基于神经网络的垃圾分类

张嘉浩[1], 代洋飞[2], 朱博医[3]

1. 浙江大学电气工程学院，自动化（电气学院）1903，3190103683

2. 浙江大学机械工程学院，机械工程1902，3190105373

3. 浙江大学机械工程学院，机械工程1906，3190100772

摘　要: 随着社会的飞速发展，生活中产生的消耗废品日益剧增，如何更好地分类与回收这些“垃圾”已经成为了急需解决的问题。为了我国能够更好更快的建立健全城市垃圾分类处理制度以及方便人们对于垃圾分类有更全面的认识，本文利用人工智能技术，设计了基于深度学习的垃圾分类模型，包括了全连接神经网络，卷积神经网络以及MobileNetV2神经网络，对包含2700多张图片的数据集进行分类训练。训练结果表明，在全连接神经网络，卷积神经网络和MobileNetV2神经网络中，MoblieNetV2网络的效果最好，准确率可以到达93.97%，实现了较好的分类效果。

关键词: 垃圾分类,卷积神经网络,MobileNetV2

Garbage Classification On Neural Network

Jiahao Zhang[1], Yangfei Dai[2], Boyi Zhu[2]

1. Automation, College of Electrical Engineering, Zhejiang University

2. Mechanical Engineering, College of Mechanical Engineering, Zhejiang University

Abstract: With the rapid development of society, the consumable waste generated in life is increasing rapidly. How to better classify and recycle these ”garbage” has become a problem that needs to be solved urgently. In order for my country to better and faster establish and improve the urban waste classification and treatment system and facilitate people to have a more comprehensive understanding of waste classification, this article uses artificial intelligence technology to design a waste classification model based on deep learning, including a fully connected neural network. Convolutional neural network and MobileNetV2 neural network classify and train a data set containing more than 2700 pictures. The training results show that among the fully connected neural network, convolutional neural network and MobileNetV2 neural network, the MoblieNetV2 network has the best effect, with an accuracy rate of 93.97%, which achieves a better classification effect.

Key Words: Garbage Sorting, Convolutional Neural Network, MobileNetV2

1　引言

1.1　选题背景

垃圾分类，一般是指按一定规定或标准将垃圾分类储存、分类投放和分类搬运，从而转变成公共资源的一系列活动的总称。自今年7月1日起，上海市将正式实施《上海市生活垃圾管理条例》。垃圾分类，看似是微不足道的“小事”，实则关系到13亿多人生活环境的改善，理应大力提倡。

1.2　研究意义

分类的目的是提高垃圾的资源价值和经济价值，力争物尽其用。进行垃圾分类收集可以减少垃圾处理量和处理设备，降低处理成本，减少土地资源的消耗，具有社会、经济、生态等几方面的效益。生活垃圾由于种类繁多，具体分类缺乏统一标准，大多人在实际操作时会“选择困难”，基于深度学习技术建立准确的分类模型，利用技术手段改善人居环境。

1.3　相关文献综述

早期, 学者们只能借助经典的图像分类算法完成垃圾图像分类任务, 这要通过手动提取的图像特征并结合相应的分类器完成. 吴健等利用颜色和纹理特征, 初步完成了废物垃圾识别. 由于不同数据集的图像背景、尺寸、质量不尽相同, 传统算法需要根据相应数据人工提取不同的特征, 算法的鲁棒性较差, 并且处理方式复杂, 所需时间较长, 无法达到实时的效果. 随着卷积神经网络(Convolution Neural Network, CNN) 的飞速发展, 深度学习广泛应用于图像识别领域. 作为数据驱动的算法, CNN 具有强大的特征拟合能力, 可以有效、 自动地提取图像特征, 并具有较快的运行速度. 2012 年, AlexNet取得了ImageNet图像分类竞赛的冠军, 标志着深度学习的崛起. 随后几年, GoogleNet、VGGNet、ResNet等算法提升了图像分类的精度, 并成功应用于人脸识别、 车辆检测等多个领域. 垃圾图像分类, 在深度学习算法的帮助下同样取得了较大的突破. 斯坦福大学的Yang等建立了TrashNet Dataset公开数据集, 包含6个类别, 共计2527张图片. Ozkaya等通过对比不同CNN网络的分类能力, 搭建神经网络(本文称之为TrashNet)并进行参数微调, 在数据集TrashNet Dataset上取得了97.86%的准确率, 是目前这一数据集上最佳分类网络. 在非公开数据集方面, Mittal等自制了2561 张的垃圾图片数据集GINI, 使用GarbNet 模型, 得到了87.69%的准确率. 国内方面, 郑海龙等用SVM 方法进行了建筑垃圾分类方面的研究. 向伟等使用分类网络CaffeNet, 调整卷积核尺寸和网络深度, 使其适用于水面垃圾分类, 在其

自制的1500张图片数据集上取得了95.75% 的识别率. 2019年, 华为举办垃圾图像分类竞赛, 构建了样本容量为一万余张的数据集, 进一步促进了该领域的发展。

2 算法描述与求解结果

2.1 算法描述与求解结果

小组需要以MobileNetV2+ 垃圾分类数据集为例，使用深度学习框架（如Pytorch/Tensorflow）在CPU/GPU平台上进行训练，实现对26种垃圾进行分类。或者自己搭建神经网络，实现垃圾分类。

2.2 训练样本

训练样本由mo平台提供，图片分为干垃圾，可回收物，湿垃圾和有害垃圾四个大类共26 种物品共2375张图片组成，在训练过程中使用以下函数将训练样本划分为训练集和测试集两大类，供后续的训练和验证所用：

```
train_db, val_db =
    torch.utils.data.random_split(dataset,[2250,125])
```

2.3 搭建过程

对于垃圾分类问题，我们小组主要进行了三轮模型的调试，具体过程如下：

2.3.1 第一轮尝试：DNN

我们首先搭建了简单的一个三层全连接神经网络（DNN），我们首先将输入的图片展开为一个一维向量：

```
# flaten inputs
inputs = Input(shape=input_shape)
dnn = Flatten()(inputs)
```

然后将一维向量输入到搭建完的神经网络中，每一层网络由BN层，激活层组成，第一层网络的结构如下：

```
# fully connected layer
dnn = Dense(6)(dnn)
dnn = BatchNormalization(axis=-1)(dnn)
dnn = Activation('sigmoid')(dnn)
dnn = Dropout(0.25)(dnn)
```

在三层网络中，我们分别采用了$sigmoid$, $relu$和$softmax$ 激活函数。通过对该神经网络进行长达一个小时的训练，但是由于图像信息很大，而全连接层并不适合，测试集准确度很低，最高仅达到0.3左右。

2.3.2 第二轮尝试：CNN

卷积神经网络通常被用于处理多阵列形式的数据，例如有三个二维阵列组成的RGB图像，每个二位整列包含的是三个彩色通道的像素强度。该类网络的背后有四个关键思想，它们利用了自然信号的特性，分别是：本地连接、共享权重、池化和多层的使用。

由于CNN的特征检测层通过训练数据进行学习，所以在使用CNN时，避免了显式的特征抽取，而隐式地从训练数据中进行学习；再者由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。

网络结构 该神经网络由数据输入层(Input layer)，卷积计算层(CONV layer)，ReLU激励层(ReLU layer)，池化层(Pooling layer)，全连接层(FC layer)组成，其结构入下图所示：

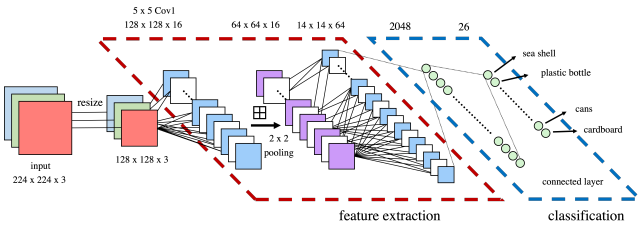

图 1: 卷积神经网络结构

数据输入层对图像进行去均值、归一化等预处理；特征提取部分主要由4层卷积与池化层组成，卷积、池化的操作将一张图片的维度进行了压缩，提取了图像的特征并能有效的防止过拟合；全连接层通常在卷积神经网络尾部负责网络与输出的联接。从图示上我们不难看出卷积网络的精髓就是适合处理结构化数据，而该数据在跨区域上依然有关联。

我们自己搭建了一个五层的卷积神经网络（CNN），每一层神经网络由卷积层，BN（归一化）层，激活层与最大池化层组成，以第一层结构为例：

```
# conv1: Conv2d -> BN -> ReLU -> MaxPool
self.conv1 = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=16,
        kernel_size=3, stride=1, padding=1),
    nn.Conv2d(in_channels=16, out_channels=16,
        kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1),
    )
```

图片变换 我们对图片进行了一系列变换，如旋转、翻转、灰度化以增强稳定度：

```
transform = transforms.Compose([
    #size transformation
    transforms.Resize((128,128)),
    #rotation
    transforms.RandomRotation((30,30)),
    #vertical flip
    transforms.RandomVerticalFlip(0.1),
    #Grayscale
    transforms.RandomGrayscale(0.1),
```

```
    #transform tensor
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)
```

调整超参 在训练过程中主要调整以下参数：

```
# #hyper parameter
batch_size = 32
num_epochs = 50
lr = 0.0001
num_classes = 25
image_size = 128 #### 64
```

训练过程 对于该神经网络，设置epoch = 50 进行训练，得到训练的损失曲线如下图所示：虽然训练集的

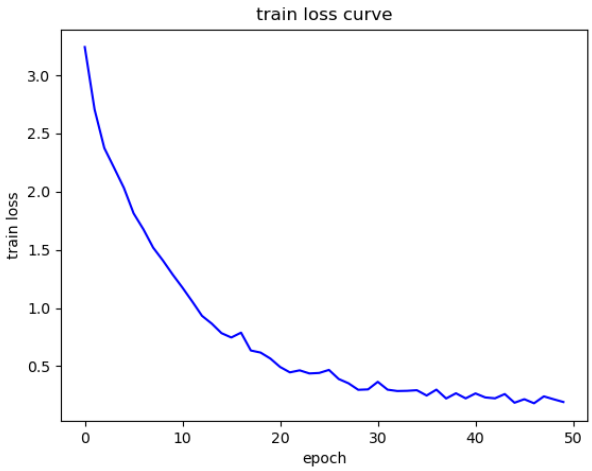

图 2: 卷积神经网络训练结果

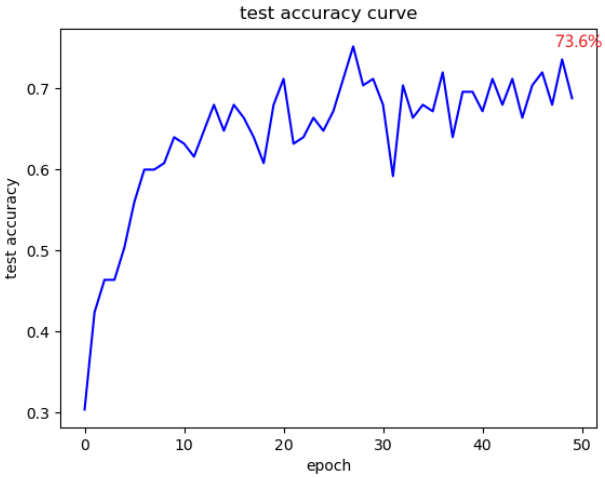

图 3: 卷积神经网络在测试集的结果

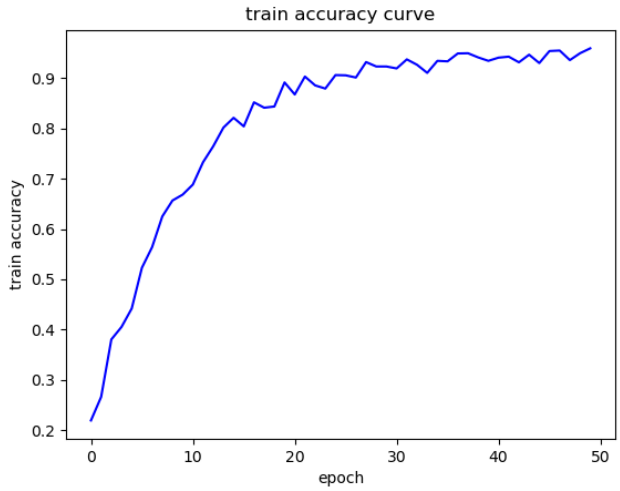

图 4: 卷积神经网络在验证集的结果

损失值随着epoch的增加而接近于0，但是神经网络在测试集的表现程度并不好，从图2可以看到，随着训练次数的增加，正确率在0.65左右波动，最后正确率在70%左右，说明模型进入了过拟合阶段。它虽然在训练集的表现中越来越好，但是由于训练集样本数量过少，使得训练出的网络不能很好的应用于所有的情况。

2.3.3 第三轮尝试：MobileNetV2

此次尝试采用了mo平台提供的利用MindSpore 搭建MobileNetV2网络模型的基础上进行参数修改。

由于在深度学习计算中，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。那么我们的工作也是在原有的网络结构上进行微调得到的。

网络简介

MobileNet 在现实场景下，诸如移动设备、嵌入式设备、自动驾驶等等，计算能力会受到限制，MobileNet由此提出。相较于传统网络，它有两个特点

- 使用深度可分离卷积(depthwise separable convolutions)替代传统卷积。
- 引入了两个收缩超参数(shrinking hyperparameters)：宽度乘子(width multiplier)和分辨率乘子(resolution multiplier)。

MobileNetV2 MobileNetV2 主要引入了两个改动：Linear Bottleneck 和Inverted Residual Blocks。

- Inverted Residual Blocks: MobileNetV2 结构基于inverted residual。其本质是一个残差网络设计，传统Residual block 是block 的两端channel 通道数

多，中间少，而MobileNetV2 设计的inverted residual是block 的两端channel 通道数少，block 内channel多，类似于沙漏和梭子形态的区别。

- Linear Bottlenecks: 感兴趣区域在ReLU 之后保持非零，近似认为是线性变换。ReLU 能够保持输入信息的完整性，但仅限于输入特征位于输入空间的低维子空间中。对于低纬度空间处理，论文中把ReLU 近似为线性转换。

MobileNetV2的网络结构如图5所示：

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k		

图 5: 卷积神经网络结构

其中，网络模块Bottleneck的结构如图6所示，这是一种残差结构的网络，它将通过卷积之后的结果与输入直接相加，从而达到了将高维特征映射到低维空间的作用。

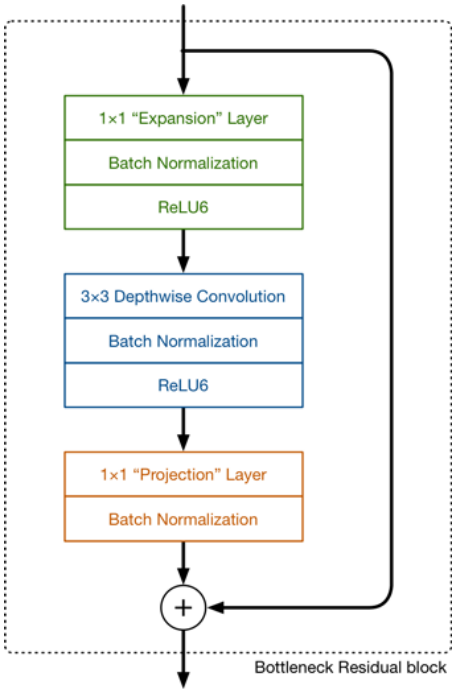

图 6: MobileNetV2网络结构

在Bottleneck网络的开头有一个Expansion layer，它以将网络的结构进行扩大，这样使得最后卷积出来的结果能与输入相加，达到将低维空间映射到高维空间的效果，它的结构如图7所示：

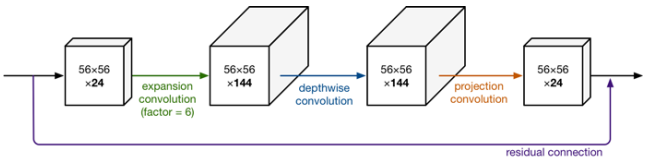

图 7: MobileNetV2网络结构

数据准备 将脚本、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
```

图 8: 文件组织形式

参数配置 配置后续训练、验证、推理用到的参数。可以调整以下超参以提高模型训练后的验证精度，如下所示：

- epochs：在训练集上训练的代数；
- lr_max：学习率，或者动态学习率的最大值；
- decay_type：学习率下降策略；
- momentum：Momentum 优化器的动量参数，通常为0.9；
- weight_decay：正则化项的系数。

```
# Train super parameters
config = EasyDict({
    # the dimension of the output layer
    "num_classes": 26,
    # mean, max, Head part of the pooling method
    "reduction":'mean',
    "image_height": 224,
    "image_width": 224,
    # In view of the performance of the CPU
    "batch_size": 10,
    "eval_batch_size": 10,
    "epochs": 50,
    "lr_max": 0.015,
    "decay_type":'constant',
    "momentum": 0.91,
    "weight_decay": 1e-3,
    # ... Ingore other parameters
})
```

训练策略 一般情况下，模型训练时采用静态学习率，如0.01。随着训练步数的增加，模型逐渐趋于收敛，对权重参数的更新幅度应该逐渐降低，以减小模型训练后期的抖动。所以，模型训练时可以采用动态下降的学习率，常见的学习率下降策略有：

- polynomial decay/square decay;
- cosine decay;
- exponential decay;
- stage decay.

这里实现cosine decay 和square decay 下降策略。

训练策略 在模型训练过程中， 通过添加检查点（Checkpoint）用于保存模型的参数，以便进行推理及中断后再训练使用。使用场景如下：

训练后推理场景

- 模型训练完毕后保存模型的参数，用于推理或预测操作。
- 训练过程中，通过实时验证精度，把精度最高的模型参数保存下来，用于预测操作。

再训练场景

- 进行长时间训练任务时， 保存训练过程中的Checkpoint文件，防止任务异常退出后从初始状态开始训练。
- Fine-tuning（微调）场景，即训练一个模型并保存参数，基于该模型，面向第二个类似任务进行模型训练。

这里加载ImageNet数据上预训练的MobileNetv2进行Fine-tuning，并在训练过程中保存Checkpoint。训练有两种方式：

- 冻结网络的Backbone，只训练修改的FC层（Head）。其中，Backbone再全量数据集上做一遍推理，得到Feature Map，将Feature Map作为训练Head的数据集，可以极大节省训练时间。
- 先冻结网络的Backbone， 只训练网络Head； 再对Backbone+Head做整网做微调。

参数调整 我们对网络的调整主要是以下几个参数：

"reduction"，部分池化方式 池化层的目的就是对大量的特征信息进行过滤，去除其中的冗余信息并筛选出最具代表性的特征信息，因此可以把池化层当作是一个滤波器。池化层的作用包括减少网络中参数的数量、压缩数据以及减少网络的过拟合。池化层里面主要包含了两个参数，分别是步长和池化核大小。池化核以滑动窗口的方式对输入的特征图进行处理，经过不同的池化函数的计算，得到相应的关键特征，其中每个池化层中的池化函数是固定的，一般不需要再引入其他参数。池化函数是池化层的核心，池化函数的不同也就对应着不同的池化方法。一个较好的池化方法通常能够在删除大量的无关信息的同时并且尽可能多的保留关键信息，进而在很大程度上提升整个卷积神经网络的性能。

池化方法中最常见的方法是最大池化和平均池化。最大池化只保留池化框中的最大值，因而最大池化可以有效提取出特征图中最具代表性的信息。平均池化则计算出池化框中所有值的均值，因而可以平均获取特征图中的所有信息，进而不致丢失过多关键信息。这两种方法由于计算简单且效果较好因而被广泛利用在了各种结构的卷积神经网络中，但这两种方法的缺点也是不可忽视的。最大池化由于完全删除了最大值以外的其他值，这往往导致保留了特征图中的前景信息而忽略了所有的背景信息；而平均池化由于取得了所有值之和的均值，虽然对特征图中的背景信息有所保留，但是无法将特征图中的前景信息和背景信息有效地区分开。

基于对两种池化方法（max 和mean）的考虑，在该网络中我们选择了最大池化方法，那么在采取了最大池化之后，整个网络的准确率有了很大的提高，从开始的60% 一下子上升到80%。

"batch_size"，批尺寸 Batch 决定了梯度下降的方向。如果数据集比较小，完全可以采用全数据集(Full Batch Learning)的形式，这样做至少有2 个好处：其一，由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。其二，由于不同权重的梯度值差别巨大，因此选取一个全局的学习率很困难。Full Batch Learning 可以使用Rprop （弹性反向传播）只基于梯度符号并且针对性单独更新各权值。对于更大的数据集，以上2 个好处又变成了2 个坏处：其一，随着数据集的海量增长和内存限制，一次性载入所有的数据进来变得越来越不可行。其二，以Rprop 的方式迭代，会由于各个Batch 之间的采样差异性，各次梯度修正值相互抵消，无法修正。

在合理范围内，增大Batch_Size 可以提高内存利用率。跑完一次epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快。在一定范围内，一般来说Batch_Size 越大，其确定的下降方向越准，引起训练震荡越小。但Batch_Size 过大，可能使得内存容量不足而训练失败，同时跑完一次epoch（全数据集）所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加了，从而对参数的修正也就显得更加缓慢。

我们根据尝试，选择了最佳的batch_size = 10, 使得训练性能达到最佳。

"lr_rate"，最大学习率 学习率(Learning rate)作为监督学习以及深度学习中重要的超参，其决定着目标函数能否收敛到局部最小值以及何时收敛到最小值。合适的学习率能够使目标函数在合适的时间内收敛到局部最小值。当学习率设置的过小时，收敛过程将变得十分缓慢。而当学习率设置的过大时，梯度可能会在最小值附近来回震荡，甚至可能无法收敛。

同时，学习率和batchsize的紧密相连，通常当我们增加batchsize为原来的N倍时，要保证经过同样的样本后更新的权重相等，按照线性缩放规则，学习率应该增加为原来的N倍。但是如果要保证权重的方差不变，则学习率应该增加为原来的sqrt(N)倍。

那么经过大量尝试，我们将学习率最终确定为了lr_max = 0.015 以达到相对来说的最优值。

训练结果 将参数调整为合适值进行训练，训练35轮，最终得到准确率为93.97%，损失曲线如下：

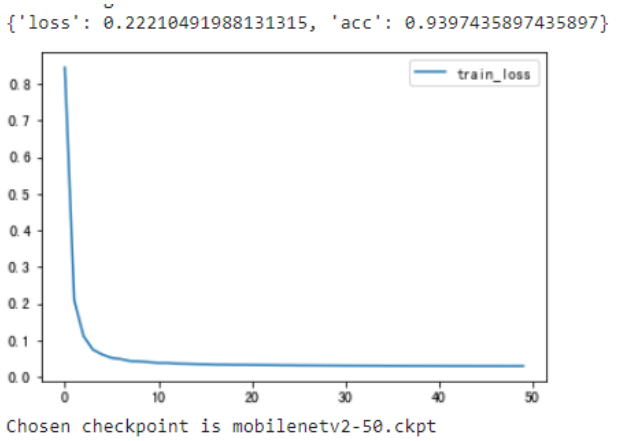

图 9: MobileNetV2训练结果

预测结果 在所给的验证集中，大多数种类垃圾可以全部识别正确，个别垃圾种类会识别错误一张，预测的结果如下图所示：

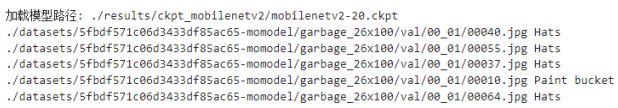

图 10: 预测帽子结果

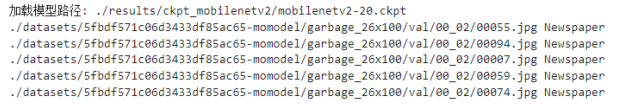

图 11: 预测报纸结果

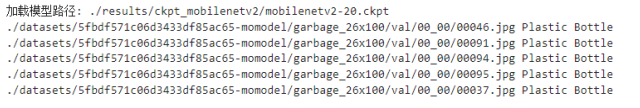

图 12: 预测塑料瓶结果

那么最后在MO平台的非公开数据集中的测试结果如下，可以看到，虽然在自己的训练集和验证集中，我们的神经网络已经能达到很好的效果（93.85分）。

2.3.4 第四轮尝试：Swim-Transformer

我们后续还引入了Swim-Transformer网络，Swim-Transformer是一种Transformer网络。传统的Transformer都是基于全局来计算注意力的，因此计算复杂度十分高。而Swin Transformer则将注意力的计算限制在每个窗口内，进而减少了计算量。swim-transformer的网络结构如下所示：

图 13: 在MO平台非公开数据集的测试结果

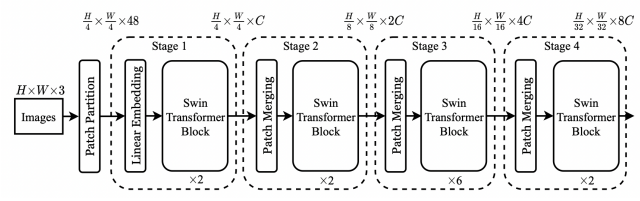

图 14: Swim-Transformer网络结构

那么在经过了80次的训练，网络在验证集与训练集中都达到了90%的准确率，我们在网络中找了一张图片，经网络进行验证，得到识别结果为Hats，概率为0.971。但是由于Pytorch版本的差异（Swim-T网络要求Pytorch>1.70，而MO平台上的Pytorch=1.40）在将模型进行迁移的时候遇到了比较大的困难。

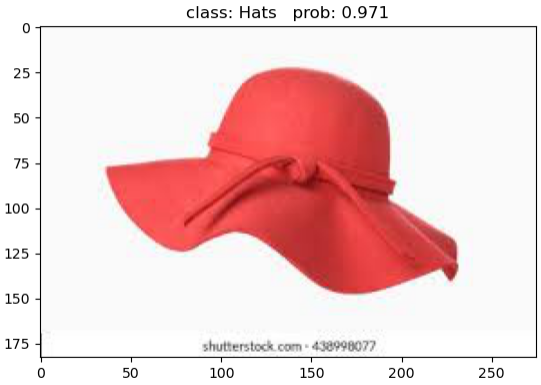

图 15: Hat 识别结果

3 比较分析&结论

在此次实验中我们实验的网络主要是CNN和MobileNet。

首先，我们之所以选择CNN是因为卷积网络的精髓就是适合处理结构化数据，而该数据在跨区域上依然有关联，常被用于图像分析。它具有参数共享机制的优点，在卷积层中每个神经元连接数据窗的权重是固定的，每个神经元只关注一个特性。神经元就是图

像处理中的滤波器，比如边缘检测专用的Sobel 滤波器，即卷积层的每个滤波器都会有自己所关注一个图像特征，比如垂直边缘，水平边缘，颜色，纹理等等，这些所有神经元加起来就好比就是整张图像的特征提取器集合。另外，CNN也无需手动选取特征，训练好权重，即得输入特征（自动的），分类效果好一般比较好。但是，CNN的缺点就是需要调参，需要大样本量，训练最好要GPU，而在本次实验中，我们认为我们自己搭建的CNN的准确率不太理想的原因是因为网络模型还不是最优，并且样本数量不足。

而在MobileNet网络中，虽然最后准确率可以稳定在90%以上，并且预测效果也算比较理想，在所给的验证集中，大多数种类垃圾可以全部识别正确，个别垃圾种类会识别错误一张。在引入了残差网络结构之后，通过将卷积之前的图像与卷积之后的图像相叠加加，保留了图像的特征，使得网络的识别准确率大大增加了。

从全连接网络，卷积神经网络到MobileNetV2再到Swim-Transformer，我们模型的准确率不断提高，这种提高与网络的结构密切相关。从全连接网络到卷积神经网络，我们看到了卷积层对特征提取的重要作用，从CNN到MobileNetV2，我们见识到了残差网络对于保留特征的作用，也意识到了卷积神经网络的局限性。在最后尝试的Swim-Transformer中，我们也体会到了注意力机制的威力。

4 研究感想&组员分工

4.1 研究感想

张嘉浩 在搭建神经网络的过程中，我从结构上对神经网络有了新的理解。搭建一个网络与学习一个网络有很大的不同，需要注意的地方也更多，比如网络的大小和深度等等。那么在调整参数的过程中，我也意识到调参并不是盲目的，需要结合各个参数的意义和网络的输出结果进行针对性的调整。在搜集资料的过程中，我们小组也通过前人的文章对人工智能与机器学习这个领域有了更加深入的了解和全新的认识，收获颇丰。

代洋飞 垃圾分类神经网络与MO平台的四个小实验有很多不同，最重要的一点就是网络的结构更多，层数更深，这也使得模型的调参更加困难。通过教程提到的Fine-tune微调和设置checkpoint，我们的调参顺利了很多。

朱博医 通过本次大作业，我切身体验了搭建一个神经网络的全过程。从最初的梯度下降算法，到采用pytorch框架建立神经网络结构；从最基础的全连接层网络，到比较复杂的卷积神经网络，再到MobileNetV2，我切身体验到了对不同的分类任务选择不同的神经网络算法的重要性，也目睹了准确率的逐步上升。相信通过此次学习，将来我将会更加熟练得使用神经网络。

4.2 成员分工

- 张嘉浩：搭建神经网络，模型调参，报告撰写
- 代洋飞：搭建神经网络，模型调参，报告撰写
- 朱博医：数据集读取，模型调参，PPT制作

参考文献

[1] 吕思敏. 以史为鉴, 开启垃圾分类新时代, 城乡建设,2020(3): 30-32.

[2] Lowe DG, Distinctive image features from scale-invariant keypoints, in *International Journal of Computer Vision*, 2004, 60(2): 91-110.

[3] Harri C, Stephens M, A combined corner and edge detector in *Proceedings of the 4th Alvey Vision Conference*, Manchester, UK. 1988. 207–217.

[4] Zhang XK, Wang Y, Gou MR, et al, Efficient temporal sequence comparison and classification using gram matrix embeddings on a riemannian manifold in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA. 2016. 4498–4507.

[5] Ozkaya U, Seyfi L. Fine-tuning models comparisons on garbage classification for recyclability, *arXiv: 1908.04393, 2019.*

[6] Mittal G, Yagnik KB, Garg M, et al. SpotGarbage: Smartphone app to detect garbage using deep learning, *Proceedings of 2016 ACM International Joint Conference*, Heidelberg, Germany. 2016. 940–945.

[7] Kingma DP, Ba J. Adam. A method for stochastic optimization, *arXiv: 1412.6980, 2017*

[8] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, Lake Tahoe, NV, USA. 2012. 1106–1114.

[9] Harri C, Stephens M, A combined corner and edge detector, *Proceedings of the 4th Alvey Vision Conference*, Manchester, UK. 1988. 207–217.

[10] Vapnik V. Statistical Learning Theory, *New York: Wiley*,1998. 401–492.

[11] 吴健, 陈豪, 方武. 基于计算机视觉的废物垃圾分析与识别研究, 信息技术与信息化, 2016(10): 81-83

[12] 向伟, 史晋芳, 刘桂华, 等.改进CaffeNet模型在水面垃圾识别中的应用, 传感器与微系统, 2019, 38(8): 150-152, 156.

[13] Kingma DP, Ba J. Adam, A method for stochastic optimization, *arXiv: 1412.6980, 2017.*

附录

CNN 代码

```
# Temp 2
### 浙江大学人工智能课程——实现垃圾分类CNN
### 代洋飞张嘉浩朱博医
### 2021/12/15

import torch
import torch.nn as nn
import torch.utils.data as Data
import torchvision
import time
import os
from torchvision import models
import matplotlib.pyplot as plt
from torchviz import make_dot


class MyCNN(nn.Module):
    """网络模型

    """
    def __init__(self, image_size, num_classes):
        super(MyCNN, self).__init__()
        # conv1: Conv2d -> BN -> ReLU -> MaxPool
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding='same'),
            nn.Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=1),
        )
        # conv2: Conv2d -> BN -> ReLU -> MaxPool  nn.AdaptiveAvgPool2d
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding='same'),
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2),
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding='same'),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2),
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding='same'),
            nn.BatchNorm2d(64),
```

```
48              nn.ReLU(),
49              nn.AvgPool2d(kernel_size=2, stride=2),
50          )
51          self.conv5 = nn.Sequential(
52              nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding='same'),
53              nn.BatchNorm2d(64),
54              nn.ReLU(),
55              nn.AvgPool2d(kernel_size=2, stride=1),
56          )
57          # fully connected layer
58          self.dp1 = nn.Dropout(0.50)
59          self.fc1 = nn.Linear(12544, 2048)
60          self.dp2 = nn.Dropout(0.50)
61          self.fc2 = nn.Linear(2048, 256)
62          self.dp3 = nn.Dropout(0.20)
63          self.fc3 = nn.Linear(256, num_classes)
64
65      def forward(self, x):
66          """
67          input: N * 3 * image_size * image_size
68          output: N * num_classes
69          """
70          x = self.conv1(x)
71          x = self.conv2(x)
72          x = self.conv3(x)
73          x = self.conv4(x)
74          x = self.conv5(x)
75          # view(x.size(0), -1): change tensor size from (N ,H , W) to (N, H*W)
76          x = x.view(x.size(0), -1)
77          x = self.dp1(x)
78          x = self.fc1(x)
79          x = self.dp2(x)
80          x = self.fc2(x)
81          x = self.dp3(x)
82          output = self.fc3(x)
83          return output
84
85
86  def getRsn():
87      model = models.resnet18(pretrained=True)
88      num_fc_in = model.fc.in_features
89      model.fc = nn.Linear(num_fc_in, 6)
90      return model
91
92  def getMbnet():
93      model = models.mobilenet_v2(pretrained=True)
94      model.classifier = nn.Sequential(
95          nn.Linear(in_features=1280,out_features=64),
96          nn.Dropout(p=0.5,inplace=False),
97          nn.Linear(in_features=64,out_features=6,bias=True),
98      )
```

```
99      return model
100
101 def train(model, train_loader, loss_func, optimizer, device):
102     """训练模型
103
104     train model using loss_fn and optimizer in an epoch.
105     model: CNN networks
106     train_loader: a Dataloader object with training data
107     loss_func: loss function
108     device: train on cpu or gpu device
109     """
110     total_loss = 0
111     # train the model using minibatch
112     for i, (images, targets) in enumerate(train_loader):
113         images = images.to(device)
114         targets = targets.to(device)
115
116         # forward
117         outputs = model(images)
118         _,preds = torch.max(outputs.data,1)
119         loss = loss_func(outputs, targets)
120
121         # backward and optimize
122         optimizer.zero_grad()
123         loss.backward()
124         optimizer.step()
125
126         total_loss += loss.item()
127         if (i + 1) % 100 == 0:
128             print ("Step [{}/{}] Train Loss: {:.4f} Train acc".format(i+1, len(train_loader), loss.item()))
129     save_model(model, save_path="results/cnn.pth")
130     return total_loss / len(train_loader)
131
132
133 def evaluate(model, val_loader, device, name):
134     """评估模型
135
136     model: CNN networks
137     val_loader: a Dataloader object with validation data
138     device: evaluate on cpu or gpu device
139     return classification accuracy of the model on val dataset
140     """
141     # evaluate the model
142     model.eval()
143     # context-manager that disabled gradient computation
144     with torch.no_grad():
145         correct = 0
146         total = 0
147
148         for i, (images, targets) in enumerate(val_loader):
149             # device: cpu or gpu
```

```
150             images = images.to(device)
151             targets = targets.to(device)
152             outputs = model(images)
153 
154             # return the maximum value of each row of the input tensor in the
155             # given dimension dim, the second return vale is the index location
156             # of each maxium value found(argmax)
157             _, predicted = torch.max(outputs.data, dim=1)
158             correct += (predicted == targets).sum().item()
159 
160             total += targets.size(0)
161 
162         accuracy = correct / total
163         print('Accuracy on '+name+' Set: {:.4f} %'.format(100 * accuracy))
164         return accuracy
165 
166 
167 def save_model(model, save_path="results/cnn.pth"):
168     '''保存模型'''
169     # save model
170     torch.save(model.state_dict(), save_path)
171 
172 
173 def show_curve(ys, title):
174     """
175     plot curlve for Loss and Accuacy
176     Args:
177         ys: loss or acc list
178         title: loss or accuracy
179     """
180     x = np.array(range(len(ys)))
181     y = np.array(ys)
182     plt.plot(x, y, c='b')
183     plt.axis()
184     plt.title('{} curve'.format(title))
185     plt.xlabel('epoch')
186     plt.ylabel('{}'.format(title))
187     plt.show()
188 
189 
190 
191 
192 
193 def fit(model, num_epochs, optimizer, device):
194     """
195     train and evaluate an classifier num_epochs times.
196     We use optimizer and cross entropy loss to train the model.
197     Args:
198         model: CNN network
199         num_epochs: the number of training epochs
200         optimizer: optimize the loss function
```

```
    """

    # loss and optimizer
    loss_func = nn.CrossEntropyLoss()
    model.to(device)
    loss_func.to(device)

    # log train loss and test accuracy
    losses = []
    accs = []
    accst = []
    i = 0
    for epoch in range(num_epochs):

        print('Epoch {}/{}:'.format(epoch + 1, num_epochs))
        # train step
        loss = train(model, train_loader, loss_func, optimizer, device)
        losses.append(loss)

        # evaluate step
        accuracy = evaluate(model, test_loader, device, 'test')
        accuracy1 = evaluate(model, train_loader, device, 'train')
        accs.append(accuracy)
        accst.append(accuracy1)


    # show curve
    show_curve(losses, "train loss")
    show_curve(accs, "test accuracy")
    show_curve(accst, "train accuracy")

# model = models.vgg16_bn(pretrained=True)

# model_ft= models.resnet18(pretrained=True)

import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch
from torchvision import datasets, transforms
from torch.utils import model_zoo
from torch.optim import lr_scheduler

# #hyper parameter
batch_size = 32
num_epochs = 50
lr = 0.0001
num_classes = 25
```

```
252 image_size = 128 #### 64
253
254 path = "train"
255 transform = transforms.Compose([
256     transforms.Resize((128,128)),
257     transforms.RandomRotation((30,30)),
258     transforms.RandomVerticalFlip(0.1),
259     transforms.RandomGrayscale(0.1),
260     transforms.ToTensor(),
261     transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))])
262
263 dataset = datasets.ImageFolder(path, transform=transform)
264
265 print("dataset.classes",dataset.classes)
266 print("dataset.class_to_idx",dataset.class_to_idx)
267 idx_to_class = dict((v, k) for k, v in dataset.class_to_idx.items())
268 print("idx_to_class",idx_to_class)
269 print('len(dataset)', len(dataset))
270
271 """将训练集划分为训练集和验证集"""
272 train_db, val_db = torch.utils.data.random_split(dataset, [2050, 325])#####
273 print('train:', len(train_db), 'validation:', len(val_db))
274
275 # 训练集
276 train_loader = torch.utils.data.DataLoader(
277     train_db,
278     batch_size=batch_size,
279     shuffle=True,
280     drop_last=False)
281 test_loader = torch.utils.data.DataLoader(
282     val_db,
283     batch_size=batch_size,
284     shuffle=True)
285
286 classes = set(['Seashell', 'Lighter', 'Old Mirror', 'Broom', 'Ceramic Bowl', 'Toothbrush', 'Disposable
        Chopsticks', 'Dirty Cloth',
287 'Newspaper', 'Glassware', 'Basketball', 'Plastic Bottle', 'Cardboard', 'Glass Bottle', 'Metalware',
        'Hats', 'Cans', 'Paper',
288 'Vegetable Leaf', 'Orange Peel', 'Eggshell', 'Banana Peel',
289 'Battery', 'Tablet capsules', 'Paint bucket'])
290
291
292 # declare and define an objet of MyCNN
293 mycnn = MyCNN(image_size, num_classes)
294 print(mycnn)
295 # mycnn = getRsn()
296 # mycnn = getMbnet()
297 # os.environ['PATH']
298
299 # x = torch.randn(16, 3, 224, 224).requires_grad_(True)  # 定义一个网络的输入值
300 # y = mycnn(x)    # 获取网络的预测值
```

```
301
302 # MyConvNetVis = make_dot(y, params=dict(list(mycnn.named_parameters()) + [('x', x)]))
303 # MyConvNetVis.format = "png"
304 # # 指定文件生成的文件夹
305 # MyConvNetVis.directory = "result"
306 # # 生成文件
307 # MyConvNetVis.view()
308
309 # device = torch.device('cuda:0')
310 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
311
312 optimizer = torch.optim.Adam(mycnn.parameters(), lr=lr)
313
314 # start training on cifar10 dataset
315 fit(mycnn, num_epochs, optimizer, device)
```

MobileNetV2 代码

```
# Temp 3
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, 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({
    "num_classes": 26, # 分类数，即输出层的维度
    "reduction": 'max', # mean, max, 部分池化采用的方式Head
    "image_height": 224,
    "image_width": 224,
    "batch_size": 26, # 鉴于容器性能，太大可能会导致训练卡住CPU
    "eval_batch_size": 15,
```

```
    "epochs": 35, # 请尝试修改以提升精度
    "lr_max": 0.003, # 请尝试修改以提升精度
    "decay_type": 'square', # 请尝试修改以提升精度
    "momentum": 0.9, # 请尝试修改以提升精度
    "weight_decay": 2.0, # 请尝试修改以提升精度
    "dataset_path": "./datasets/5fbdf571c06d3433df85ac65-momodel/garbage_26x100",
    "features_path": "./results/garbage_26x100_features", # 临时目录，保存冻结层Feature ，可随时删除Map
    "class_index": index,
    "save_ckpt_epochs": 1,
    "save_ckpt_path": './results/ckpt_mobilenetv2',
    "pretrained_ckpt": './src_mindspore/mobilenetv2-200_1067_cpu_gpu.ckpt',
    "export_path": './results/mobilenetv2.mindir'

})

ds = create_dataset(config=config, training=False)
data = ds.create_dict_iterator(output_numpy=True).get_next()
images = data['image']
labels = data['label']

for i in range(1, 5):
    plt.subplot(2, 2, i)
    plt.imshow(np.transpose(images[i], (1,2,0)))
    plt.title('label: %s' % inverted[labels[i]])
    plt.xticks([])
plt.show()

def build_lr(total_steps, lr_init=0.0, lr_end=0.0, 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

steps = 5*93
plt.plot(range(steps), build_lr(steps, lr_max=0.1, decay_type='constant'))
plt.plot(range(steps), build_lr(steps, lr_max=0.1, decay_type='square'))
plt.plot(range(steps), build_lr(steps, lr_max=0.1, decay_type='cosine'))
plt.show()

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)
    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")

    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:
```

```
148         Tensor, output tensor.
149
150     Examples:
151         >>> GlobalAvgPooling()
152     """
153
154     def __init__(self, reduction='mean'):
155         super(GlobalPooling, self).__init__()
156         if reduction == 'max':
157             self.mean = ms.ops.ReduceMax(keep_dims=False)
158         else:
159             self.mean = ms.ops.ReduceMean(keep_dims=False)
160
161     def construct(self, x):
162         x = self.mean(x, (2, 3))
163         return x
164
165
166 class MobileNetV2Head(nn.Cell):
167     """
168     MobileNetV2Head architecture.
169
170     Args:
171         input_channel (int): Number of channels of input.
172         hw (int): Height and width of input, 7 for MobileNetV2Backbone with image(224, 224).
173         num_classes (int): Number of classes. Default is 1000.
174         reduction: mean or max, which means AvgPooling or MaxpPooling.
175         activation: Activation function for output logits.
176     Returns:
177         Tensor, output tensor.
178
179     Examples:
180         >>> MobileNetV2Head(num_classes=1000)
181     """
182
183     def __init__(self, input_channel=1280, hw=7, num_classes=1000, reduction='mean', activation="None"):
184         super(MobileNetV2Head, self).__init__()
185         if reduction:
186             self.flatten = GlobalPooling(reduction)
187         else:
188             self.flatten = nn.Flatten()
189             input_channel = input_channel * hw * hw
190         self.dense = nn.Dense(input_channel, num_classes, weight_init='ones', has_bias=False)
191         if activation == "Sigmoid":
192             self.activation = nn.Sigmoid()
193         elif activation == "Softmax":
194             self.activation = nn.Softmax()
195         else:
196             self.need_activation = False
197
198     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)
    eval_dataset = create_dataset(config=config)
    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=0,
        decay_type=config.decay_type)
    opt = nn.Momentum(head.trainable_params(), lrs, config.momentum, config.weight_decay)
    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))
    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))
        if (epoch + 1) % config.save_ckpt_epochs == 0:
            save_checkpoint(network, os.path.join(config.save_ckpt_path, f"mobilenetv2-{epoch+1}.ckpt"))

    # evaluate
    print('validating the model...')
```

```
248     eval_model = Model(network, loss, metrics={'acc', 'loss'})
249     acc = eval_model.eval(eval_dataset, dataset_sink_mode=False)
250     print(acc)
251 
252     return history
253 
254 if os.path.exists(config.save_ckpt_path):
255     shutil.rmtree(config.save_ckpt_path)
256 os.makedirs(config.save_ckpt_path)
257 
258 history = train_head()
259 
260 plt.plot(history, label='train_loss')
261 plt.legend()
262 plt.show()
263 
264 CKPT = f'mobilenetv2-{config.epochs}.ckpt'
265 print("Chosen checkpoint is", CKPT)
266 
267 def image_process(image):
268     """Precess one image per time.
269 
270     Args:
271         image: shape (H, W, C)
272     """
273     mean=[0.485*255, 0.456*255, 0.406*255]
274     std=[0.229*255, 0.224*255, 0.225*255]
275     image = (np.array(image) - mean) / std
276     image = image.transpose((2,0,1))
277     img_tensor = Tensor(np.array([image], np.float32))
278     return img_tensor
279 
280 def infer_one(network, image_path):
281     image = Image.open(image_path).resize((config.image_height, config.image_width))
282     logits = network(image_process(image))
283     pred = np.argmax(logits.asnumpy(), axis=1)[0]
284     print(image_path, inverted[pred])
285     return pred
286 
287 def infer(images):
288     backbone = MobileNetV2Backbone()
289     head = MobileNetV2Head(input_channel=backbone.out_channels, num_classes=config.num_classes,
            reduction=config.reduction)
290     network = mobilenet_v2(backbone, head)
291     print('加载模型路径:',os.path.join(config.save_ckpt_path, CKPT))
292     load_checkpoint(os.path.join(config.save_ckpt_path, CKPT), net=network)
293     for img in images:
294         infer_one(network, img)
295 
296 test_images = list()
297 folder = os.path.join(config.dataset_path, 'val/00_08') # Hats
```

```
for img in os.listdir(folder):
    test_images.append(os.path.join(folder, img))

infer(test_images)

backbone = MobileNetV2Backbone()
# 导出带有层的模型Softmax
head = MobileNetV2Head(input_channel=backbone.out_channels, num_classes=config.num_classes,
                       reduction=config.reduction, activation='Softmax')
network = mobilenet_v2(backbone, head)
load_checkpoint(os.path.join(config.save_ckpt_path, CKPT), net=network)

input = np.random.uniform(0.0, 1.0, size=[1, 3, 224, 224]).astype(np.float32)
export(network, Tensor(input), file_name=config.export_path, file_format='MINDIR')

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')

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'}

config = EasyDict({
    "num_classes": 26,
    "reduction": 'mean',
    "image_height": 224,
    "image_width": 224,
    "eval_batch_size": 10
})

```

```
# 4. 自定义模型部分Head
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


backbone = MobileNetV2Backbone()
head = MobileNetV2Head(input_channel=backbone.out_channels, num_classes=config.num_classes,
    reduction=config.reduction)
network = mobilenet_v2(backbone, head)

model_path = './results/ckpt_mobilenetv2/mobilenetv2-4.ckpt'
load_checkpoint(model_path, net=network)

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

    Args:
        image: shape (H, W, C)
```

```
396     """
397     mean=[0.485*255, 0.456*255, 0.406*255]
398     # mean=[0.456*255, 0.406*255, 0.485*255]
399     std=[0.229*255, 0.224*255, 0.225*255]
400     # std=[0.224*255, 0.225*255, 0.229*255]
401     image = (np.array(image) - mean) / std
402     image = image.transpose((2,0,1))
403     img_tensor = Tensor(np.array([image], np.float32))
404     return img_tensor
405 
406 def predict(image):
407 
408     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
409     image = cv2.resize(image,(config.image_height, config.image_width))
410     image = image_process(image)
411 
412     logits = network(image)
413     pred = np.argmax(logits.asnumpy(), axis=1)[0]
414 
415     return inverted[pred]
416 
417 # 输入图片路径和名称
418 image_path = './datasets/5fbdf571c06d3433df85ac65-momodel/garbage_26x100/val/00_00/00037.jpg'
419 
420 # 使用 opencv 读取图片
421 image = cv2.imread(image_path)
422 
423 # 打印返回结果
424 print(predict(image))
```