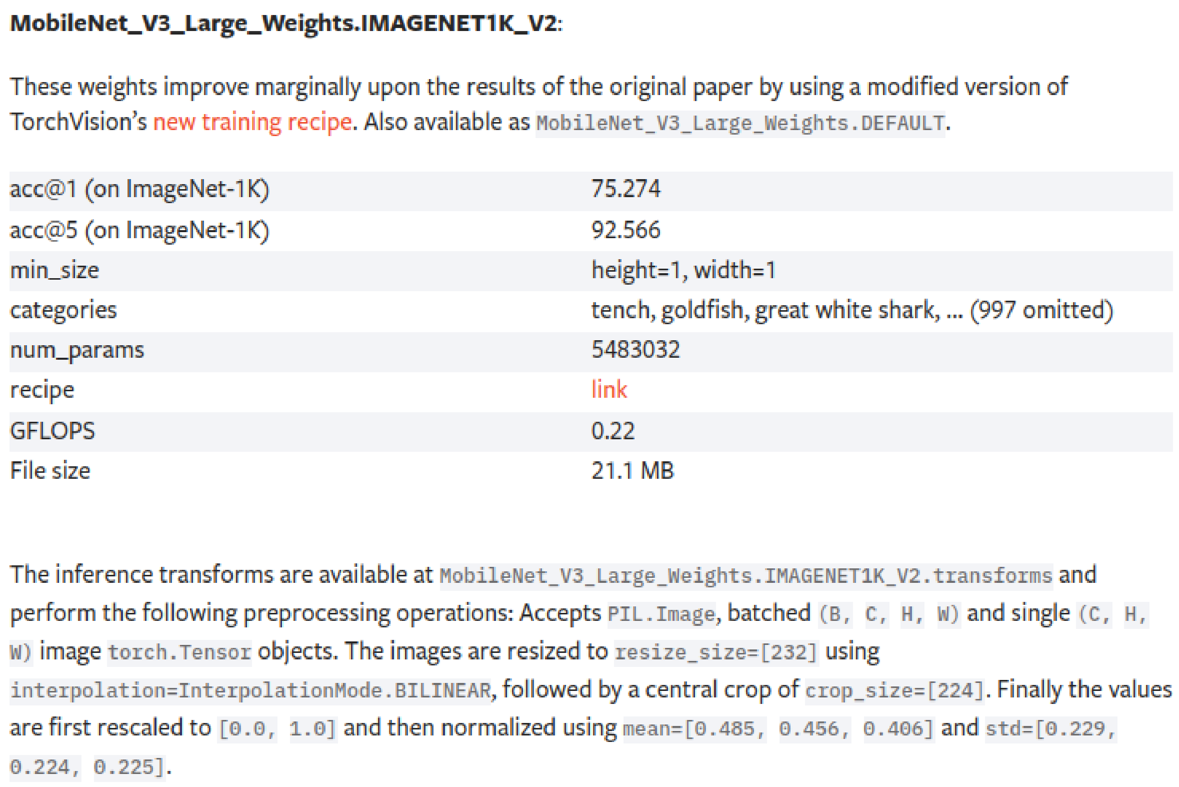
Parameters:

- **weights** (`MobileNet_V3_Large_Weights`, optional) – The pretrained weights to use. See `MobileNet_V3_Large_Weights` below for more details, and possible values. By default, no pre-trained weights are used.
- **progress** (*bool, optional*) – If True, displays a progress bar of the download to stderr. Default is True.
- ****kwargs** – parameters passed to the `torchvision.models.mobilenet.MobileNetV3` base class. Please refer to the source code for more details about this class.

CLASSvision.models.MobileNet_V3_Large_Weights(*value*) [SOURCE]

The model builder above accepts the following values as the `weights` parameter. `MobileNet_V3_Large_Weights.DEFAULT` is equivalent to `MobileNet_V3_Large_Weights.IMAGENET1K_V2`. You can also use strings, e.g. `weights='DEFAULT'` or `weights='IMAGENET1K_V1'`.

其中的权重说明如下



MobileNet_V3_Large_Weights.IMAGENET1K_V2:

These weights improve marginally upon the results of the original paper by using a modified version of TorchVision's new training recipe. Also available as `MobileNet_V3_Large_Weights.DEFAULT`.

acc@1 (on ImageNet-1K)	75.274
acc@5 (on ImageNet-1K)	92.566
min_size	height=1, width=1
categories	tench, goldfish, great white shark, ... (997 omitted)
num_params	5483032
recipe	link
GFLOPS	0.22
File size	21.1 MB

The inference transforms are available at `MobileNet_V3_Large_Weights.IMAGENET1K_V2.transforms` and perform the following preprocessing operations: Accepts `PIL.Image`, batched `(B, C, H, W)` and single `(C, H, W)` image `torch.Tensor` objects. The images are resized to `resize_size=[232]` using `interpolation=InterpolationMode.BILINEAR`, followed by a central crop of `crop_size=[224]`. Finally the values are first rescaled to `[0.0, 1.0]` and then normalized using `mean=[0.485, 0.456, 0.406]` and `std=[0.229, 0.224, 0.225]`.

于是我也相应地去改进了我的数据预处理

```
transforms = T.Compose([
        T.Resize([232]),
        # T.RandomRotation(15,center=(height/2,width/2)),
        # T.RandomResizedCrop((height, width),scale = (0.7,1.0)),
        T.CenterCrop([224]),
        # T.Random
        # T.RandomHorizontalFlip(0.1),  # 进行随机水平翻转
        # T.RandomVerticalFlip(0.1),  # 进行随机竖直翻转
        T.ToTensor(),  # 转化为张量
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),  # 归一化
    ])

    train_dataset = ImageFolder(data_path+"/train", transform=transforms)
    test_dataset = ImageFolder(data_path+"/val", transform=transforms)
```

但是效果仍然不见起色。

如果带着backbone一起训练呢，又出现了严重的过拟合，训练精度在98%的时候验证集精度才70%左右。

百思不得其解。

正好周五的时候有一个班大作业汇报，我去看了一下智云，发现他们用原始的模型是可以达到90%以上的精度的，然后我就怀疑Mindspore的预训练模型可能不是在Imagenet上训练的那个，而是另外预训练过的。这个时候想想这个图片的数据量规模才2k，再看看Imagenet，似乎也正常。此外，它在ImageNet-1K上的acc@1上的精度也不过75.274%,想想我的验证集精度70%其实也合理

那怎么办呢？只有开始学习Mindspore

Mindspore实践

其实main.ipynb上给的已经差不多的，我主要改了一下几个方面

repeat

```
create_dataset(config, training=True, buffer_size=1000, repeat=1)
```

create_dataset其实还有几个参数，和我之前pytorch的transforms一样是有数据增强的，所以repeat几次对于防止过拟合其实是大有裨益的。

eval & best model

然后自然而然要从众多的step中选出最好的一个，保存best model。但是其实原来并没有给出eval部分，这里是自己完成的。选择验证集中效果最好的，可以很好地避免过拟合。

```
    eval_dataset = create_dataset(config=config, training=False, repeat=1)
    eval_train_dataset = create_dataset(config=config, training=True, repeat=1)
    max_acc = 0
    for epoch in range(config.epochs):
        #train ....
        eval_model = Model(network, loss, metrics={'acc', 'loss'})
        acc = eval_model.eval(eval_dataset, dataset_sink_mode=False)
        if (acc['acc'] > max_acc):
            max_acc = acc['acc']
            save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2_all-best.ckpt"))
        print("Validation accuracy is", acc)
        acc = eval_model.eval(eval_train_dataset, dataset_sink_mode=False)
        print("Train accuracy is", acc)
```

后面也发现，压根训不出来，通常就是一开始精度不错，后来一直再降，这里部分还是非常有用的。

train_all

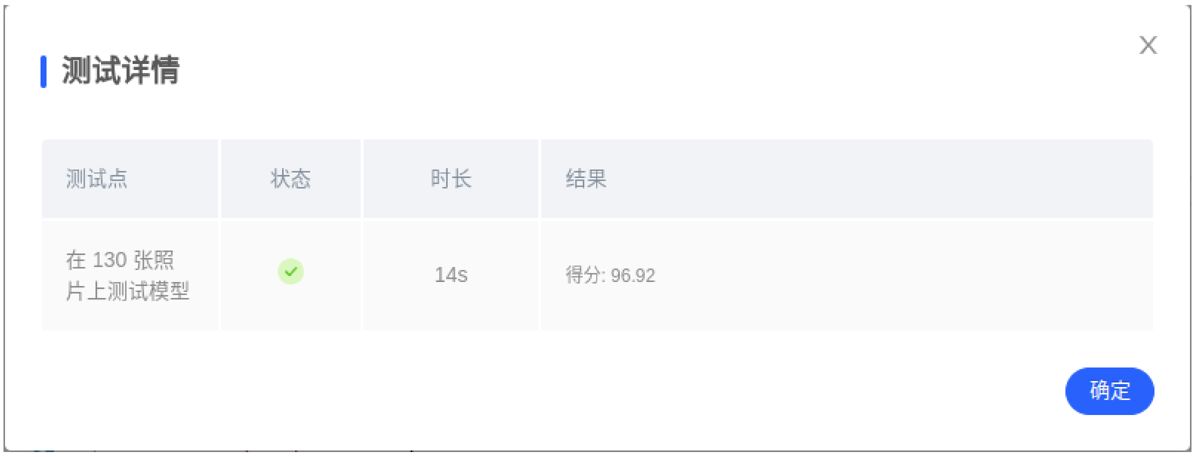
这里就是finetune整个网络，这里去掉了之前的导入预训练权重的部分

```
backbone = MobileNetV2Backbone()

    head = MobileNetV2Head(input_channel=backbone.out_channels,
num_classes=config.num_classes,
                           reduction=config.reduction)
    network = mobilenet_v2(backbone, head)
    if(config.pretrained_ckpt_all):
        load_checkpoint(config.pretrained_ckpt_all, net=network)
    else:
        print("Load best")
        load_checkpoint(os.path.join(config.save_ckpt_path, f"mobilenetv2-
best.ckpt"), net=network)
```

然后忽然意识到，训练中，还是使用了之前解压的feature。（感觉来搞笑的）

而在这之前，精度5个小时的GPU running time 和诸多调参，叠加best model带来的幸运者选择，我的模型精度已经来到了


测试详情

测试点	状态	时长	结果
在 130 张照片上测试模型	✓	14s	得分: 96.92

本以为下班收工开始写报告，写着写着发现问题不对劲。又得回去重新肝。

修改代码如下

```
net = nn.WithLossCell(network, loss)
train_step = nn.TrainOneStepCell(net, opt)
train_step.set_train()
data_iter = train_dataset.create_dict_iterator()
for epoch in range(config.epochs):
    for i, data in enumerate(data_iter):
        image = Tensor(data["image"])
        label = Tensor(data["label"])
        losses.append(train_step(image, label).asnumpy())
        epoch_seconds = (time.time() - epoch_start)
        epoch_loss = np.mean(np.array(losses))
```

然后又进行了多次训练，又用了5个小时的GPU,其中三次的训练效果如下

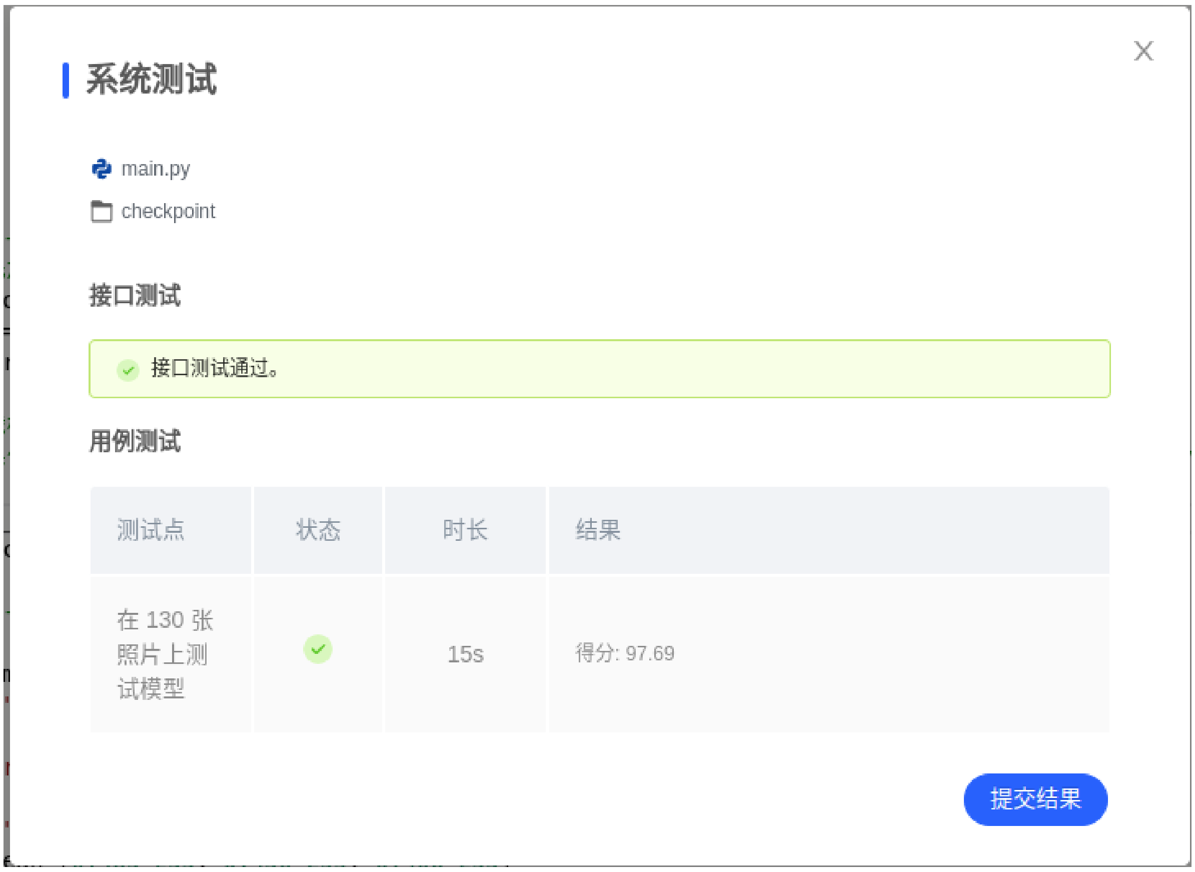
```
est1 Momentum lr=0.01，提交得分为90.0，训练时最好结果如下
2023-12-30 23:10:17.514200 Validation accuracy is {'loss': 0.33183986177811253,
'acc': 0.9461538461538461}
2023-12-30 23:10:27.312400 Train accuracy is {'loss': 0.3333841413259506, 'acc':
0.9418402777777778}
2023-12-30 23:11:16.433500 epoch: 40, time cost: 48.97537350654602, avg loss:
0.406130313873291

Test2 Momentum lr=0.02，提交得分为89.23，训练时最好结果如下
2023-12-30 23:09:10.901500 Validation accuracy is {'acc': 0.9461538461538461,
'loss': 0.31773222237825394}
2023-12-30 23:09:24.010400 Train accuracy is {'acc': 0.9401041666666666, 'loss':
0.31984901138477856}
2023-12-30 23:10:28.088900 epoch: 35, time cost: 63.929718017578125, avg loss:
0.40085709095000122

Test3 Adam lr=0.01，提交得分为89.23，训练时最好结果如下
2023-12-30 22:51:03.435400 Validation accuracy is {'loss': 0.3359083367081789,
'acc': 0.9461538461538461}
2023-12-30 22:51:10.229900 Train accuracy is {'loss': 0.3482605467240016, 'acc':
0.9275173611111112}
2023-12-30 22:51:54.730900 epoch: 35, time cost: 44.455583572387695, avg loss:
0.41424086689949036
```

根据这个结果，我猜测是出题人在所有的数据集上进行了训练，然后把backbone的权重部分拆出来给我们finetune，如果我们自己训练就是会出现严重的过拟合。怪不得我之前用Pytorch精度一直不行，破案了。不过我强烈建议以后这种事情应该写明.....浪费不少时间。

所以最终我们实验的测试给分是上述的97.68，但是和上面那个96.92交的是同一个权重......



系统测试

main.py
checkpoint

接口测试

接口测试通过。

用例测试

测试点	状态	时长	结果
在 130 张照片上测试模型	✓	15s	得分: 97.69

总结

不愧是大作业，花的时间也最多。不仅学习了Pytorch和Mindspore，同时也确实发现了很多问题。第一次去学会看model card，第一次去冻结backbone做finetune等等，总而言之还是受益匪浅。不过最后这个只能fientune的事实让我有点难绷。

源码

train

注:部分参数可能不是最优，优化器也需要在Adam和Momentum中选择，当时是挑了最优的一个

```
###############################################################################
# 重要: 请务必把任务(jobs)中需要保存的文件存放在 results 文件夹内
# Important : Please make sure your files are saved to the 'results' folder
# in your jobs
# 本代码来源于 Notebook cell 里面的模型，大家进行离线任务时尽量只训练模型，不要进行模型评估等操作
###############################################################################
import math
import numpy as np
import os
import cv2
import random
import shutil
import time
```

```
from matplotlib import pyplot as plt
from easydict import EasyDict
from PIL import Image

import mindspore as ms
from mindspore import context
from mindspore import nn
from mindspore import Tensor
from mindspore.train.model import Model
from mindspore.train.serialization import load_checkpoint, save_checkpoint,
export
from mindspore.train.callback import Callback, LossMonitor, ModelCheckpoint,
CheckpointConfig

from src_mindspore.dataset import create_dataset # 数据处理脚本
from src_mindspore.mobilenetv2 import MobileNetV2Backbone, MobileNetV2Head,
mobilenet_v2 # 模型定义脚本

os.environ['GLOG_v'] = '2' # Log Level = Error
has_gpu = (os.system('command -v nvidia-smi') == 0)
print('Excuting with', 'GPU' if has_gpu else 'CPU', '.')
context.set_context(mode=context.GRAPH_MODE, device_target='GPU' if has_gpu else
'CPU')

# 垃圾分类数据集标签，以及用于标签映射的字典。
index = {'00_00': 0, '00_01': 1, '00_02': 2, '00_03': 3, '00_04': 4, '00_05': 5,
'00_06': 6, '00_07': 7,
         '00_08': 8, '00_09': 9, '01_00': 10, '01_01': 11, '01_02': 12, '01_03':
13, '01_04': 14,
         '01_05': 15, '01_06': 16, '01_07': 17, '02_00': 18, '02_01': 19,
'02_02': 20, '02_03': 21,
         '03_00': 22, '03_01': 23, '03_02': 24, '03_03': 25}
inverted = {0: 'Plastic Bottle', 1: 'Hats', 2: 'Newspaper', 3: 'Cans', 4:
'Glassware', 5: 'Glass Bottle', 6: 'Cardboard', 7: 'Basketball',
            8: 'Paper', 9: 'Metalware', 10: 'Disposable Chopsticks', 11:
'Lighter', 12: 'Broom', 13: 'Old Mirror', 14: 'Toothbrush',
            15: 'Dirty Cloth', 16: 'Seashell', 17: 'Ceramic Bowl', 18: 'Paint
bucket', 19: 'Battery', 20: 'Fluorescent lamp', 21: 'Tablet capsules',
            22: 'Orange Peel', 23: 'Vegetable Leaf', 24: 'Eggshell', 25: 'Banana
Peel'}

# 训练超参
config = EasyDict({
    "repeat":10,
    "num_classes": 26, # 分类数，即输出层的维度
    "reduction": 'mean', # mean, max, Head部分池化采用的方式
    "image_height": 224,
    "image_width": 224,
    "batch_size": 64, # 鉴于CPU容器性能，太大可能会导致训练卡住
    "eval_batch_size": 10,
    "epochs": 40,
    "lr_max": 0.02,
    "decay_type": 'cosine',
    "momentum": 0.9,
    "weight_decay": 0.01,
    "dataset_path": "./datasets/5fbdf571c06d3433df85ac65-momodel/garbage_26x100",
    "features_path": "./results/garbage_26x100_features", # 临时目录，保存冻结层
Feature Map，可随时删除
```

```
    "class_index": index,
    "save_ckpt_epochs": 1,
    "save_ckpt_path": './results/ckpt_mobilenetv2_11',
    "pretrained_ckpt": './src_mindspore/mobilenetv2-200_1067_cpu_gpu.ckpt',
    "pretrained_ckpt_all": None,
    # "pretrained_ckpt": './checkpoint/mobilenetv2-247-792.ckpt',
    "export_path": './results/mobilenetv2.mindir'

})


def build_lr(total_steps, lr_init=0.005, lr_end=0.0001, lr_max=0.1,
warmup_steps=0, decay_type='cosine'):
    """
    Applies cosine decay to generate learning rate array.

    Args:
       total_steps(int): all steps in training.
       lr_init(float): init learning rate.
       lr_end(float): end learning rate
       lr_max(float): max learning rate.
       warmup_steps(int): all steps in warmup epochs.

    Returns:
       list, learning rate array.
    """
    lr_init, lr_end, lr_max = float(lr_init), float(lr_end), float(lr_max)
    decay_steps = total_steps - warmup_steps
    lr_all_steps = []
    inc_per_step = (lr_max - lr_init) / warmup_steps if warmup_steps else 0
    for i in range(total_steps):
        if i < warmup_steps:
            lr = lr_init + inc_per_step * (i + 1)
        else:
            if decay_type == 'cosine':
                cosine_decay = 0.5 * (1 + math.cos(math.pi * (i - warmup_steps) /
decay_steps))
                lr = (lr_max - lr_end) * cosine_decay + lr_end
            elif decay_type == 'square':
                frac = 1.0 - float(i - warmup_steps) / (total_steps -
warmup_steps)
                lr = (lr_max - lr_end) * (frac * frac) + lr_end
            else:
                lr = lr_max
        lr_all_steps.append(lr)

    return lr_all_steps


def extract_features(net, dataset_path, config):
    if not os.path.exists(config.features_path):
        os.makedirs(config.features_path)
    dataset = create_dataset(config=config,training=True, repeat=config.repeat)
    step_size = dataset.get_dataset_size()
    if step_size == 0:
        raise ValueError("The step_size of dataset is zero. Check if the images
count of train dataset is more \
            than batch_size in config.py")
```

```python
    data_iter = dataset.create_dict_iterator()
    for i, data in enumerate(data_iter):
        features_path = os.path.join(config.features_path, f"feature_{i}.npy")
        label_path = os.path.join(config.features_path, f"label_{i}.npy")
        if not os.path.exists(features_path) or not os.path.exists(label_path):
            image = data["image"]
            label = data["label"]
            features = net(image)
            np.save(features_path, features.asnumpy())
            np.save(label_path, label.asnumpy())
        print(f"Complete the batch {i+1}/{step_size}")
    return

backbone = MobileNetV2Backbone()
load_checkpoint(config.pretrained_ckpt, net=backbone)
extract_features(backbone, config.dataset_path, config)


class GlobalPooling(nn.Cell):
    """
    Global avg pooling definition.

    Args:
        reduction: mean or max, which means AvgPooling or MaxpPooling.

    Returns:
        Tensor, output tensor.

    Examples:
        >>> GlobalAvgPooling()
    """

    def __init__(self, reduction='mean'):
        super(GlobalPooling, self).__init__()
        if reduction == 'max':
            self.mean = ms.ops.ReduceMax(keep_dims=False)
        else:
            self.mean = ms.ops.ReduceMean(keep_dims=False)

    def construct(self, x):
        x = self.mean(x, (2, 3))
        return x


class MobileNetV2Head(nn.Cell):
    """
    MobileNetV2Head architecture.

    Args:
        input_channel (int): Number of channels of input.
        hw (int): Height and width of input, 7 for MobileNetV2Backbone with
image(224, 224).
        num_classes (int): Number of classes. Default is 1000.
        reduction: mean or max, which means AvgPooling or MaxpPooling.
        activation: Activation function for output logits.
    Returns:
        Tensor, output tensor.
    Examples:
        >>> MobileNetV2Head(num_classes=1000)
    """

    def __init__(self, input_channel=1280, hw=7, num_classes=1000,
reduction='mean', activation="None"):
        super(MobileNetV2Head, self).__init__()
        if reduction:
            self.flatten = GlobalPooling(reduction)
        else:
            self.flatten = nn.Flatten()
            input_channel = input_channel * hw * hw
        self.dense = nn.Dense(input_channel, num_classes, weight_init='ones',
has_bias=False)
        if activation == "Sigmoid":
            self.activation = nn.Sigmoid()
        elif activation == "Softmax":
            self.activation = nn.Softmax()
        else:
            self.need_activation = False

    def construct(self, x):
        x = self.flatten(x)
        x = self.dense(x)
        if self.need_activation:
            x = self.activation(x)
        return x

def train_head():
    train_dataset = create_dataset(config=config,training=True,
repeat=config.repeat)
    eval_train_dataset = create_dataset(config=config, training=True, repeat=1)
    eval_dataset = create_dataset(config=config,training=False, repeat=1)
    step_size = train_dataset.get_dataset_size()

    backbone = MobileNetV2Backbone()
    # Freeze parameters of backbone. You can comment these two lines.
    for param in backbone.get_parameters():
       param.requires_grad = False
    load_checkpoint(config.pretrained_ckpt, net=backbone)

    head = MobileNetV2Head(input_channel=backbone.out_channels,
num_classes=config.num_classes, reduction=config.reduction)
    network = mobilenet_v2(backbone, head)

    loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
    lrs = build_lr(config.epochs * step_size, lr_max=config.lr_max,
warmup_steps=5, decay_type=config.decay_type)
    opt = nn.Momentum(head.trainable_params(), lrs, config.momentum,
config.weight_decay)
    # opt = nn.Adam(params=head.trainable_params(), learning_rate=lrs,
weight_decay=config.weight_decay, use_nesterov=True)
    net = nn.WithLossCell(head, loss)
    train_step = nn.TrainOneStepCell(net, opt)
    train_step.set_train()

    # train
    history = list()
    features_path = config.features_path
    idx_list = list(range(step_size))
    max_acc=0
    for epoch in range(config.epochs):
        random.shuffle(idx_list)
        epoch_start = time.time()
        losses = []
        for j in idx_list:
            feature = Tensor(np.load(os.path.join(features_path,
f"feature_{j}.npy")))
            label = Tensor(np.load(os.path.join(features_path,
f"label_{j}.npy")))
            losses.append(train_step(feature, label).asnumpy())
        epoch_seconds = (time.time() - epoch_start)
        epoch_loss = np.mean(np.array(losses))

        history.append(epoch_loss)
        print("epoch: {}, time cost: {}, avg loss: {}".format(epoch + 1,
epoch_seconds, epoch_loss))

        eval_model = Model(network, loss, metrics={'acc', 'loss'})
        acc = eval_model.eval(eval_dataset, dataset_sink_mode=False)
        if(acc['acc']>max_acc):
            max_acc=acc['acc']
            save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2-best.ckpt"))
        print("Validation accuracy is", acc)
        acc = eval_model.eval(eval_train_dataset, dataset_sink_mode=False)
        print("Train accuracy is", acc)
        if (epoch + 1) % config.save_ckpt_epochs == 0:
            save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2-{epoch+1}.ckpt"))
    save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2_final.ckpt"))

    return history


def train_all():
    train_dataset = create_dataset(config=config, training=True,
repeat=config.repeat)
    eval_dataset = create_dataset(config=config, training=False, repeat=1)
    eval_train_dataset = create_dataset(config=config, training=True, repeat=1)
    step_size = train_dataset.get_dataset_size()

    backbone = MobileNetV2Backbone()

    head = MobileNetV2Head(input_channel=backbone.out_channels,
num_classes=config.num_classes,
                           reduction=config.reduction)
    network = mobilenet_v2(backbone, head)
    if(config.pretrained_ckpt_all):
        load_checkpoint(config.pretrained_ckpt_all, net=network)
    else:
        print("Load best")
        load_checkpoint(os.path.join(config.save_ckpt_path, f"mobilenetv2-
best.ckpt"), net=network)
```

```
    loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
    lrs = build_lr(config.epochs * step_size, lr_max=config.lr_max*0.5,
warmup_steps=5, decay_type=config.decay_type)
    # opt = nn.Adam(params=head.trainable_params(), learning_rate=lrs,
weight_decay=config.weight_decay, use_nesterov=True)
    opt = nn.Momentum(head.trainable_params(), lrs, config.momentum,
config.weight_decay)
    # opt = nn.Adam(params=head.trainable_params(), learning_rate=0.002,
weight_decay=0.001, use_nesterov=True)
    net = nn.WithLossCell(network, loss)
    train_step = nn.TrainOneStepCell(net, opt)
    train_step.set_train()

    # train
    history = list()
    features_path = config.features_path
    idx_list = list(range(step_size))
    max_acc = 0
    for epoch in range(config.epochs):
        random.shuffle(idx_list)
        epoch_start = time.time()
        losses = []
        data_iter = train_dataset.create_dict_iterator()
        for i, data in enumerate(data_iter):
            image = Tensor(data["image"])
            label = Tensor(data["label"])
            losses.append(train_step(image, label).asnumpy())
        epoch_seconds = (time.time() - epoch_start)
        epoch_loss = np.mean(np.array(losses))

        history.append(epoch_loss)
        print("epoch: {}, time cost: {}, avg loss: {}".format(epoch + 1,
epoch_seconds, epoch_loss))

        eval_model = Model(network, loss, metrics={'acc', 'loss'})
        acc = eval_model.eval(eval_dataset, dataset_sink_mode=False)
        if (acc['acc'] > max_acc):
            max_acc = acc['acc']
            save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2_all-best.ckpt"))
        print("Validation accuracy is", acc)
        acc = eval_model.eval(eval_train_dataset, dataset_sink_mode=False)
        print("Train accuracy is", acc)
        if (epoch + 1) % config.save_ckpt_epochs == 0:
            save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2_all-{epoch + 1}.ckpt"))

    save_checkpoint(network, os.path.join(config.save_ckpt_path,
f"mobilenetv2_all_final.ckpt"))
    return history

if os.path.exists(config.save_ckpt_path):
    shutil.rmtree(config.save_ckpt_path)
os.makedirs(config.save_ckpt_path)

history = train_head()
history2 = train_all()
```

```
CKPT = f'mobilenetv2-{config.epochs}.ckpt'
print("Chosen checkpoint is", CKPT)
print('training is ok!!!')
```

predict

```
## 生成 main.py 时请勾选此 cell
# 本示范以 NoteBook 训练模型通过平台测试为例：

# 1. 导入相关包
import os
import cv2
import numpy as np
import mindspore as ms
from mindspore import nn
from mindspore import Tensor
from easydict import EasyDict
from mindspore import context
from mindspore.train.serialization import load_checkpoint
from src_mindspore.mobilenetv2 import MobileNetV2Backbone, mobilenet_v2  # 模型定
义脚本

os.environ['GLOG_v'] = '2'  # Log Level = Error
has_gpu = (os.system('command -v nvidia-smi') == 0)
print('Excuting with', 'GPU' if has_gpu else 'CPU', '.')
context.set_context(mode=context.GRAPH_MODE, device_target='GPU' if has_gpu else
'CPU')

# 2.系统测试部分标签与该处一致，请不要改动
# 垃圾分类数据集标签，以及用于标签映射的字典。
index = {'00_00': 0, '00_01': 1, '00_02': 2, '00_03': 3, '00_04': 4, '00_05': 5,
'00_06': 6, '00_07': 7,
         '00_08': 8, '00_09': 9, '01_00': 10, '01_01': 11, '01_02': 12, '01_03':
13, '01_04': 14,
         '01_05': 15, '01_06': 16, '01_07': 17, '02_00': 18, '02_01': 19,
'02_02': 20, '02_03': 21,
         '03_00': 22, '03_01': 23, '03_02': 24, '03_03': 25}
inverted = {0: 'Plastic Bottle', 1: 'Hats', 2: 'Newspaper', 3: 'Cans', 4:
'Glassware', 5: 'Glass Bottle', 6: 'Cardboard', 7: 'Basketball',
            8: 'Paper', 9: 'Metalware', 10: 'Disposable Chopsticks', 11:
'Lighter', 12: 'Broom', 13: 'Old Mirror', 14: 'Toothbrush',
            15: 'Dirty Cloth', 16: 'Seashell', 17: 'Ceramic Bowl', 18: 'Paint
bucket', 19: 'Battery', 20: 'Fluorescent lamp', 21: 'Tablet capsules',
            22: 'Orange Peel', 23: 'Vegetable Leaf', 24: 'Eggshell', 25: 'Banana
Peel'}

## 生成 main.py 时请勾选此 cell

# 3. NoteBook 模型调整参数部分，你可以根据自己模型需求修改、增加、删除、完善部分超参数
# 训练超参
config = EasyDict({
    "num_classes": 26,
    "reduction": 'mean',
    "image_height": 224,
    "image_width": 224,
    "eval_batch_size": 10
```

```python
})

# 4. 自定义模型Head部分
class GlobalPooling(nn.Cell):
    def __init__(self, reduction='mean'):
        super(GlobalPooling, self).__init__()
        if reduction == 'max':
            self.mean = ms.ops.ReduceMax(keep_dims=False)
        else:
            self.mean = ms.ops.ReduceMean(keep_dims=False)

    def construct(self, x):
        x = self.mean(x, (2, 3))
        return x


class MobileNetV2Head(nn.Cell):
    def __init__(self, input_channel=1280, hw=7, num_classes=1000,
reduction='mean', activation="None"):
        super(MobileNetV2Head, self).__init__()
        if reduction:
            self.flatten = GlobalPooling(reduction)
        else:
            self.flatten = nn.Flatten()
            input_channel = input_channel * hw * hw
        self.dense = nn.Dense(input_channel, num_classes, weight_init='ones',
has_bias=False)
        if activation == "Sigmoid":
            self.activation = nn.Sigmoid()
        elif activation == "Softmax":
            self.activation = nn.Softmax()
        else:
            self.need_activation = False

    def construct(self, x):
        x = self.flatten(x)
        x = self.dense(x)
        if self.need_activation:
            x = self.activation(x)
        return x


# ------------------------- 5.请加载您最满意的模型 --------------------------
# 首先加载网络模型
backbone = MobileNetV2Backbone()
head = MobileNetV2Head(input_channel=backbone.out_channels,
num_classes=config.num_classes, reduction=config.reduction)
network = mobilenet_v2(backbone, head)

# 加载模型,加载请注意 model_path 是相对路径, 与当前文件同级。
# 如果你的模型是在 results 文件夹下的模型, 则 model_path =
'./results/ckpt_mobilenetv2/mobilenetv2-4.ckpt'

model_path = "checkpoint/mobilenetv2_all-best_7.ckpt"
load_checkpoint(model_path, net=network)

# ---------------------------------------------------------------------------
```

```
def image_process(image):
    """Precess one image per time.

    Args:
        image: shape (H, W, C)
    """
    mean=[0.485*255, 0.456*255, 0.406*255]
    std=[0.229*255, 0.224*255, 0.225*255]
    image = (np.array(image) - mean) / std
    image = image.transpose((2,0,1))
    img_tensor = Tensor(np.array([image], np.float32))
    return img_tensor

def predict(image):
    """
    加载模型和模型预测
    主要步骤:
        1.图片处理,此处尽量与训练模型数据处理一致
        2.用加载的模型预测图片的类别
    :param image: OpenCV 读取的图片对象，数据类型是 np.array，shape (H, W, C)
    :return: string, 模型识别图片的类别,
            包含 'Plastic Bottle','Hats','Newspaper','Cans'等共 26 个类别
    """
    # -------------------------- 实现图像处理部分的代码 --------------------------
    # 该处是与 NoteBook 训练数据预处理一致；
    # 如使用其它方式进行数据处理，请修改完善该处，否则影响成绩
    if isinstance(image, np.ndarray):
        # 转化为 PIL.JpegImagePlugin.JpegImageFile 类型
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    image = cv2.resize(image,(config.image_height, config.image_width))
    image = image_process(image)

    # -------------------------- 实现模型预测部分的代码 --------------------------
    logits = network(image)
    pred = np.argmax(logits.asnumpy(), axis=1)[0]

    return inverted[pred]
```

torch version of train

```
import warnings
# 忽视警告
warnings.filterwarnings('ignore')

import cv2
from PIL import Image
import numpy as np
import copy
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import ImageFolder
import torchvision.transforms as T
from torch.utils.data import DataLoader
from MobileNet import MobileNet
```

```
# from torch_py.Utils import plot_image
# from torch_py.MTCNN.detector import FaceDetector
# from torch_py.MobileNet import MobileNet
# from torch_py.FaceRec import Recognition

# 垃圾分类数据集标签，以及用于标签映射的字典。
index = {'00_00': 0, '00_01': 1, '00_02': 2, '00_03': 3, '00_04': 4, '00_05': 5,
'00_06': 6, '00_07': 7,
         '00_08': 8, '00_09': 9, '01_00': 10, '01_01': 11, '01_02': 12, '01_03':
13, '01_04': 14,
         '01_05': 15, '01_06': 16, '01_07': 17, '02_00': 18, '02_01': 19,
'02_02': 20, '02_03': 21,
         '03_00': 22, '03_01': 23, '03_02': 24, '03_03': 25}
inverted = {0: 'Plastic Bottle', 1: 'Hats', 2: 'Newspaper', 3: 'Cans', 4:
'Glassware', 5: 'Glass Bottle',
            6: 'Cardboard', 7: 'Basketball',
            8: 'Paper', 9: 'Metalware', 10: 'Disposable Chopsticks', 11:
'Lighter', 12: 'Broom', 13: 'Old Mirror',
            14: 'Toothbrush',
            15: 'Dirty Cloth', 16: 'Seashell', 17: 'Ceramic Bowl', 18: 'Paint
bucket', 19: 'Battery',
            20: 'Fluorescent lamp', 21: 'Tablet capsules',
            22: 'Orange Peel', 23: 'Vegetable Leaf', 24: 'Eggshell', 25: 'Banana
Peel'}
# def my_transforms():

def processing_data(data_path, height=224, width=224, batch_size=32,
                    test_split=0.1):
    """
    数据处理部分
    :param data_path: 数据路径
    :param height:高度
    :param width: 宽度
    :param batch_size: 每次读取图片的数量
    :param test_split: 测试集划分比例
    :return:
    """
    transforms = T.Compose([
        T.Resize([232]),
        # T.RandomRotation(15,center=(height/2,width/2)),
        # T.RandomResizedCrop((height, width),scale = (0.7,1.0)),
        T.CenterCrop([224]),
        # T.Random
        # T.RandomHorizontalFlip(0.1),  # 进行随机水平翻转
        # T.RandomVerticalFlip(0.1),  # 进行随机竖直翻转
        T.ToTensor(),  # 转化为张量
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),  # 归一化
    ])

    train_dataset = ImageFolder(data_path+"/train", transform=transforms)
    test_dataset = ImageFolder(data_path+"/val", transform=transforms)
    # print(dataset.class_to_idx)
    # # 划分数据集
    # train_size = int((1-test_split)*len(dataset))
    # test_size = len(dataset) - train_size
    # train_dataset, test_dataset = torch.utils.data.random_split(dataset,
[train_size, test_size])
```

```
    # 创建一个 DataLoader 对象
    train_data_loader = DataLoader(train_dataset,
batch_size=batch_size,shuffle=True)
    valid_data_loader = DataLoader(test_dataset,
batch_size=batch_size,shuffle=True)

    return train_data_loader, valid_data_loader
# def test():
#     img=Image.open('./datasets/5fbdf571c06d3433df85ac65-
momodel/garbage_26x100/train/00_00/00001.jpg')

data_path="./datasets/5fbdf571c06d3433df85ac65-momodel/garbage_26x100"
train_data_loader, valid_data_loader = processing_data(data_path=data_path,
height=160, width=160, batch_size=64)

device = torch.device("cuda:0") if torch.cuda.is_available() else
torch.device("cpu")
# device="cpu"

epochs = 200
model = MobileNet(classes=26).to(device)
# model.load_state_dict(
#                  torch.load('./results/temp1.pth', map_location=device))
optimizer = optim.Adam(model.parameters(), lr=0.01)  # 优化器
print('加载完成...')
# milestones=[10,50,100,250,500,750,1000]
# # 学习率下降的方式，acc三次不下降就下降学习率继续训练，衰减学习率
# scheduler =torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones,
gamma=0.5, last_epoch=-1, verbose=False)
milestones=[10,50,100,150]
# 学习率下降的方式，acc三次不下降就下降学习率继续训练，衰减学习率
scheduler =torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.5,
last_epoch=-1, verbose=False)
# 损失函数
criterion = nn.CrossEntropyLoss()

best_loss = 1e9
best_acc=0
best_acc1=0
best_model_weights = copy.deepcopy(model.state_dict())
loss_list = []  # 存储损失函数值
acc_list=[]
for epoch in range(epochs):
    model.train()
    total_loss= 0
    total_acc=0
    total_num=0
    total_acc1=0
    total_num1=0
    for batch_idx, (x, y) in tqdm(enumerate(train_data_loader, 1)):
        x = x.to(device)
        y = y.to(device)
        pred_y = model(x)

        # print(pred_y.shape)
        # print(y.shape)

        loss = criterion(pred_y, y)
        pred_y = torch.max(pred_y, dim=1)[1]
        # print(pred_y,y)
        total_acc1+= (pred_y==y).sum()
        total_num1+=x.shape[0]
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss+= loss.item()
        # if(batch_idx==1):
        #     print('step:' + str(batch_idx) + '/' + str(len(train_data_loader))
+ ' || Loss: %.4f' % (loss)+ ' || Train Acc: %.4f' % ((pred_y==y).sum()))
        # print('step:' + str(batch_idx) + '/' + str(len(train_data_loader)) + '
|| Total Loss: %.4f' % (total_loss/total_num1)+ ' || Train Acc: %.4f' %
(total_acc1/total_num1))
    scheduler.step()
    print(optimizer.state_dict()['param_groups'][0]['lr'])
    model.eval()
    for batch_idx, (x, y) in tqdm(enumerate(valid_data_loader, 1)):
        x = x.to(device)
        y = y.to(device)
        with torch.no_grad():
            pred_y = model(x)
        # get the predicted labels of pred_y
        pred_y = torch.max(pred_y, dim=1)[1]
        total_acc+= (pred_y==y).sum()
        total_num+=x.shape[0]

    acc_list.append(total_acc/total_num)
    loss_list.append(loss.to("cpu").detach().numpy() )
    if total_acc > best_acc or (total_acc == best_acc and total_acc1>best_acc1):
        best_model_weights = copy.deepcopy(model.state_dict())
        best_acc = total_acc
    print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total Loss: %.4f' %
(total_loss/total_num1)+ ' || Train Acc: %.4f' % (total_acc1/total_num1)+' ||
Valid Acc: %.4f' % (total_acc/total_num))

    if(epoch%10==0):
        torch.save(best_model_weights, './results/temp.pth')
        # print('Finish Training.')
torch.save(best_model_weights, './results/temp.pth')
print('Finish Training.')
```