手写数字识别与垃圾分类

——AI 课程设计作业 3

实验报告

计算机科学与技术学院

2352018 刘彦

2025 年 5 月 20 日

AI 课程设计作业 3

——深度学习模块：手写数字识别与垃圾分类

2352018 刘彦

1. 问题概述

⑴问题的直观描述

①实验内容

A.LeNet-5+MNIST：深度学习入门实践

LeNet-5+MNIST 被誉为深度学习的“Hello world”，是入门者学习图像分类的经典示例，有助于理解网络结构、数据加载、模型训练与评估的基本流程。

该任务的目的是了解深度学习基本流程，掌握 MindSpore 框架基本使用。任务内容是使用 MindSpore 框架构建 LeNet-5 神经网络，在 MNIST 手写数字数据集上训练模型，验证模型准确率。

B.MobileNetV2+垃圾分类数据集：模型微调实践

预训练模型能够显著加快训练速度，提升模型精度，并有效减少陷入局部最优解和过拟合的风险。在深度学习实践中，常用的预训练数据集包括 ImageNet、COCO、VOC 等，这些大规模数据集提供了丰富的特征提取能力。微调（Fine-Tune）作为一种在预训练模型基础上调整参数以适应新任务的方法，已成为工业界广泛采用的策略，尤其适用于计算资源有限或样本量较小的场景。

该任务目的是掌握基于预训练模型进行 Fine-Tune，以适配特定领域任务。通过使用 MindSpore 加载 MobileNetV2 预训练模型，并在 26 类垃圾分类数据集上进行迁移学习与微调，学习如何在 CPU 或 GPU 平台上高效训练模型，最终评估模型在特定任务中的分类性能。

总体来说，本实验要求通过两个实践项目，从基础到进阶，掌握 MindSpore 框架使用，熟悉 LeNet-5 与 MobileNetV2 两种网络结构的使用方法，理解从零训练与模型微调的差异，掌握在特定数据集上进行图像分类任务的完整流程，从而提升部署 AI 模型的能力。

②实验要求

本实验要求学生掌握使用 MindSpore 框架进行图像分类任务的基本流程，具体包括：了解如何使用 MindSpore 开发简单的卷积神经网络，掌握在简单图片分类任务中的模型训练方法，以及学习如何对分类模型进行效果验证，为后续更复杂的图像识别任务打下基础。

同时，在实验开始前，本实验要求学生具备一定的预备知识，以顺利完成相关任务。这些知识包括：熟练使用 Python，了解 Shell 与 Linux 系统的基本操作，并能在 CPU/GPU 环

境下进行调试；掌握深度学习的基础理论，如卷积神经网络、损失函数、优化器及训练策略等；熟悉 MindSpore 深度学习框架的使用，并能够通过其官网 https://www.mindspore.cn 获取相关资源。

⑵解决问题的思路分析

第一阶段的任务是使用 MindSpore 框架构建 LeNet-5 神经网络，并在 MNIST 手写数字数据集上进行训练与测试。在本任务中，我的主要目标是掌握深度学习图像分类任务的完整流程。LeNet-5 是一个结构相对简单的卷积神经网络，适用于初学者理解网络结构的基本单元，如卷积层、池化层和全连接层。MNIST 手写数字数据集则为入门提供了理想的测试环境，数据处理难度适中且结果直观可视化。

解决问题的第一步是数据预处理。我计划使用 MindSpore 提供的 MnistDataset 类加载训练集和测试集，同时进行图像的归一化与维度调整，以适配 LeNet-5 的输入要求（如 28x28 灰度图）。接着定义 LeNet-5 网络结构，明确每一层的功能和参数设置。

在模型训练阶段，我的思路是配置合适的损失函数（如交叉熵）和优化器（如 Adam 或 Momentum），并设置合适的学习率与训练周期。通过 MindSpore 的训练接口，执行多轮迭代训练，实时观察损失下降和准确率上升过程，借此理解训练动态。

然后，我计划对模型在测试集上的表现进行评估，使用准确率等指标判断模型性能，并进行可视化输出。本实验的思路是完成从“搭建网络—加载数据—模型训练—性能评估”的基本流程，为后续复杂任务打下基础。

第二阶段的任务引入了更具实用性的场景——垃圾图像分类，并采用 MobileNetV2 预训练模型进行微调。本任务目标是学习如何基于已有的预训练模型（MobileNetV2）进行迁移学习和模型微调，从而完成特定领域的分类任务。本项目使用的垃圾分类数据集具有较多类别（26 类）且图像风格 差异较大，较 MNIST 任务更贴近现实场景。

我计划解决问题的第一步是理解预训练模型的结构和加载方式。MobileNetV2 作为轻量级网络，其特征提取能力强、计算效率高，适用于移动端和资源受限设备。

在数据准备阶段，需要将垃圾分类图像进行清洗、标签编码、尺寸缩放（一般调整到 224x224）等预处理工作，并划分训练集与验证集。在网络结构上，需要替换 MobileNetV2 最后的分类头（通常是全连接层），使其输出节点数从原始 ImageNet 的 1000 类调整为本任务的 26 类。

微调过程中，我们通常“冻结”网络的其他部分，仅训练 MobileNetV2Head（两个全连接层），避免过拟合。训练配置上设置较低学习率、合适的 batch size，并在训练过程中监控验证集的 loss 与 accuracy，选择最优模型保存。

2.模型设计

⑴CNN 模型介绍

卷积神经网络（Convolutional Neural Network, CNN）是一种专门为处理结构化网格数据（如图像、时间序列）设计的深度学习模型，广泛应用于计算机视觉任务，如图像分类、目标检测和人脸识别。

CNN 通过模拟人类视觉系统，结合局部感受区域(Local Receptive Field)、权重共享(Weight Sharing)和空间子采样(Spatial Subsampling)，有效提取图像的空间层次特征。与全连接神经网络相比，CNN 参数更少、计算效率更高，适合处理高维输入如图像。

局部感受区域、权重共享和空间子采样是 CNN 的核心设计理念，共同构成 CNN 的高效特征提取机制。局部感受区域指卷积核仅关注输入图像的局部区域，提取边缘、纹理等特征，模拟生物视觉的局部响应，显著减少参数量相比全连接层。权重共享通过在整个特征图上复用同一卷积核的权重，降低计算复杂度，并赋予平移不变性，适合检测位置无关的模式。空间子采样通过池化操作，降低特征图维度（如从 24×24 到 12×12），减少计算量，增强对平移、旋转的鲁棒性，同时提取更抽象的特征。

CNN 核心组件包括卷积层、激活函数、池化层、全连接层和输出层。卷积层通过局部感受区域和权重共享提取图像局部特征，如边缘和纹理，显著减少参数量并增强平移不变性。激活函数（如 ReLU）引入非线性，使模型能捕捉复杂模式。池化层通过最大或平均池化降低特征图维度，减少计算量并提升鲁棒性。全连接层整合特征为高维向量，输出层则生成最终预测，如使用 softmax 输出分类概率。这些组件协同工作，使 CNN 在图像分类、目标检测等计算机视觉任务中表现出色。

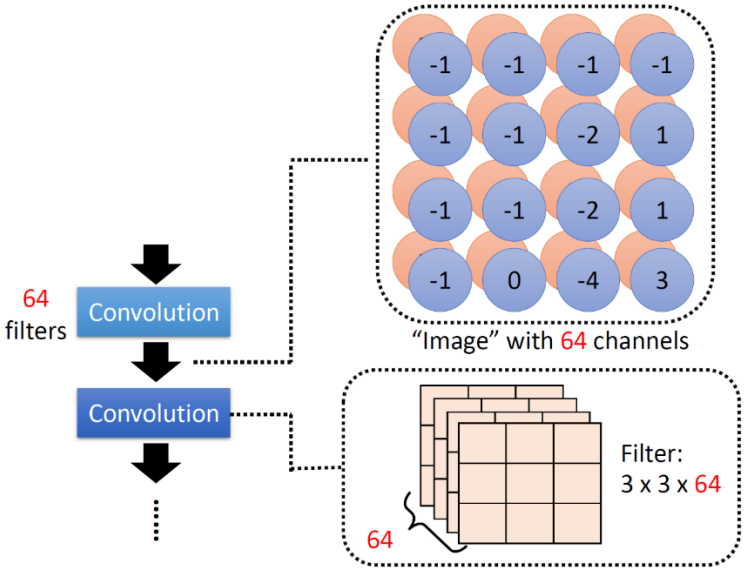


图 1 CNN 卷积层结构示意图[1]

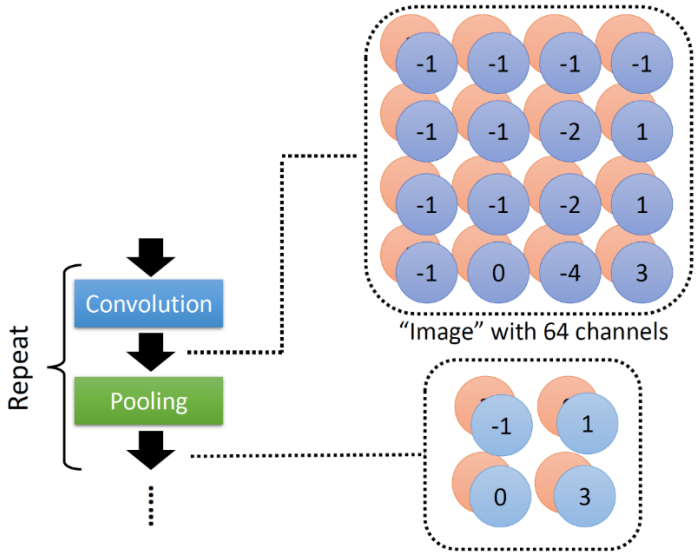


图 2 CNN 池化层结构示意图[1]

⑵模型架构

①LeNet-5

LeNet-5 是一个 7 层（不包括输入层）的卷积神经网络，包含卷积层、池化层（子采样层）和全连接层。它的设计体现了 CNN 的核心思想：局部感受区域(Local Receptive Field)、权重共享(Weight Sharing)和空间子采样(Spatial Subsampling)。

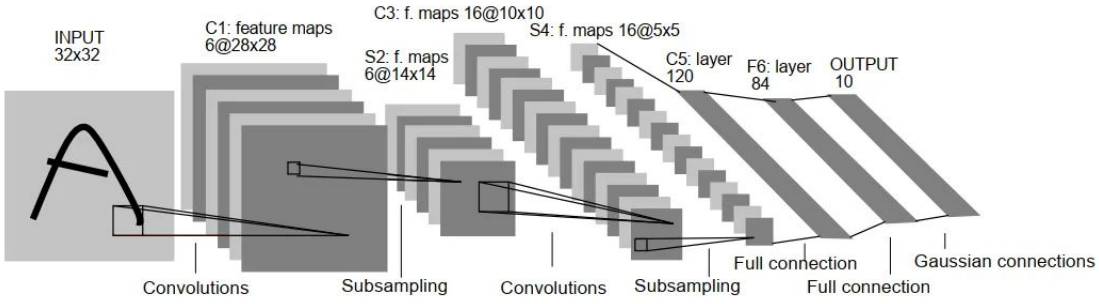


图 3 LeNet-5 模型结构图[2]

A.卷积层（C1, C3, C5）

卷积层通过卷积操作提取图像的局部特征。每个卷积核与输入特征图进行卷积，生成新的特征图(Feature Map)。

对于特征图 X，权重 W，偏置 b，激活函数 σ（在 LeNet-5 中常为 tanh），输出特征图的计算公式为：

$$O_{ij}^{(k)} = \sigma\left(\sum_{m=1}^{M} W_m^{(k)} * X^{(m)} + b^{(k)}\right)$$

B.池化层（S2, S4）

池化层（子采样）通过降采样减少特征图的空间维度，增强平移不变性。LeNet-5 使用平均池化。

对于特征图 X，池化区域大小 k×k，步幅 s，激活函数 σ（在 LeNet-5 中常为 tanh），输出特征图的计算公式为：

$$O_{ij}^{(k)} = \sigma(\beta \cdot \frac{1}{k^2} \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} X_{ki+a,ki+b}^{(k)} + b^{(k)})$$

C.全连接层（F6）

将卷积/池化提取的特征整合为高维向量，LeNet-5 的 F6 层将 120 个神经元映射到 84 个神经元，用于分类或回归。

$$\boldsymbol{y_i} = \boldsymbol{\sigma}(\sum_{j} \boldsymbol{w_{ij}} \cdot \boldsymbol{x_j} + \boldsymbol{b_i})$$

D.输出层

生成最终预测，LeNet-5 的输出层有 10 个神经元，对应 0-9 数字。原始 LeNet-5 使用 RBF 损失函数，现代实现常用 softmax。

RBF 损失：$E = \sum_1^{10} \|y_i - t_i\|^2$

Softmax：$P(y_i) = \frac{e^{z_i}}{\sum_{i=1}^{10} e^{z_j}}$，损失函数通常为交叉熵：$L = -\sum_i t_i \log(P(y_i))$。

②MobileNetV2

MobileNetV2 是 Google 提出的轻量级卷积神经网络，适用于移动和嵌入式设备。其核心思想是：利用深度可分离卷积（Depthwise Separable Convolution）和倒残差结构（Inverted Residuals）来减少参数量与计算开销，同时保持较高精度。网络整体结构包括：初始标准卷积层（Conv2d + BN + ReLU6），多个倒残差模块（Inverted Residual Blocks），最后进行全局平均池化，采用全连接层输出分类结果。

图 4 MobileNetV2 模型结构图

特别的，MobileNetV2 采用非线性激活函数 ReLU6，相比传统 ReLU，ReLU6 限制最大值为 6，定义为：

$$ReLU6(x) = \min(\max(0, x), 6)$$

能更好地适应低精度计算（如量化）需求，提高模型在移动设备上的稳定性。

A.深度可分离卷积(Depthwise Separable Convolution)

将标准卷积分解为两步：深度卷积（Depthwise Convolution）和 逐点卷积（Pointwise Convolution），大幅减少计算量。

在卷积核尺寸 D_K，特征图尺寸 D_F，输入输出通道数为 M 和 N 时，普通卷积和深度可分离卷积计算分别如下：

$$Convolution:\ D_K \times D_K \times M \times N \times D_F \times D_F$$

$$Depthwise\ Separable\ Convolution:\ D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F$$

● 深度卷积

每个通道独立卷积，核大小通常为 3×3。对于特征图 X，权重 W，输出通道 Y：

$$Y_{ij}^{(k)} = \sigma(\sum_{p,q} W_{p,q}^{(k)} * X_{i+p,i+q}^{(k)} + b^{(k)})$$

● 逐点卷积

1×1 卷积，用于融合通道信息。对于 1×1 卷积核 W，融合 k 个输入通道到 m 个输出通道：

$$Y_{ij}^{(m)} = \sigma(\sum_{k} W_{1,1,k}^{(m)} * X_{i,j}^{(k)} + b^{(m)})$$

在 MobileNetV2 中，每个瓶颈残差块的深度卷积处理空间特征，逐点卷积融合通道信息，显著降低参数量。

B.倒残差结构(Inverted Residuals)

与 ResNet 的残差结构（先压缩再扩展）相反，MobileNetV2 先通过 1×1 卷积扩展通道数（高维），再进行深度卷积，最后通过 1×1 卷积压缩通道数（低维）。

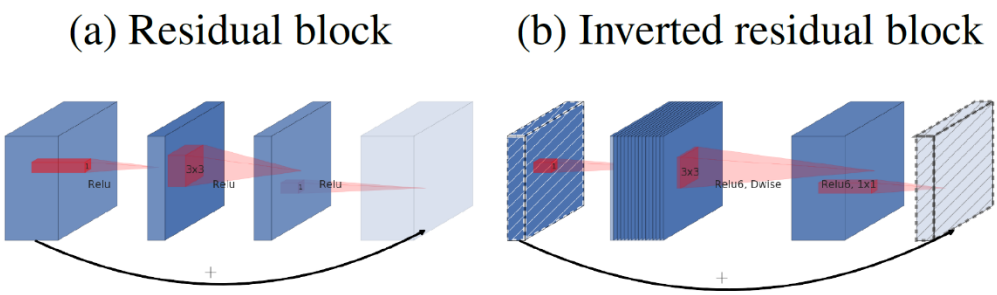


图 5 MobileNetV2 与 ResNet 残差块结构对比图[3]

与 ResNet 的残差结构（先压缩再扩展）相反，MobileNetV2 先通过 1×1 卷积扩展通道数（高维），再进行深度卷积，最后通过 1×1 卷积压缩通道数（低维）。在 ResNet 提出的 Residuals 结构中,先使用 1×1 卷积实现降维,然后通过 3×3 卷积,最后通过 1×1 卷积实现升维,即两头大中间小。在 MobileNetV2 中,将降维和升维的顺序进行了调换, 其核心流程是：先用 1×1 卷积进行升维（Expansion Layer），再用计算量极低的 3×3 深度可分离卷积（Depthwise Convolution）提取特征，最后通过另一个 1×1 卷积降维（Projection Layer）。,即两头小中间大。

在 MobileNetV2 中，Inverted Residuals 的结构与 ResNet 不同的是，只有当 stride=1 且输入特征矩阵与输出特征矩阵 shape 相同的时候才有 shortcut 连接。每个残差块使用扩展因子 t=6，先扩展通道（如 32→192），再深度卷积，最后压缩通道并添加残差连接。

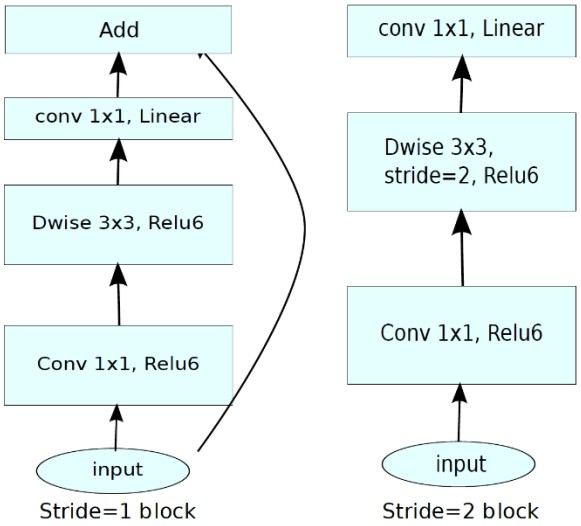


图 6 MobileNetV2 的 Inverted Residuals 网络结构图[3]

特别的，为了解决 ReLU 在低维空间可能丢失信息的问题，MobileNetV2 在压缩层（逐点卷积）后不使用非线性激活（如 ReLU6），直接输出线性结果保留更多特征。因为 Inverted Residuals 结构两头小中间大，在最后输出的时候是一个低维的特征了，所以改为线性激活函数而非 ReLU 激活函数。Sandler M(2018)在实验中证明[3]，使用 linear bottleneck 可以防止非线性破坏太多信息。

C.全局平均池化 (Global Average Pooling)

将特征图的空间维度（H×W）压缩为，生成固定长度的向量。MobileNetV2 在残差块后，特征图（如 7×7×1280）通过全局平均池化，生成 1280 维向量，送入全连接层。

$$Y_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_{i,j}^{(k)}$$

⑵模型功能

LeNet-5+MNIST：LeNet-5 是一种经典的卷积神经网络，旨在解决手写数字识别问题，在 MNIST 数据集的 28×28 灰度图像上，通过卷积层提取边缘和纹理特征，池化层降维并增强鲁棒性，最终通过全连接层输出 0-9 数字的分类结果。训练过程完成后，模型在测试集上输出分类准确率（通常超过 98%），并可视化预测结果。

MobileNetV2+垃圾分类数据集：MobileNetV2 是一种轻量级卷积神经网络，针对移动设备优化，旨在高效处理图像分类任务，在垃圾分类实验中，利用 ImageNet 预训练的 MobileNetV2，通过迁移学习和微调适配 26 类垃圾分类数据集，目标是掌握预训练模型的使用和在特定任务上的高效训练，训练后输出分类准确率和损失曲线。

⑶对 MindSpore 的介绍

MindSpore 是华为于 2019 年推出的开源深度学习框架，旨在为开发者提供高效、灵活和易用的工具，构建和部署人工智能模型，特别在图像分类、目标检测和自然语言处理等任务中表现出色。

框架通过统一的数据处理接口加载和预处理数据集，支持数据增强和高效批处理；通过灵活的网络构建 API（如 nn.Module 和 Cell），开发者可快速定义模型结构或加载预训练模型；利用自动微分和多种优化器（如 Adam、SGD）进行高效训练，支持动态图（便于调试）和静态图（优化推理）混合编程；最终模型可部署到 CPU、GPU 或华为 Ascend 芯片，输出预测结果并可视化性能（如准确率曲线、分类报告）[4]。

使用方法示例如下：

```
import mindspore
import mindspore.nn as nn
import mindspore.dataset as ds
# 定义简单神经网络
class SimpleNet(nn.Cell):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(28 * 28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Dense(128, 10)
    def construct(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

⑷模型思路与流程

①LeNet-5

LeNet-5 旨在解决手写数字识别问题，通过在 MNIST 数据集（28×28 灰度图像）上构建简洁的 CNN，首先通过卷积层扫描图像，提取边缘和纹理等局部特征；然后通过池化层压缩特征图尺寸，减少计算量并增强对位置变化的鲁棒性；最后通过全连接层整合特征，输出 0-9 数字的分类结果。

②MobileNetV2

MobileNetV2 旨在高效处理复杂图像分类任务，针对移动设备优化，首先加载 ImageNet 预训练模型，利用其通用特征提取能力；通过扩展层增加通道数以丰富特征表示，深度卷积处理空间信息，压缩层减少通道数并通过残差连接优化训练稳定性；最后通过全局平均池化

和全连接层输出分类结果。在垃圾分类实验中，流程聚焦迁移学习，微调预训练模型以适配26类垃圾分类数据集，调整高层参数以优化性能。

3. 模型实现

由于全部代码过长，这里只呈现关键部分，其余部分见源代码文件。

⑴LeNet-5 的主要模型架构函数

①LeNet-5 网络结构

```
class LeNet5(nn.Cell):
    def __init__(self):
        super(LeNet5, self).__init__()
        # 输出: (24, 24, 6)
        self.conv1 = nn.Conv2d(1, 6, 5, pad_mode='valid')
        # 输出: (8, 8, 16)
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        # 计算展平后的特征维度：16 * 4 * 4 = 256
        self.fc1 = nn.Dense(256, 120)  # 修改输入维度为 256
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, 10)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
    def construct(self, x):
        # 输入 x: (batch_size, 1, 28, 28)
        x = self.conv1(x)                    # -> (batch_size, 6, 24, 24)
        x = self.relu(x)
        x = self.max_pool2d(x)               # -> (batch_size, 6, 12, 12)
        x = self.conv2(x)                    # -> (batch_size, 16, 8, 8)
        x = self.relu(x)
        x = self.max_pool2d(x)               # -> (batch_size, 16, 4, 4)
        x = self.flatten(x)                  # -> (batch_size, 256)
        x = self.relu(self.fc1(x))           # -> (batch_size, 120)
        x = self.relu(self.fc2(x))           # -> (batch_size, 84)
        x = self.fc3(x)                      # -> (batch_size, 10)
        return x
```

②LeNet-5 训练函数

```
def train_model(train_dataset, test_dataset, epochs,
if_early_stop=False):
    # 定义网络
    network = LeNet5()
    # 定义损失函数和优化器
```

```
    net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True,
reduction='mean')
    net_opt = nn.Momentum(network.trainable_params(),
learning_rate=0.01, momentum=0.9)
    # 定义训练模型
    model = ms.Model(network, net_loss, net_opt, metrics={'accuracy'})
    # 用于记录训练过程
    train_losses = []
    train_accuracies = []
    test_accuracies = []
    # 获取总步数用于进度显示
    steps_per_epoch = train_dataset.get_dataset_size()
    class ProgressCallback(ms.train.callback.Callback):
        def __init__(self, epochs, steps_per_epoch):
            super().__init__()
            self.pbar = tqdm(total=epochs * steps_per_epoch,
                             desc='Training',
                             unit='step')
        def step_end(self, run_context):
            self.pbar.update(1)
        def epoch_end(self, run_context):
            self.pbar.refresh()
        def train_end(self, run_context):
            self.pbar.close()
    class CustomCallback(ms.train.callback.Callback):
        def __init__(self, model, test_dataset):
            super().__init__()
            self.model = model
            self.test_dataset = test_dataset
            self.epoch_losses = []
            self.current_epoch = 0
        def step_end(self, run_context):
            cb_params = run_context.original_args()
            loss = cb_params.net_outputs.asnumpy()
            self.epoch_losses.append(loss)
        def epoch_end(self, run_context):
            self.current_epoch += 1
            # 计算当前 epoch 的平均损失
            avg_loss = np.mean(self.epoch_losses)
            train_losses.append(avg_loss)
            self.epoch_losses = []
            # 计算训练集准确率
            train_metrics = self.model.eval(train_dataset)
            train_accuracies.append(train_metrics['accuracy'])
```

```
            # 计算测试集准确率
            test_metrics = self.model.eval(test_dataset)
            test_accuracies.append(test_metrics['accuracy'])
            # 打印当前 epoch 的结果
            print(f"\nEpoch [{self.current_epoch}/{epochs}]")
            print(f"损失: {avg_loss:.4f}")
            print(f"训练集准确率: {train_metrics['accuracy']:.4f}")
            print(f"测试集准确率: {test_metrics['accuracy']:.4f}")
    class EarlyStopCallback(ms.train.callback.Callback):
        # 这个类是定义早停机制的类，具体实现不在这里展示
    # 创建回调
    progress_cb = ProgressCallback(epochs, steps_per_epoch)
    custom_cb = CustomCallback(model, test_dataset)
    early_stop_cb = EarlyStopCallback(patience=3)
    if if_early_stop:
        callbacks = [progress_cb, custom_cb, early_stop_cb]
    else:
        callbacks = [progress_cb, custom_cb]
    # 训练模型
    print(f"\n开始训练，共 {epochs} 个 epoch:")
    print(f"早停条件: {early_stop_cb.patience} 个 epoch 内验证集损失无改善
")
    model.train(epochs, train_dataset, callbacks=callbacks)
    # 如果发生早停，调整训练历史图表的范围
    # 绘制训练过程
    # 绘制损失值变化
    # 绘制准确率变化
    return model
```

⑵MobileNetV2 的主要微调设置函数

①MobileNetV2 的 Backbone 网络

```
class MobileNetV2Backbone(nn.Cell):
    """
    MobileNetV2 architecture.
    Args:
        class_num (int): number of classes.
        width_mult (int): Channels multiplier for round to 8/16 and
others. Default is 1.
        has_dropout (bool): Is dropout used. Default is false
        inverted_residual_setting (list): Inverted residual settings.
Default is None
        round_nearest (list): Channel round to . Default is 8
    Returns:
        Tensor, output tensor.
```

```
    Examples:
        >>> MobileNetV2(num_classes=1000)
    """
    def __init__(self, width_mult=1., inverted_residual_setting=None,
round_nearest=8,
                 input_channel=32, last_channel=1280):
        super(MobileNetV2Backbone, self).__init__()
        block = InvertedResidual
        # setting of inverted residual blocks
        self.cfgs = inverted_residual_setting
        if inverted_residual_setting is None:
            self.cfgs = [  # t, c, n, s
                [1, 16, 1, 1],
                [6, 24, 2, 2],
                [6, 32, 3, 2],
                [6, 64, 4, 2],
                [6, 96, 3, 1],
                [6, 160, 3, 2],
                [6, 320, 1, 1],
            ]
        # building first layer
        input_channel = _make_divisible(input_channel * width_mult,
round_nearest)
        self.out_channels = _make_divisible(last_channel * max(1.0,
width_mult), round_nearest)
        features = [ConvBNReLU(3, input_channel, stride=2)]
        # building inverted residual blocks
        for t, c, n, s in self.cfgs:
            output_channel = _make_divisible(c * width_mult,
round_nearest)
            for i in range(n):
                stride = s if i == 0 else 1
                features.append(block(input_channel, output_channel,
stride, expand_ratio=t))
                input_channel = output_channel
        # building last several layers
        features.append(ConvBNReLU(input_channel, self.out_channels,
kernel_size=1))
        # make it nn.CellList
        self.features = nn.SequentialCell(features)
        self._initialize_weights()
    def construct(self, x):
        x = self.features(x)
        return x
```

```
    def _initialize_weights(self):
        """
        Initialize weights.
        Args:
        Returns:
            None.
        """
```

②MobileNetV2 的 Head 网络

```
class MobileNetV2Head(nn.Cell):
    """
    MobileNetV2 architecture.
    Args:
        class_num (int): Number of classes. Default is 1000.
        has_dropout (bool): Is dropout used. Default is false
    Returns:
        Tensor, output tensor.
    Examples:
        >>> MobileNetV2(num_classes=1000)
    """
    def __init__(self, input_channel=1280, num_classes=1000,
has_dropout=False, activation="None"):
        super(MobileNetV2Head, self).__init__()
        # mobilenet head
        head = ([GlobalAvgPooling(), nn.Dense(input_channel,
num_classes, has_bias=True)] if not has_dropout else
                [GlobalAvgPooling(), nn.Dropout(0.2),
nn.Dense(input_channel, num_classes, has_bias=True)])
        self.head = nn.SequentialCell(head)
        self.need_activation = True
        if activation == "Sigmoid":
            self.activation = P.Sigmoid()
        elif activation == "Softmax":
            self.activation = P.Softmax()
        else:
            self.need_activation = False
        self._initialize_weights()
    def construct(self, x):
        x = self.head(x)
        if self.need_activation:
            x = self.activation(x)
        return x
    def _initialize_weights(self):
        """
        Initialize weights.
```

```
        Args:
        Returns:
            None.
        """
    @property
    def get_head(self):
        return self.head
```

③MobileNetV2 的 Inverted Residual 结构

```
class InvertedResidual(nn.Cell):
    """
    Mobilenetv2 residual block definition.
    Args:
        inp (int): Input channel.
        oup (int): Output channel.
        stride (int): Stride size for the first convolutional layer.
Default: 1.
        expand_ratio (int): expand ration of input channel
    Returns:
        Tensor, output tensor.
    """
    def __init__(self, inp, oup, stride, expand_ratio):
        super(InvertedResidual, self).__init__()
        assert stride in [1, 2]
        hidden_dim = int(round(inp * expand_ratio))
        self.use_res_connect = stride == 1 and inp == oup
        layers = []
        if expand_ratio != 1:
            layers.append(ConvBNReLU(inp, hidden_dim, kernel_size=1))
        layers.extend([
            ConvBNReLU(hidden_dim, hidden_dim,
                       stride=stride, groups=hidden_dim),   # dw
            nn.Conv2d(hidden_dim, oup, kernel_size=1,
                      stride=1, has_bias=False),   # pw-linear
            nn.BatchNorm2d(oup),
        ])
        self.conv = nn.SequentialCell(layers)
        self.add = TensorAdd()
        self.cast = P.Cast()
    def construct(self, x):
        identity = x
        x = self.conv(x)
        if self.use_res_connect:
            return self.add(identity, x)
        return x
```

④初始化网络函数

```
def define_net(config, is_training):
    backbone_net = MobileNetV2Backbone()
    activation = config.activation if not is_training else "None"
    head_net = MobileNetV2Head(input_channel=backbone_net.out_channels,
                               num_classes=config.num_classes,
                               activation=activation)
    net = mobilenet_v2(backbone_net, head_net)
    return backbone_net, head_net, net
```

⑤微调方法

```
    if args_opt.pretrain_ckpt == "" or args_opt.freeze_layer == "none":
        loss_scale = FixedLossScaleManager(config.loss_scale,
drop_overflow_update=False)
        opt = Momentum(filter(lambda x: x.requires_grad,
net.get_parameters()), lr,
                       config.momentum, config.weight_decay,
config.loss_scale)
        model = Model(net, loss_fn=loss, optimizer=opt,
loss_scale_manager=loss_scale, metrics={"Accuracy": Accuracy()})
        cb = config_ckpoint(config, lr, step_size)
        print("============== Starting Training ==============")
        for epoch in range(epoch_size):
            model.train(1,dataset, callbacks=cb, dataset_sink_mode=True)
        print("============== End Training ==============")
    else:
        opt = Momentum(filter(lambda x: x.requires_grad,
head_net.get_parameters()), lr,
                       config.momentum, config.weight_decay)
        network = WithLossCell(head_net, loss)
        network = TrainOneStepCell(network, opt)
        network.set_train()
        features_path = args_opt.dataset_path + '_features'
        idx_list = list(range(step_size))
        rank = 0
        if config.run_distribute:
            rank = get_rank()
        save_ckpt_path = os.path.join(config.save_checkpoint_path,
'ckpt_' + str(rank) + '/')
        os.makedirs(save_ckpt_path, exist_ok=True)
        for epoch in range(epoch_size):
            random.shuffle(idx_list)
            epoch_start = time.time()
            epoch_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")))
                loss_val = network(feature, label).asnumpy()
                epoch_losses.append(loss_val)
            mean_loss = np.mean(epoch_losses)
            losses_list.append(mean_loss)
            epoch_mseconds = (time.time() - epoch_start) * 1000
            per_step_mseconds = epoch_mseconds / step_size
            print(f"epoch[{epoch+1}/{epoch_size}], iter[{step_size}]
cost: {epoch_mseconds:.3f}, "
                  f"per step time: {per_step_mseconds:.3f}, avg loss:
{mean_loss:.3f}")
            if (epoch + 1) % 10 == 0:
                print(f"===== Eval at epoch {epoch + 1} =====")
                net.set_train(False)
                eval_dataset = create_dataset(dataset_path='./data_new',
do_train=False, config=config)
                eval_model = Model(net, loss_fn=loss,
metrics={'Accuracy': Accuracy()})
                eval_result = eval_model.eval(eval_dataset)

                acc_val = eval_result.get("Accuracy", 0.0)
                accs_list.append(acc_val)
                epochs_record.append(epoch + 1)
                print(f"Epoch {epoch + 1} eval result: {eval_result}")
                net.set_train(True)
            if (epoch + 1) % config.save_checkpoint_epochs == 0:
                save_checkpoint(net, os.path.join(save_ckpt_path,
f"mobilenetv2_{epoch+1}.ckpt"))
        print("total cost {:.4f} s".format(time.time() - start))
```

这段代码展示了两种训练策略：

第一种是从头训练（或不冻结层），未加载预训练模型或不冻结任何层，即训练整个网络。使用 Momentum 优化器 + FixedLossScaleManager。构建 Model 对象（包含网络、损失函数、优化器等）。最后基于 MindSpore 的 Model.train()方法，常规方式训练完整模型。

第二种是加载预训练模型后，仅训练头部分类器（head_net），也是我在本次实验中使用的方法，只训练网络的“头部”结构 head_net。首先构建训练单步单元 TrainOneStepCell，加载预训练特征（feature_*.npy 和 label_*.npy），不再从原始数据训练全模型。手动循环控制训练，每次从特征文件加载 batch，手动前向+反向+优化。并且定期进行评估，保存模型 checkpoint。

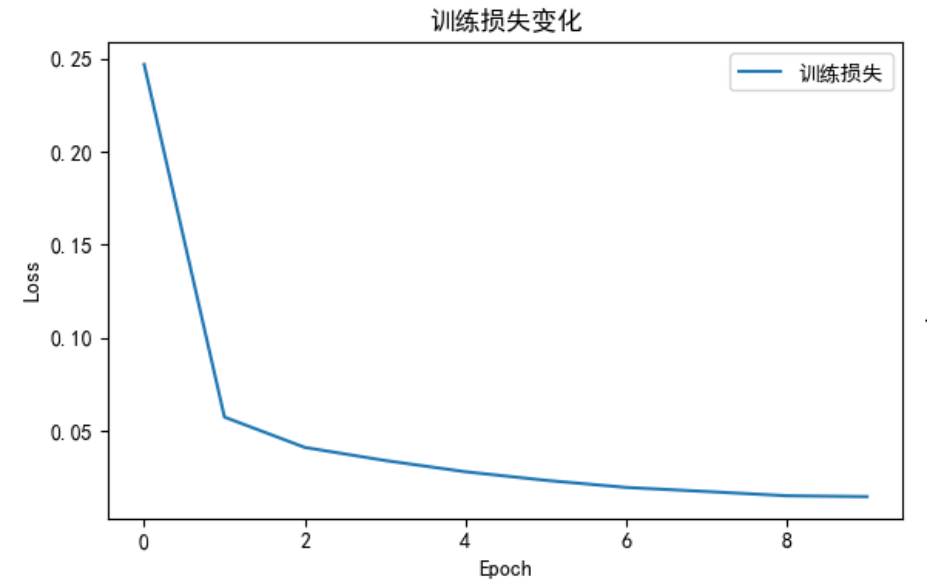
⑥配置参数

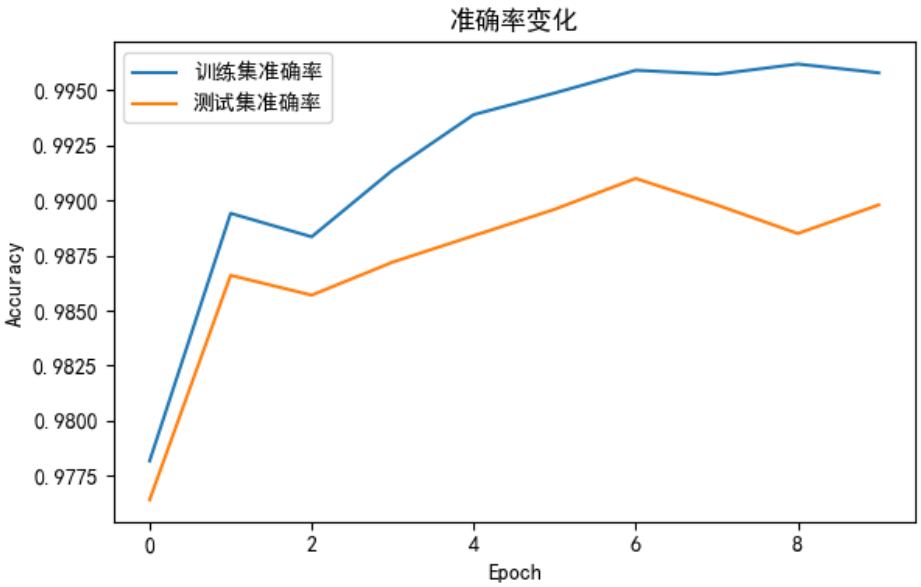
```
def set_config(args):
    config_cpu = ed({
        "num_classes": 30,
        "image_height": 224,
        "image_width": 224,
        "batch_size": 64,
        "epoch_size": 100,
        "warmup_epochs": 5,
        "lr_init": 0.001,
        "lr_end": 0.00001,
        "lr_max": 0.01,
        "momentum": 0.9,
        "weight_decay": 1e-4,
        "label_smooth": 0.1,
        "loss_scale": 1024,
        "save_checkpoint": True,
        "save_checkpoint_epochs": 5,
        "keep_checkpoint_max": 20,
        "save_checkpoint_path": "./",
        "platform": args.platform,
        "run_distribute": False,
        "activation": "Softmax",
        "freeze_layer": "backbone",
    })
```

4. 实验结果

⑴LeNet-5

在训练过程中，我统计了每个 epoch 的 loss 和 accurary，并画出相应曲线图。由于任务较为简单，且为了避免过拟合，我设置了早停机制，在第 10 个 epoch 处模型停下，具体训练结果如下：





可以看出，此时 loss 已经接近收敛，准确率也不再显著上升，处于较理想的训练程度。

此外，我还对不同数字模型预测的原始分数值和经 softmax 后的预测概率进行了可视化，核心概率计算公式如下：

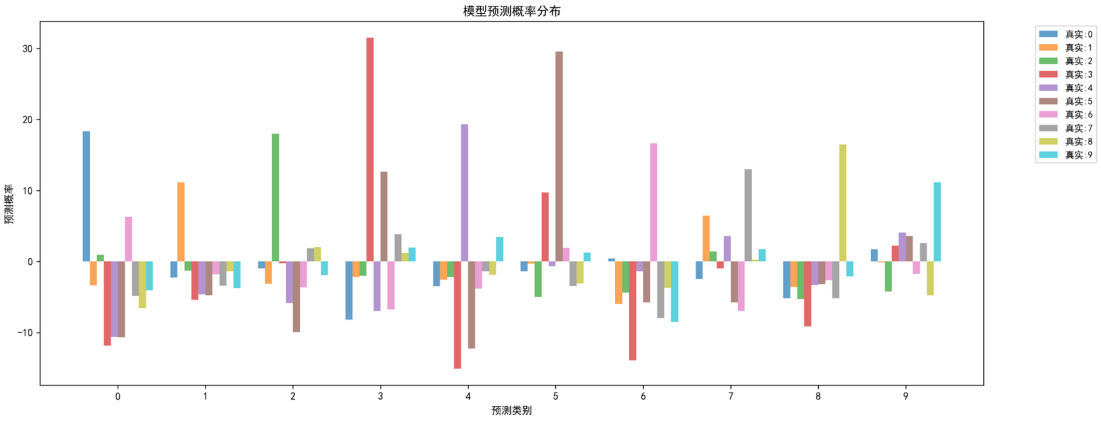
```
    # 获取测试集数据
    data_iter = test_dataset.create_dict_iterator()
    while len(found_digits) < 10:
        try:
            batch = next(data_iter)
            images, labels = batch['image'], batch['label']
            for i, label in enumerate(labels.asnumpy()):
                digit = int(label)
                if digit not in found_digits:
```

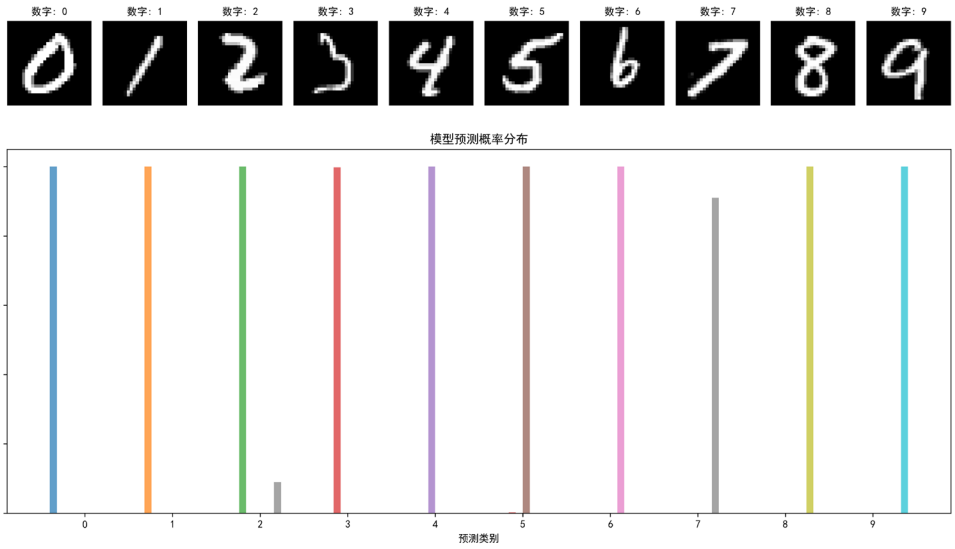
```
                    found_digits.add(digit)
                    image = images[i].asnumpy()
                    images_dict[digit] = image

                    prediction = model.predict(images[i:i+1])
                    predicted_digit = np.argmax(prediction.asnumpy())
                    probabilities = softmax(prediction.asnumpy()[0])
                    # probabilities = prediction.asnumpy()[0]
                    predictions_dict[digit] = (predicted_digit,
probabilities)

            if len(found_digits) == 10:
                break
        except StopIteration:
            data_iter = test_dataset.create_dict_iterator()
```

最终结果如下：

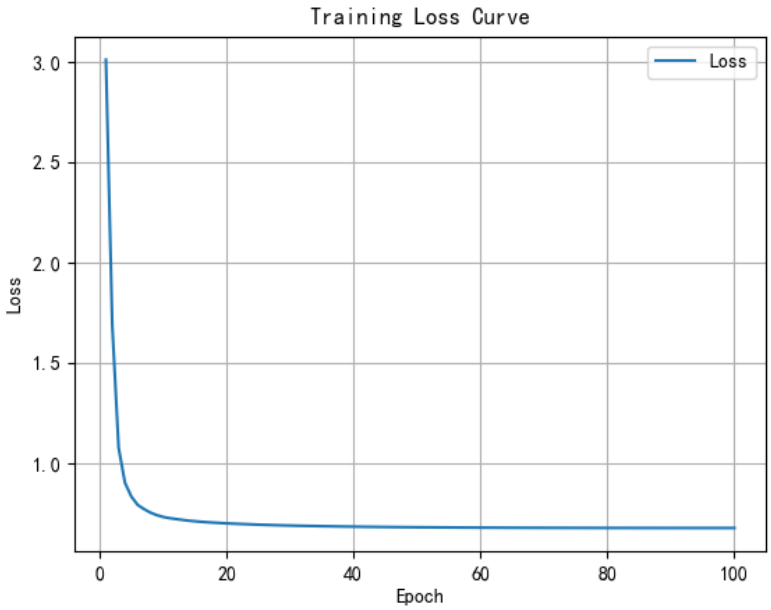


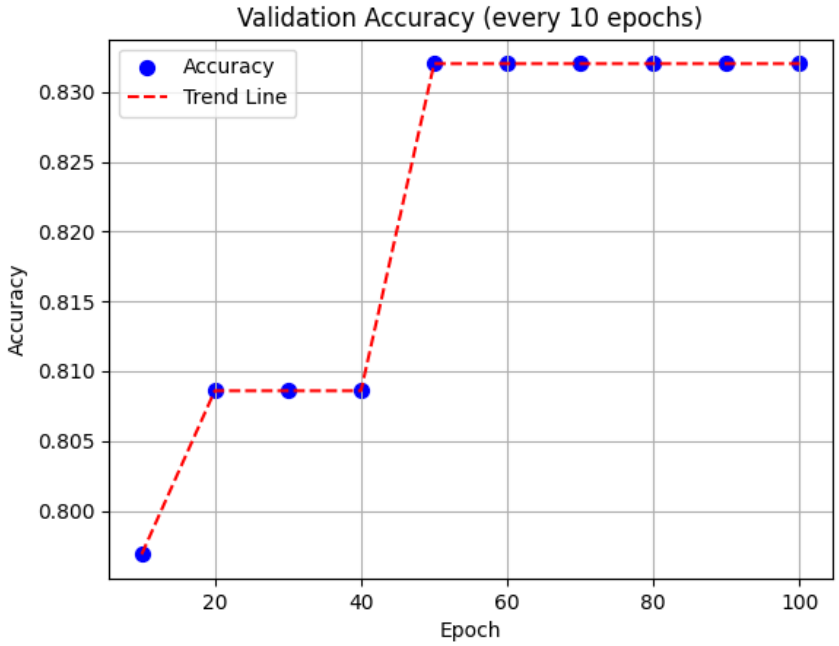


可以看出，在 softmax 归一化前，模型预测正确的概率很高，形状相近的数字在错误的预测类别中的预测概率越大（当然依然小于该预测类别正确对应数字的预测概率），在 softmax 归一化后，几乎所有预测类别正确对应数字的预测概率都为 1，说明模型预测效果很好。

⑵MobileNetV2

在训练过程中，我统计了每个 epoch 的 loss 和 accurary，并画出相应曲线图。在训练过程中，使用的是 data/train 和 data/test，均是未经处理的数据集，在最终检验时，使用的是 data_new/test，这个是经过噪声和旋转变换处理后的数据集，为了验证模型的泛化能力。





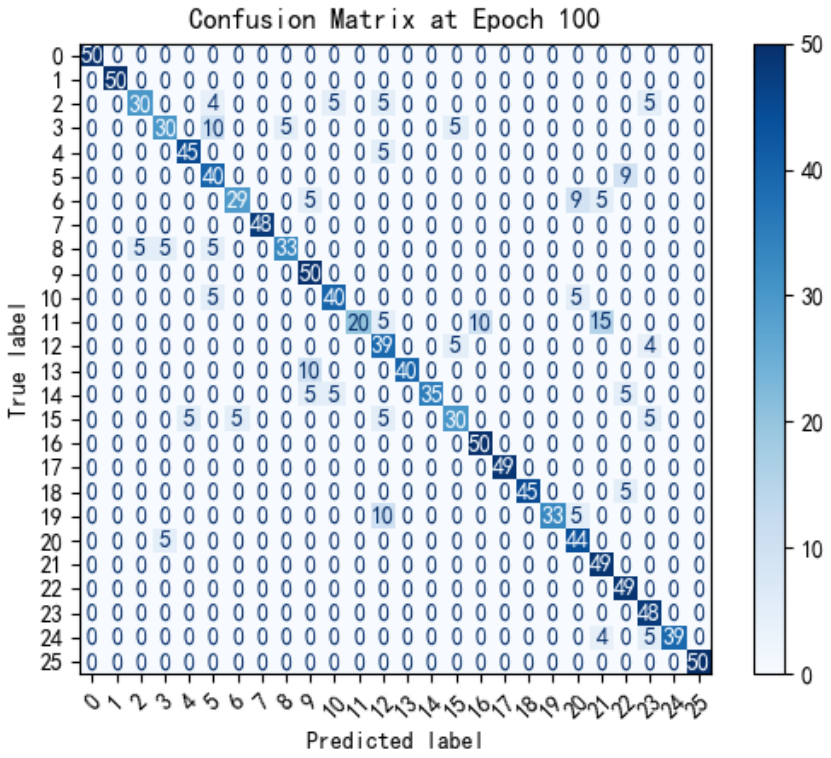

```
验证结果:
top_1_accuracy: 0.8234
top_5_accuracy: 0.9203
验证完成!
```

可以看出，loss 很早就已经接近收敛，准确率在第 50 个 epoch 后也不再显著上升，在 83%，处于较理想的训练程度。相应的，我还画了混淆矩阵和 t-SNE 可视化，t-SNE 方法的实现如下：

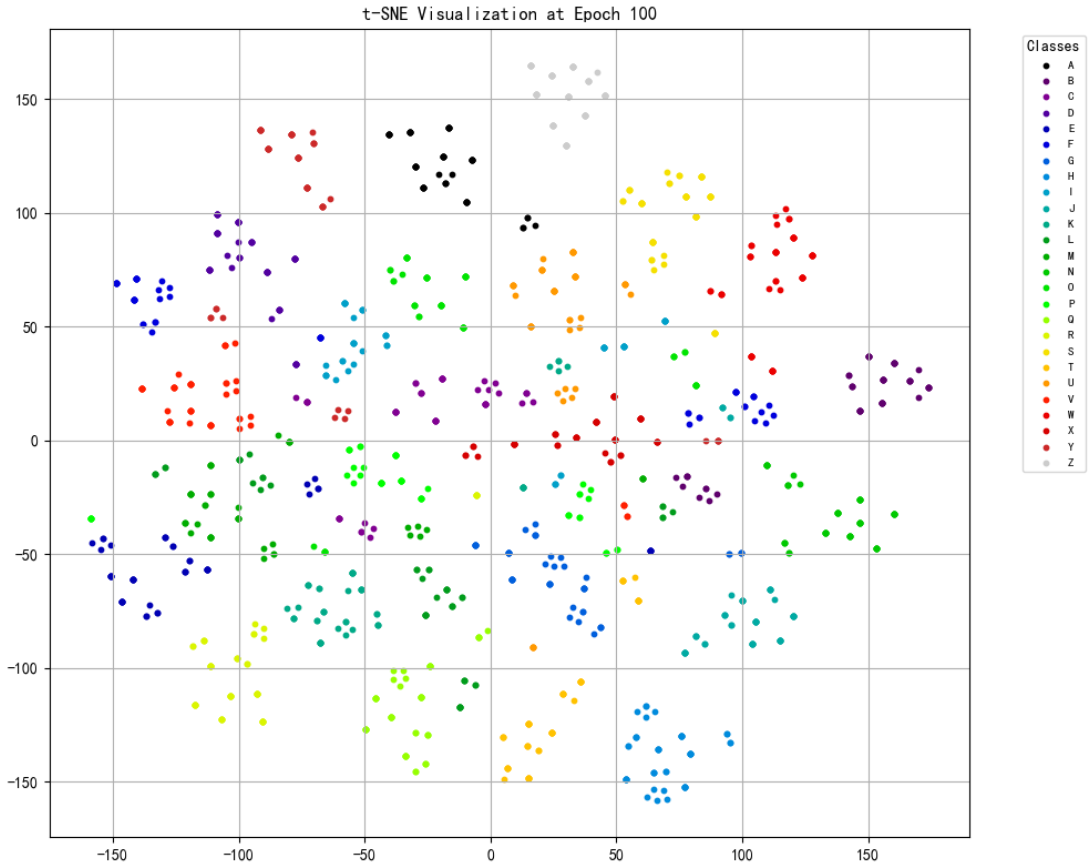
```
for data in eval_dataset.create_dict_iterator():
    image = data['image']
    label = data['label'].asnumpy()
    feature = net(image).asnumpy()  # 输出应为 logits
    pred = np.argmax(feature, axis=1)
    all_features.append(feature)
    all_labels.extend(label)
    all_preds.extend(pred)
features = np.concatenate(all_features, axis=0)
labels = np.array(all_labels)
preds = np.array(all_preds)
# ===== t-SNE visualization =====
tsne = TSNE(n_components=2, random_state=0)
tsne_result = tsne.fit_transform(features)
colors = plt.cm.get_cmap('nipy_spectral', 26)
class_names = list(string.ascii_uppercase)  # ['A', 'B', ..., 'Z']
plt.figure(figsize=(10, 8))
```

```
for i in range(26):
    idx = labels == i
    plt.scatter(tsne_result[idx, 0], tsne_result[idx, 1],
                c=[colors(i)], label=class_names[i], s=10)
plt.title(f"t-SNE Visualization at Epoch {epoch + 1}")
# 图像设置和保存
```

混淆矩阵如下：



t-SNE 可视化结果如下：



可以看出，混淆矩阵结果较为理想，t-SNE 可视化后发现每类之间区别较大，各自类聚合较好，容易区分，说明模型训练结果较为理想。

5. 总结与分析

(1)对模型的理解和评价

①LeNet-5

适合低分辨率、简单图像分类任务（如手写数字、字符识别），作为 CNN 入门模型，常用于教学和研究。引入卷积和池化操作，减少参数量，捕捉空间特征。使用局部感受区域和权重共享，奠定了现代 CNN 的基础。针对简单图像任务优化，计算需求较低。

A.优点

- 结构简单，易于实现和理解，适合初学者学习 CNN 原理。
- 在 MNIST 等小规模数据集上表现优秀，准确率高（约 99%）。
- 奠定了 CNN 核心思想（如卷积、池化），影响深远。

B.缺点

- 模型容量小，无法处理复杂图像任务（如 ImageNet）。
- tanh 激活函数易导致梯度消失，训练效率较低。

● 对现代高分辨率图像和多样化任务不适用。

②MobileNetV2

2018 年针对移动设备和嵌入式设备优化，追求低计算量和高效率设计的模型。它的倒残差结构通过先扩展再压缩特征通道，增强特征表达能力，适用于移动设备、嵌入式系统（如手机、物联网设备）的图像分类、目标检测；以及实时应用场景，如人脸识别、边缘计算。作为迁移学习的基础模型，广泛用于轻量级任务。

A.优点

● 高效性：参数量小（约 3.5M 参数），计算量低，适合移动端和边缘设备。
● 性能优异：在 ImageNet 上准确率较高（约 71.8% top-1），接近更重模型。
● 灵活性：支持宽度因子（Width Multiplier）和分辨率调整，适配不同资源需求。

B.缺点

● 相比大型模型（如 ResNet、EfficientNet），在复杂任务上的精度稍逊。
● 深度可分离卷积可能丢失部分空间信息，特征提取能力有限。
● 对极低资源环境的优化仍需权衡性能和精度。

⑵调试中遇到的问题

①刚开始微调时出现准确率极低的问题

在一开始微调第一版时，训练模型后最终准确率只有不到 50%，面对这个问题，我首先对模型进行了调参。一共修改了学习率预热(warmup_epochs)、学习率、batch_size 和权重衰减(weight_decay)等参数。

● 引入学习率预热 warmup_epochs: 0→5

缓慢提升学习率有助于模型在早期稳定收敛，避免梯度震荡。

● 更合理的学习率策略

从：初始为 0→最大 0.03→结束为 0.03（即恒定），改为：初始为 0.001→最大 0.01→结束为 0.00001（动态变化）。好处是初期更快进入学习状态，中期充分优化，后期更细致收敛，提升最终准确率。

● 增加权重衰减 weight_decay: 4e-5→1e-4

提升正则化强度，降低过拟合风险，有助于泛化性能，尤其是在小数据集上。该垃圾分类数据集训练集每类只有 100 张图像，属于小数据集。

● 减少 batch_size: 150→64

更多迭代次数，梯度更丰富，有时对泛化有帮助。

最终结果中准确率确实有所改进，但又遇到了新的问题，详见下文叙述。

②过拟合问题及其解决

在继续训练时，我发现模型输出准确率达到 100%，考虑到是因为验证集 test 和训练集 train 分布相同，怀疑是模型过拟合导致的结果。

```
开始验证...
验证结果：
top_1_accuracy: 1.0000
top_5_accuracy: 1.0000
验证完成！
```

对此，我又补充了实验，我对验证集进行了添加噪声的处理，再次验证后发现准确率又回到了 47%左右。

```
开始验证...
验证结果：
top_1_accuracy: 0.4766
top_5_accuracy: 0.7539
验证完成！
```

在这时经过分析我认为模型确实是发生了过拟合，为了解决这个问题，提升模型的泛化能力和在现实中的实际应用效果，我尝试进行处理，我着眼于数据增强部分，在原有数据增强（主要集中在图像随机解码并裁剪图像，亮度变化和旋转等）的基础上，加入了高斯噪声的处理，以提升模型对图像噪声干扰的鲁棒性，尤其在真实环境下的泛化能力。

具体实现如下：

```
def add_gaussian_noise(image):
    """
    Add Gaussian noise to an image.
    Args:
        image (numpy.ndarray): Input image in HWC format (uint8).
    Returns:
        numpy.ndarray: Image with Gaussian noise (uint8).
    """
    std = 5.0  # 噪声标准差，可调整
    noise = np.random.normal(0, std, image.shape).astype(np.float32)
    noisy_image = image.astype(np.float32) + noise
    return np.clip(noisy_image, 0, 255).astype(np.uint8)
# 使用 Compose 包装高斯噪声函数
gaussian_noise_op = P.Compose([add_gaussian_noise])
# …… 省略中间内容 ……
    if do_train:
        ds = ds.map(operations=[gaussian_noise_op],
input_columns="image", num_parallel_workers=8)
```

然后重复实验，发现模型在原有验证集和添加噪声的处理的新数据集上的预测正确率都达到 80%以上，因此，我便以这个模型作为最终结果。

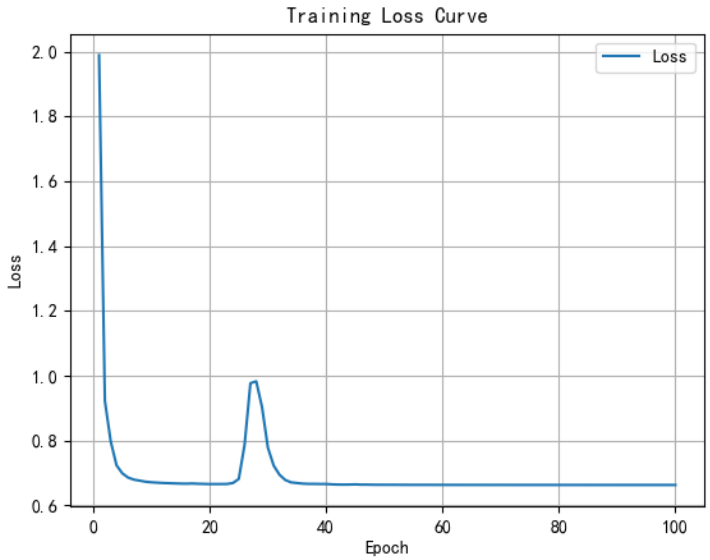
```
验证结果:
top_1_accuracy: 0.8234
top_5_accuracy: 0.9203
验证完成!
```

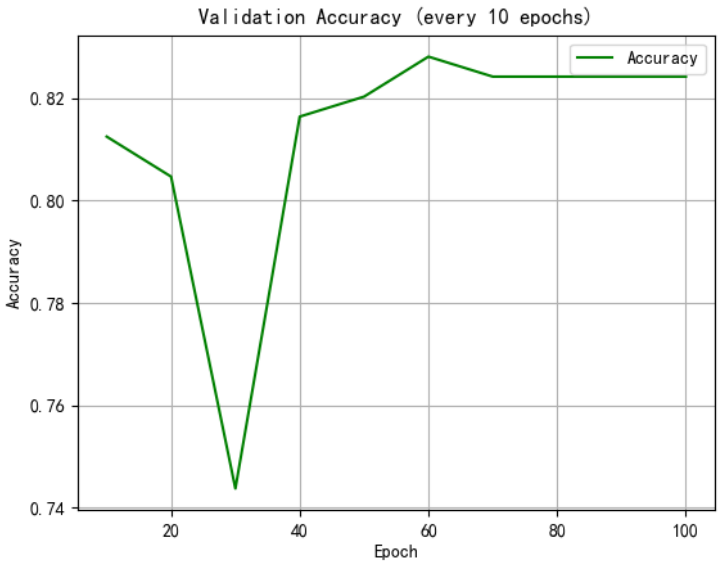
③Momentum 换用 Adam 时出现 loss 和 accuracy 波动问题

经过资料查询，我得知 Adam 优化器相较于 Momentum 优化器适合小批量训练、复杂网络结构，对学习率调节不敏感，训练更“自动”，收敛速度通常更快。如果只微调 head_net，Adam 可能更方便并且效率高。因此我换用 Adam 进行了实验，结果发现，使用后 loss 确实收敛更快，但在第 20 个 epoch 附近 loss 突然增大，但好在后来又重新降低，总体来说最终模型预测的准确率与原来类似。

```
opt = nn.Adam(
    params=filter(lambda x: x.requires_grad, head_net.get_parameters()),
    learning_rate=lr,
    weight_decay=config.weight_decay
)
```

下面是换用 Adam 后模型训练的 loss 曲线和 accuracy 曲线：





分析原因，可能是 Adam 的强自适应性虽然使其收敛快，但在训练初期若梯度信息不稳定、batch 差异大或参数太少，就容易导致 accuracy 波动，这确实符合该实验的情况。所以虽然 Adam 会为每个参数自动调整学习率，前期收敛快，最终效果也类似，但为了模型的稳定性，最终我依然选择使用 Momentum 优化器。

⑶实验收获

通过本次实验，我深入理解了 LeNet-5 和 MobileNetV2 的架构设计及其背后的理念，以及模型微调的方法。LeNet-5 作为早期 CNN 的代表，通过卷积层、池化层和全连接层的组合，展示了局部感受区域和权重共享如何减少参数量并捕捉空间特征。实验中，我注意到其使用 tanh 激活函数容易导致梯度消失，限制了模型深度。相比之下，MobileNetV2 引入了深度可分离卷积，将标准卷积分解为深度卷积和点卷积，大幅降低计算量，同时通过倒残差结构和线性瓶颈增强特征表达能力。实验让我认识到，LeNet-5 适合简单任务，而 MobileNetV2 的设计更适应现代轻量级需求，理解这些差异为后续模型选择提供了理论基础。

实验中，我掌握了使用 MindSpore 数据管道（如 MnistDataset）进行数据加载和预处理的技巧。通过实现图像归一化（如 Rescale 将像素值缩放到 0-1）和格式转换（如 HWC2CHW），我体会到预处理对模型训练稳定性和性能的关键作用。例如，LeNet-5 处理灰度图像（1 通道），而 MobileNetV2 需要 RGB 图像（3 通道），这要求我根据模型调整输入格式。实验还让我意识到，合理的数据预处理能显著提升模型收敛速度和准确率，特别是在处理不同数据集时，灵活调整预处理流程至关重要。

实验中，我实现了 LeNet-5 和 MobileNetV2 的训练流程，并通过 MobileNetV2 微调进一步理解了优化策略的差异。LeNet-5 从头训练需要较多轮次，而 MobileNetV2 微调利用预训练权重显著加快了收敛速度。使用 SoftmaxCrossEntropyWithLogits 损失函数和 Momentum 优化器，我发现微调时冻结 MobileNetV2Backbone 部分，仅更新 MobileNetV2Head 能有效保留预训练特征，同时适配新任务。实验还让我体会到微调中超参数的选择至关重要，避免破

坏预训练权重，增强了我的超参数调优能力。

微调 MobileNetV2 的实验让我掌握了迁移学习的核心技巧，特别是在资源受限场景下的高效应用。我学会了如何利用预训练模型（通常基于 ImageNet）初始化网络，通过冻结 MobileNetV2Backbone 部分，仅更新 MobileNetV2Head。实验中，我发现微调需要平衡预训练权重和新任务数据的学习，例如通过差异化学习率（早期层较低，后期层较高）提升效果。此外，我体会到小规模数据集容易导致过拟合，需结合数据增强和正则化（如 Dropout）来改善泛化能力。这次微调实验让我深刻理解了迁移学习在实际项目中的价值，尤其是在数据量有限或计算资源受限时。

总体来说，通过 LeNet-5 和 MobileNetV2 微调的实验，我从模型架构、数据处理、训练优化、应用场景、框架使用、问题解决和微调经验七个方面获得了全面的收获。这些经验不仅加深了我对 CNN 设计和迁移学习原理的理解，还提升了使用 MindSpore 进行模型开发、微调和调试的实践能力，为未来处理复杂任务奠定了坚实基础。

6. 参考文献及资料

[1] Hung-yi Lee, Machine Learning (2021 Spring), https://speech.ee.ntu.edu.tw/~hylee/ml/2021-spring.php.

[2] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[3] Sandler M, Howard A, Zhu M, et al. Mobilenetv2: Inverted Residuals and Linear Bottlenecks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 4510-4520.

[4] Huawei Cloud Community, MindSpore1.0: MobileNetV2 网络实现微调 https://bbs.huaweicloud.com/forum/thread-83710-1-1.html.