# 数据集验证日志

## 数据集要求

为保证在后续研究中，基于本数据集得到的结果和结论具有可靠性和说服力，需要先验证数据集质量。本研究对数据集的质量要求如下：

1. 数据完整 准确 唯一，数据格式/采集工况/预处理方式一致；
2. 标签能正确反应类别，没有标注错误或语义混乱；
3. 标签与数据本身分离，没有标签泄露或设备伪影；
4. 分布合理性与样本代表性。

本研究使用到的数据集为200\*200 pixel png彩色(三通道)图片，其中数据标注、数据完整性和采集条件一致性等已经在仿真中的数据处理环节完成，本实验主要验证数据集的稳定性与特征分布。在前文《本次仿真日志》的最后已经将数据集信息存入了MySQL表格，因此本实验直接从数据库中获取数据路径及标签。为确保没有因操作失误等原因引入的重复图片，通过查找哈希值来检查。该程序保存为data\_Cleaning.py，在Anaconda Powershell命令行里输入cd d:\ProgramData\Python\Classifier\Validation转到脚本所在文件夹，输入命令python data\_Cleaning.py "..\Dataset" --mode phash，即可在Classifier的同级文件夹Dataset中查找重复图片。其结果保存为duplicate\_report.JSON。

结果发现重复图片多达三百多组，而且全部来自匝间短路类别同一故障程度的两个子类。在确认仿真结果确实来自不同的FE模型、不同的Simulink仿真结束时回调保存的变量后，观察原始数据，发现这些导出重复图片的工况的电流信号确实非常接近。在不同类别存在重复样本的情况下，模型不可能把它们完全区分开，尽管在这些相近类别中，个别工况存在一定区分度，但是考虑到类别平衡性，决定只保留电源端的样本。然而如此一来就损失了一个类别，因此决定补采数据。在进行后续实验以前，收集了匝间短路-B相数据(原来为C相)，这部分见《数据补采日志》。

## 简单统计指标

该数据集来自三相电流信号连续小波变换后的时频能量图，几乎无法通过目测来判断类别，因此需要借助统计特征来初步观察其规律。统计规律可以为后续分析学习效果、调整模型结构提供参考依据，但如果规律过于明显，而且不是来自类别本身的特征，则说明数据标签可能泄露，需要重新考虑预处理的方式。

### 类别数量平衡性

类别平衡性包括数量平衡性和分布平衡性。数量平衡性是衡量分类任务中不同类别的训练样本数量差异的指标。如果类别数量不平衡，模型就会倾向于偏向多数类，此时需要根据类别比例初始化输出层偏置或调整采样策略。

该数据集的类别及其样本数如表1-1所示，由于不同故障类型在相同程度(%)时反应在电流信号中的变异程度不同，后续对每种类别的故障程度统一采用1~4的级别编号表示。该数据集每种类别每种级别的样本数量都相同，大类之间的比例为1:8:8:8:1。

表1-1 样本类别平衡性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 大类 | 样本数 | 小类 | 样本数 | 程度% | 样本数 |
| 健康 | 180 | -- | 180 | 0 | 180 |
| 偏心 | 1440 | 静偏心 | 720 | 20 | 180 |
| 40 | 180 |
| 60 | 180 |
| 80 | 180 |
| 动偏心 | 720 | 20 | 180 |
| 40 | 180 |
| 60 | 180 |
| 80 | 180 |
| 退磁 | 1440 | 均匀退磁 | 720 | 10 | 180 |
| 20 | 180 |
| 30 | 180 |
| 40 | 180 |
| 局部退磁 | 720 | 10 | 180 |
| 20 | 180 |
| 30 | 180 |
| 40 | 180 |
| 匝间短路 | 1440 | C相短路 | 720 | 1.28 | 180 |
| 2.56 | 180 |
| 3.85 | 180 |
| 5.13 | 180 |
| B相短路 | 720 | 1.28 | 180 |
| 2.56 | 180 |
| 3.85 | 180 |
| 5.13 | 180 |
| 轴承 | 180 | -- | 180 | -- | 180 |

本实验取大类&小类&故障程度(即表1-1中的一行)为最小粒度，如果需要整体统计大类/小类，再根据样本比例来调整设置。

### 统计特征分布

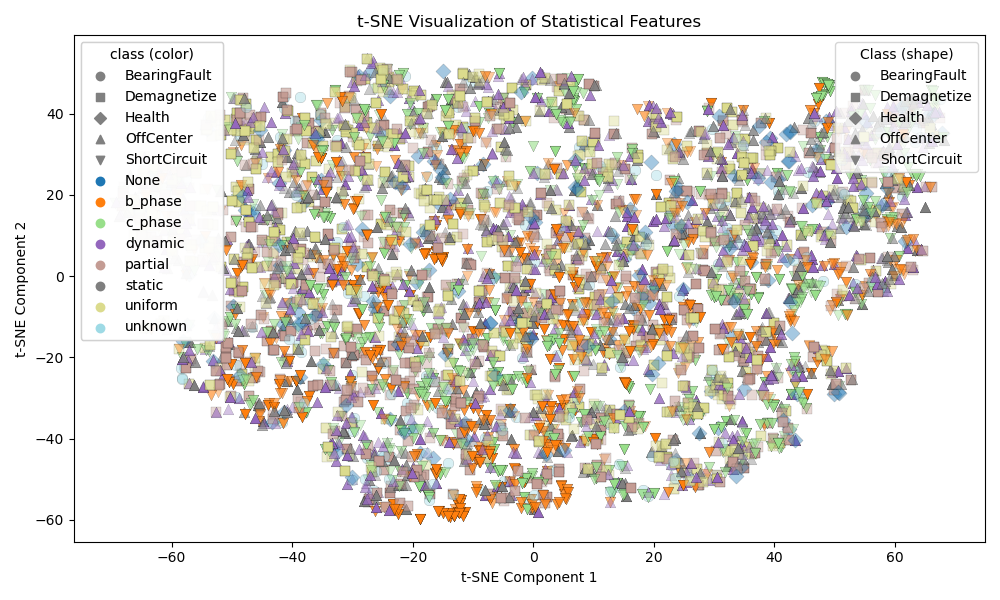
直接反应在PNG图片中的信息包括颜色特征和纹理特征。本实验取均值 方差 偏度 峰度这4种颜色特征，LBP纹理直方图和灰度共生矩阵(GLCM)这2种纹理特征，拼接以后采用t-SNE进行非线性降维可视化，得到的结果如图2-1所示。

图2-1 统计特征可视化

图中形状表示大类，颜色表示子类，颜色深度表示故障程度。观察图2-1可得，全局统计特征呈显著同质性，只是不同类别的特征集中分布的区域略有差异，比如b相匝间短路主要集中在第二个维度的负半轴，而均匀退磁则主要集中在第一个维度的正半轴。这种现象是合理的，目测结果也显示所有类别的图片非常相似，很难通过直观特征进行区分。这是由于图片的纹理主要与信号频率有关，时频能量图中频率越高的部分纹理越细，较少受到故障类别影响；而且由于每组数据的三个通道采用了相同的归一化，因此不同类别的图片也不存在与类别无关的颜色偏置。

为了进一步确认这些统计特征是否存在规律，对其作箱线图，如图2-2所示，图中横坐标为类别，一簇为一个子类，一条为一个最小颗粒，纵坐标表示特征空间中的欧氏距离，即该箱线图表示每类样本统计特征的空间分离程度。

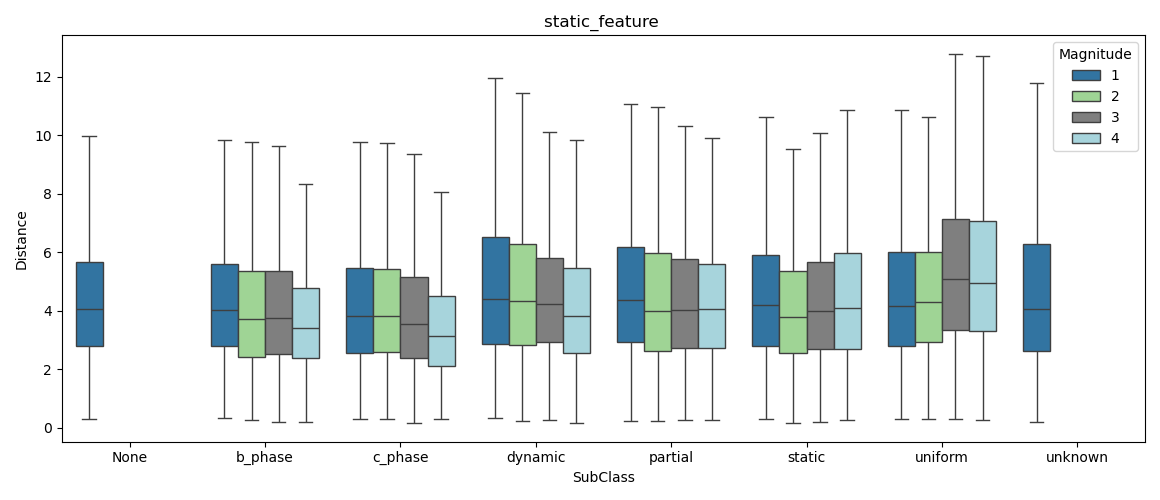


图2-2 统计特征箱线图

在图2-2中，每条箱线的高度和位置非常接近，只有动偏心、局部退磁、匝间短路的两个子类存在故障程度越大，统计特征越集中的规律。该程序保存为Visualization.py。对这些特征用线性分类器进行分类，得到其五折准确率为28.25% ± 1.99%，虽然显著高于随机分类，但是仍然较低，因此可以认为该数据集没有因为简单规律导致标签泄露。

基于简单统计指标的线性分类结果表明该数据集内的样本类别很难通过统计特征来区分，推测这可能为传统的基于电流信号连续小波变换的电机故障诊断方法准确率受限(比如受到工况限制)的主要原因。为了在此基础上进一步提高准确率，可能需要引入CNN提取空间纹理等深度特征。

## 数据集质量(模型收敛测试)

已知可靠的模型在数据集上的表现可以一定程度上反应该数据集质量的好坏。当数据集质量高时，无论怎样抽取训练集，模型总是能够学到有用特征，因此其训练效果是稳定可复现的，而且通常会随着数据增多、模型复杂度增加而提升；对于类内不稳定的数据集，模型的效果依赖抽到的样本，因此性能方差会很大；更严重地，如果数据集本身就不具备可区分/可泛化的特征，模型只能记忆样本，不仅测试准确率低、损失不收敛，而且通常对数据量和模型结构不敏感。本研究将数据集质量分为稳定性、平衡性、标注有效性三个维度，其具体指标分别为不同训练集抽样时的性能方差大小、多次随机划分时的混淆矩阵对角线深浅是否一致、学习曲线是否随着打乱标签比例的提高而下降，直至随机基线。评估数据集质量可为调整采样策略、清理不可分样本或标注无效类提供参考依据。

### 数据量敏感性测试

数据量敏感性测试作为后续复杂模型收敛性分析的前置环节，可用来快速判断一个数据集是否具备被常见模型学习的可能性。正常情况下模型准确率会随着样本数量的增加而提升，直到数据量增加到一定程度，新增数据所能提供的新模式趋于饱和时到达拐点。若任何训练数据的增加对模型性能提升均不显著，甚至使其下降，则表明该训练任务的可行性存疑——如果当前模型完全无法提取到任何有意义的特征，或者数据本身缺乏可学习的关联性，那么在此基础上进行其它模型收敛测试都是没有意义的。与之相对地，如果模型只需要少量数据就能达到很高的准确率，也未必说明数据质量高，有可能是特征单一/分布太窄导致学习曲线迅速饱和，或者数据冗余度过高，少量数据足以覆盖主要模式。特别地，如果直到训练比例趋近100%仍未出现拐点，则通常说明数据量不足、模型仍数据饥渴，理论上性能仍有提升空间；极特殊情况下会遇到非有限样本可学的任务，即任务本身的固有复杂度→∞，远超任何数据集所能承载的信息量。

为测数据量敏感性，随机抽取不同比例作为训练集，其余数据作为测试集，用一个简单的CNN网络进行分类，并记录其收敛曲线和准确率。本研究采用的CNN模型前向传播过程及每层Tensor形状如表3-1所示，直接输入200\*200 pixel \* 3通道原图，输出26个类别。该程序保存为CNN\_Classifier.py。

表3-1 三层CNN前向传播

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 类型 | 输入 | ReLU | Dropout | SE | 池化 | 输出 |
| 卷积(3,1) | 200\*200\*3 | \*1 | -- | 16 | /(2\*2) | 100\*100\*16 |
| 卷积(3,1) | 100\*100\*16 | \*1 | -- | 32 | /(2\*2) | 50\*50\*32 |
| 卷积(3,1) | 50\*50\*32 | \*1 | -- | 64 | /(2\*2) | 25\*25\*64 |
| 展平+线性 | 25\*25\*64 | \*1 | 0.5 | -- | -- | 128 |
| 输出层 | 128 | -- | -- | -- | -- | 26 |

为了程序的可移植性和可维护性，采用Trainer类统一管理训练逻辑，并将训练比例、批次大小、学习率、训练轮数、随机种子等超参数保存到JSON配置文件。其伪算法如算法1所示。该程序保存为trainer.py。

|  |  |
| --- | --- |
| 算法1: 训练并保存结果 | |
| 输入: 配置文件列表（JSON file） | |
| 输出: 最佳模型、学习曲线、混淆矩阵 | |
| 1: | 初始化：模型与日志路径 |
| 2: | **for**每个配置文件**do** |
| 3: | 加载模型与数据集 |
| 4: | **for** 训练循环 **do** |
| 5: | **if** 当前为训练阶段**then** |
| 6: | 使用训练集训练 CNN 模型，计算损失与准确率 |
| 7: | **else** |
| 8: | 使用验证集评估模型性能，计算损失与准确率 |
| 9: | **end if** |
| 10: | 记录当前epoch的loss、acc、f1到CSV文件 |
| 11: | **if** val\_acc > best\_acc **then** |
| 12: | 更新best\_acc；保存当前模型参数；保存混淆矩阵 |
| 13: | **end if** |
| 14: | **end for** |
| 15: | **end for** |

本测试共采用了5组配置，即train\_ratio分别为10%，25%，50%，75%，90%，其它参数均相同，此时学习率1e-3，权重衰减率5e-3，批次大小64；为了消除模型初始化的影响，固定随机种子42；在JSON中设置"name": " data\_ratio"，训练时用命令行转到前面所说的脚本路径，输入python trainer.py --cfg dataratio1.JSON dataratio2.JSON dataratio3.JSON dataratio4.JSON dataratio5.JSON运行程序。运行结束后即可看到result/data\_ratio路径下的CSV文件。作图程序保存为plot\_result.py，从命令行运行python plot\_result.py --exp data\_ratio --mode acc即可作准确率曲线。得到每种训练比例下的学习曲线如图2-3所示，由图可知模型最终的准确率随训练比例而提高，表明该数据集类别内的样本可能存在一定的代表性，简单CNN模型能学习到一点可泛化的特征。然而整体准确率较低，甚至没有超过统计特征的线性分类，需要继续进行数据诊断。

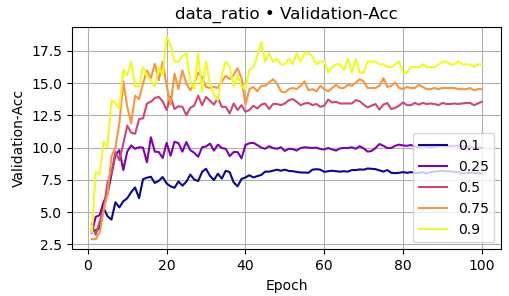


图2-3 不同训练比例下的准确率曲线

修改trainer.py，令训练epoch和测试epoch从同一数据迭代器中取数据，发现该模型的准确率能够达到100%，即模型有完全记住样本的能力，然而对于同一类别内的未见数据，泛化效果不好，即使训练比例高达90%，仍不能有效地识别其余10%的样本。为了得知该结果是否为偶然现象，需采用不同的训练/测试数据划分进行实验，如果模型表现出了类似的数据敏感性，就说明类内数据分布比较均衡。

### 数据稳定性

数据稳定性（Data Stability）是衡量数据集内在质量的指标之一。稳定的数据集无论怎样划分训练/测试集，模型表现都相似；相反，如果模型学习效果很大程度上受到划分方式的影响，则暗示数据集可能类内信息不均匀、样本代表性不一致。尽管实际的数据集通常或多或少都包含难以识别或分布边缘的样本，但如果不同划分方式的性能方差过大，甚至超过了训练比例的影响，则意味着该数据集很可能存在严重的结构失衡，需要重新对齐（Re-alignment）才能确保它训练出来的模型具有正常的泛化能力。

通过多次随机划分可以评估数据集的稳定性。设置不同随机种子可将数据集打乱为不同顺序，尽管由于Dropout扰动，即使固定了随机种子，每次训练的结果仍然会略有差异，但至少用作训练和测试的数据划分是固定的，因此不同随机种子下的准确率结果有一定的说服力。仍然采用上述三层CNN分类网络，在JSON中设置"name": "seed\_ratio"，取随机种子Seed =15，22，29，36，43，分别取train\_ratio为10%，25%，50%，75%，90%，运行python trainer.py --cfg seed15\_ratio10.JSON seed15\_ratio25.JSON seed15\_ratio50.JSON seed15\_ratio75.JSON seed15\_ratio90.JSON seed22\_ratio10.JSON seed22\_ratio25.JSON seed22\_ratio50.JSON seed22\_ratio75.JSON seed22\_ratio90.JSON seed29\_ratio10.JSON seed29\_ratio25.JSON seed29\_ratio50.JSON seed29\_ratio75.JSON seed29\_ratio90.JSON seed36\_ratio10.JSON seed36\_ratio25.JSON seed36\_ratio50.JSON seed36\_ratio75.JSON seed36\_ratio90.JSON seed43\_ratio10.JSON seed43\_ratio25.JSON seed43\_ratio50.JSON seed43\_ratio75.JSON seed43\_ratio90.JSON进行训练。训练结束后，运行python plot\_result.py --exp seed\_ratio --mode box即可作每种训练比例下的准确率箱线图，其结果如图2-4所示。

图表, 箱线图

AI 生成的内容可能不正确。

图2-4 不同随机种子时的准确率箱线图

箱线图显示该数据集稳定性好，当采用不同的训练/测试集划分方式时，除了随机种子为36，训练比例为90%时出现了低于14%的特殊情况外，整体仍表现出准确率随数据量增加而上升的规律，可见在数据收集和预处理时不同类别之间已经进行了有效的对齐。然而整体准确率仍非常的低，主要集中在7.5%到20%之间，当训练比例从10%增加到90%，准确率也只提升了10%左右。为了进一步追查准确率低的原因，继续研究其分布平衡性。

### 类别平衡性

与数据稳定性主要衡量类内信息分布的一致性不同，数据平衡性（Data Balance）是衡量类别之间的关系的一个质量指标。本研究将数据平衡性细分为分布平衡性（Distribution Balance）和难度平衡性（Difficulty Balance）。分布平衡性指的是不同类别在特征空间中的分布是否均匀，例如类别边界是否清晰、原型距离是否接近。距离较远、边界清晰的类别易于区分，而距离接近、边界复杂甚至存在重叠的类别可分性差，更有甚者当数据集中存在非完全独立的类别时，不仅会影响模型的学习效果，例如过度学习类别之间的特定依赖关系，而且还有可能直接违反某些算法的基础假设。难度平衡性则反映了数据集中不同类别学习难度的差异。即使是类别数量均匀的数据集，如果不同类别的难度相差过大，模型也会倾向那些容易的类别。因为优化器的目标是最小化总损失，当某些类别可以快速贡献损失降低时，模型就会优先学习它们的特征，此时就需要借助数据增广/过采样/动态权重损失来平衡学习过程，引导模型关注困难类别。

数据平衡性可以通过混淆矩阵来观察，在JSON中设置"name": " confusion\_matrix "，固定训练比例为0.8，再取上述随机种子，运行python trainer.py --cfg confusion\_matrix1.JSON confusion\_matrix2.JSON confusion\_matrix3.JSON confusion\_matrix4.JSON confusion\_matrix5.JSON进行训练。训练完成后，运行python plot\_result.py --exp confusion\_matrix --mode cm作混淆矩阵，其结果如图2-5所示。

图2-5显示整个混淆矩阵的对角线深浅不一，表明该数据集的各个类别十分不平衡。匝间短路故障非常容易辨别，贡献了大部分准确率；其次是故障程度最高的两个退磁，对角线比其它位置稍微高出一点；其余类别的情况不太乐观，模型几乎不认，整个偏心大类近似随机猜测。

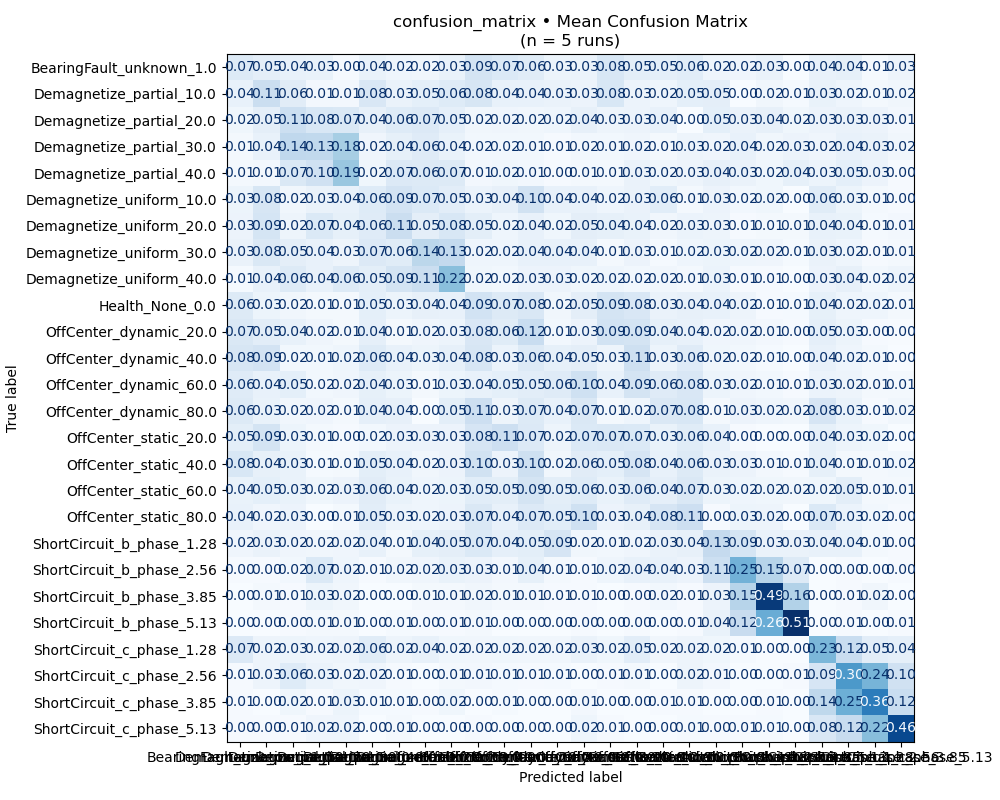
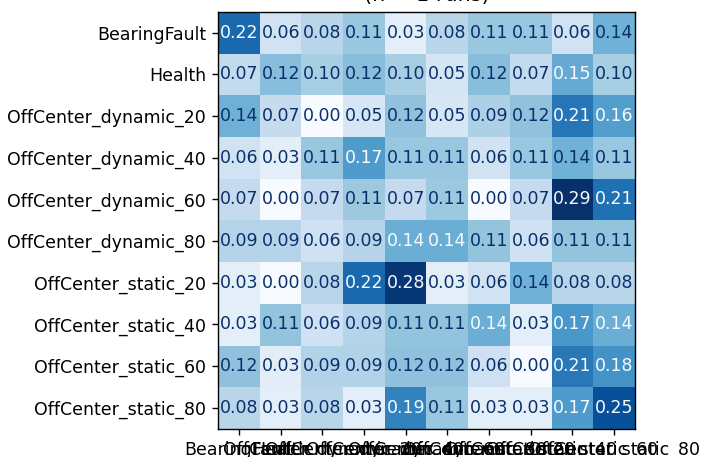


图2-5 5次训练平均混淆矩阵

表格

AI 生成的内容可能不正确。首先在混淆矩阵中没有发现模型输出倾向于准确率高的类别，即没有表现出优化器被某个类别劫持的迹象，故推测不同类别间难度相近，只是分布失衡。为了确认健康、偏心、退磁和轴承故障不平衡的来源，单独观察它们的混淆矩阵，如图2-5。

(a) 偏心大类混淆矩阵 (b) 退磁大类混淆矩阵

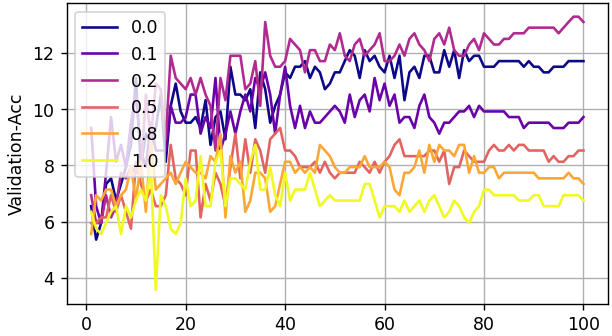
图2-5 5次训练平均混淆矩阵

多次运行得到的结果也都相似，即偏心故障内部的分类不收敛。事实上整个偏心大类的可分性相当的差，不是难度大的问题，两两之间连二分类都无法泛化，准确率稳定在50%附近。退磁大类表现出了轻微的收敛迹象，混淆矩阵的对角线也略微明显，然而上限仅有20%，可见它在整体分类中表现不佳的原因并非难度大、模型被短路大类劫持，而是其本身的可分性不高。之所以两次实验都包括了原型——健康类别，是因为任何类别都必须能够与之区分，然后它才有出现在这个数据集里的意义——否则这种故障无法被「诊断」。考虑到数据不平衡且无法通过采样或训练方式来弥补，决定根据混淆矩阵的结果和这些类别与原型之间的可分性，将数据集分成两组，分别研究其标注有效性。

### 标签有效性

通过随机标签测试可以估计数据集标签的有效性。当模型的抗干扰能力(或鲁棒性)一定时，比较打乱前后的准确率能够评估标注的质量。理想情况下，模型在正确标注的数据集上会达到较高的准确率，因为它真能学习到可泛化的模式，而在标签被随机打乱后，模型无法建立有效的映射关系，因此准确率会随打乱比例而明显下降，直至随机猜测水平。然而如果原始标注本身存在缺陷甚至无效，那么模型在打乱标签后的表现不仅不会显著低于正确标注，在特殊情况下反而能表现出一定的学习能力，这往往是因为样本特征与这些错误的标签之间依然存在某种非预期的统计相关性，使得模型学习到了某种虚假关联所致。

图表, 折线图, 散点图

AI 生成的内容可能不正确。现将数据集分成两组子集，其中数据子集1包括原型、整个匝间短路大类和两种退磁故障的30%和40%，共13类；子集2包括原型、轴承故障、整个偏心大类和两种退磁故障的10%和20%，共14类；分别将标签随机打乱，用前面的三层CNN模型进行拟合。在JSON中设置"name": "sanity\_check"，将JSON配置中的random\_label改为true，固定训练比例为0.8，随机种子为42，设置shuffle\_ratio分别为0，0.1，0.2，0.5，0.8，1，设置正确的类别数；在trainer.py中修改保存CSV文件的代码，将shuffle\_ratio添加到文件名中；运行python trainer.py --cfg sanity\_check0.JSON sanity\_check1.JSON sanity\_check2.JSON sanity\_check3.JSON sanity\_check4.JSON sanity\_check5.JSON进行训练；训练结束后在作图程序中修改文件名匹配，运行python plot\_result.py --exp sanity\_check --mode acc作准确率曲线，两组的结果如图2-8所示。

(a) 数据子集1 (b) 数据子集2

图2-8 不同随机标签比例下的学习曲线

由于前面的测试结果表明该数据集类内稳定性较好，因此可以认为该随机种子时的结果具有代表性。图2-8显示随着打乱比例的不同，数据子集1趋势模糊，甚至表现出了类似样本大量标错时的现象，不仅正常标注时准确率只有12%，略高于随机猜测，而且当置乱20%时准确率不降反升，说明此时模型在捕捉样本与错误标签之间的伪关联，由此可知这部分数据的标注质量相当的低，混在数据集里属于干扰甚至误导正常预测的水平。数据子集2表现整体符合预期，在去掉了12个低质量类别后准确率相较之前提升了近15%，其准确率随打乱比例升高而稳定下降，全部置乱时准确率逼近随机基线1/13，可见这部分的标注是基本有效的。值得注意的是，数据子集2虽然分类准确率不高，但是它在测试集上的loss几乎不降，这说明导致其分类效率不高的核心原因很可能并非类别之间固有的高相似性或复杂边界，而是模型在模式学习上仍未充分收敛(意思是模型根本就没“学会”)，故暂不考虑继续分离其中的易混淆样本或合并类别、重新对齐，而是继续验证该分类任务的难点所在，以期有针对性地解决模型不能充分收敛的问题(意思是得先让模型“学起来”)。

## 数据集难度

本研究将数据集难度（Dataset Difficulty）与样本代表性（Sample Representativeness）进行了区分。其中数据集难度旨在量化机器学习/深度学习任务中模型学会该数据集所需的结构复杂度和计算资源消耗，通常与数据集规模、样本数量以及特征内在复杂度密切相关。理想情况下，模型的学习效果会随着其参数量的增加和结构复杂度的提升而逐步改善，直到触及一个由数据内在特性所决定的上限，该上限通常由样本代表性不足、训练数据未能充分捕捉目标任务的真实分布所造成，此时模型仍有可能通过迁移学习等有望借助外部知识或对比/度量学习等具有更强大的特征表示能力的方法来获得突破。因此评估数据集难度及其难度来源可为模型选择、算法比较和参数调优提供基本参考，同时也有助于定位问题，进一步验证每个类别的样本代表性。本环节将从Acc、校准、固有维度、可解释性、参数量这5个维度来评价模型，在达到一定准确率的前提下优先选择能够用更小参数规模来压缩表征信息的架构。考虑到只有数据集质量可靠时，评估其难度才有意义，为避免数据子集1干扰模型正常收敛，本环节仅测试数据子集2的难度，以便选出合适的模型和训练超参数。

### 多模型测试

模型的分类能力依赖于其所能从数据中提取到的有效特征。不同模型结构由于固有架构设计和归纳偏置（Inductive Bias）不同，对特征的提取方式和提取能力存在显著差异，例如CNN擅长提取图片纹理，RNN、LSTM擅长提取时间序列。因此选择合适的模型结构至关重要，只有当模型能力与数据中的可分特征的形式和复杂度相匹配时，才能表现出理想的收敛趋势。针对彩色图片集，本研究采用了常用于图像分类的二维卷积神经网络为基础结构。考虑到即使同为CNN架构，采用不同的具体结构，例如不同层数/参数/正则/激活/连接/池化等，也会影响它们提取特征的能力，本环节选择了前面使用的简单三层CNN、MobileNetV3Small、ResNet18、EfficientNetB0这几种模型来进行实验，以期从中选出最适合的结构来对数据集难度进行具体评估。

值得强调的是，本研究的范围并不涵盖所有可能的先进模型架构，例如GCN、ViT。理论上，即使是通过卷积先验难以区分的数据，若能借助经过预训练的ViT，或是转化为适当的图结构来应用GCN提取特征，或许能够挖掘出更深层次的关联性。然而ViT缺乏初始偏置，GCN的性能表现往往与图结构的构建方式紧密耦合，它们的设计、优化和调参的复杂性远超常规卷积网络，难以在此类通用数据难度评估中作为直接对比的“即插即用”组件。正因如此，本研究仅能验证基于CNN架构的常用图片识别网络在该数据集上的分类潜力，不成为判定数据可分性的最终标准。

在应用深度学习框架来提取复杂特征之前，先用直接高维切割来确定初步基线。随机森林（Random Forest）和支持向量机（SVM）作为不依赖复杂非线性特征的传统机器学习，可以验证数据在原始高维特征空间中是否具备基本的可分性。对数据子集2取80%作训练集，采用随机森林和支持向量机进行分类，经过多次搜索最优参数后，得到其准确率分别为18.6%，17.8%左右，略高于随机基线。该程序保存为Rf\_SVM\_Classifier.py。基于传统机器学习的分类结果表明该数据集特征复杂，引入CNN来提取更深层次特征是有必要的。本实验采用的每种模型的层数、参数信息如表2-1所示。

表6-1 多模型任务中的不同模型对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 模型 | 总层数 | 卷积层 | 参数量 | 残差连接 | 收敛 |
| 三层CNN | 17 | 3 | 5,147,738 | 无 | 是 |
| MobileNetV3 | 106 | 34 | 655,746 | 有 | 是 |
| ResNet18 | 66 | 17 | 11,332,570 | 有 | 是 |
| EfficientNetB0 | 102 | 49 | 3,257,338 | 有 | 是 |

取80%作训练集，在JSON中设置"name": "multimodule"，在trainer.py中修改保存CSV文件的代码，将model\_name添加到文件名中，并且保存最佳准确率时的模型参数以备后续使用。在命令行中转到脚本所在路径，运行python trainer.py --cfg raitio08\_md1.JSON raitio08\_md2.JSON raitio08\_md3.JSON raitio08\_md4.JSON进行训练。训练结束后在作图程序中修改文件名匹配，运行python plot\_result.py --exp multimodule --mode f1作准确率曲线，其结果如图2-9所示。由图可见MobileNetV3准确率最低，仅30%；ResNet18相较于简单三层CNN效果提了升10%左右，EfficientNetB0在参数量更少的情况下取得了近似的学习效果。

图表, 折线图

AI 生成的内容可能不正确。

图2-9 多模型准确率曲线

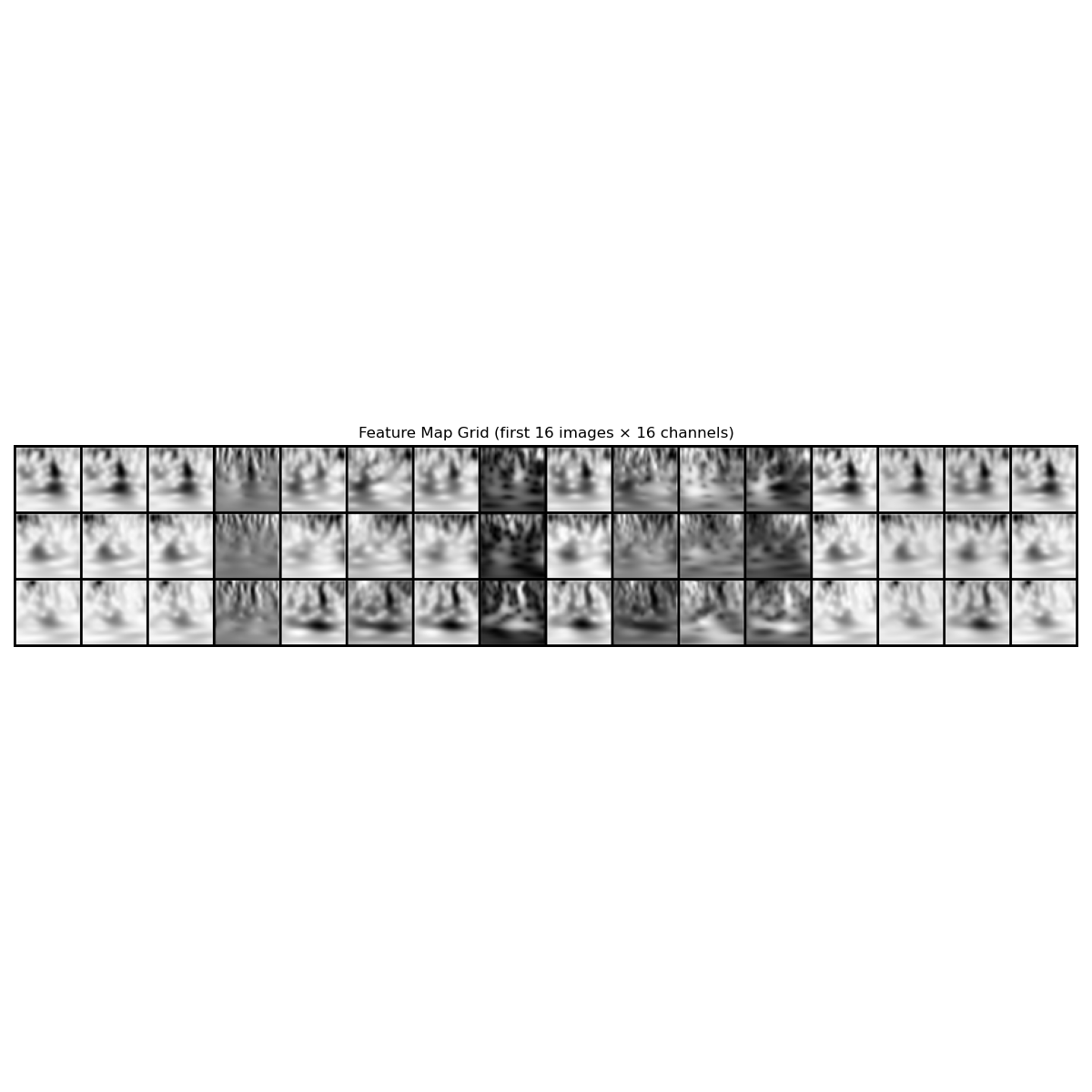
之所以采用F1分数而不是acc，是因为在划分训练/测试集时并没有将每个类别都严格按照相等比例划分，尽管每个类别内的样本总数完全相同，但是划到训练集里的样本可能不一样多。当数据不平衡时，模型会倾向于输出多数类来增大整体“猜对”的概率，使得预测结果中的少数类不公平地减少，而F1分数综合了预测为该类中正确的比例(精确率)与实际为该类中被正确预测的比例(召回率)，为少数类别单独计算了一个用来衡量预测正确性的指标，所以当样本数量极不平衡时，F1分数比Acc更具有说服力。本测试属于多分类任务，最终F1分数来自每个类别F1分数的宏平均。多模型测试结果表明，卷积先验对本数据集有一定的作用，同时使用深度网络有助于提升分类效率。

### 可解释性分析

可解释性（Interpretability）分析环节旨在理解模型学习了哪些模式、如何做出分类决策，以及从同一类别的样本中提取到的特征是否一致且具有判别力。值得注意的是，在多模型初步测试中，尽管ResNet18的准确率比简单的CNN高出约10%，但其训练损失并未显著降低，使得它的学习路径较为可疑。本研究认为，即使在准确率相近的情况下，是否具备良好可解释性、能否提取到有意义且一致性特征，应作为模型能否用于后续容量测试和泛化能力验证的依据之一。为了得知模型的学习是否建立在合理的语义特征之上，需要可视化数据特征图和模型梯度激活热图来查看它们提取到的局部纹理以及重点关注的区域。

为了保证实验结果的可再现性，用multimodule实验中保存下来的模型提取特征。从数据集中挑选2个类别的6张图片，00209.png、00219.png、00419.png；24209.png(类别3)、 24219.png、24419.png(类别4)，放入imgs文件夹下，类别编号可通过打印ImageDataset的self.class\_to\_idx查看；运行python infer\_cam.py --cfg .config/raitio08\_md1.JSON --images result/imgs/00209.png result/imgs/00219.png result/imgs/00419.png --cls 3将这三张图片输入保存的模型，分别得到它们由三层CNN模型提取到的类别3的100%可再现grad-CAM，保存在imgs文件夹下。同样的方法可以得到其它3种模型/另外3张图片/其它类别下的grad-CAM。

4个模型的浅卷积层的前16个卷积通道提取到的特征图如图2-10所示，由图可知所有模型的不同卷积核都得出了有效的特征图，没有全灰的“死核”，而且同一通道确实倾向于提取相似的模式。三层CNN网络主要提取了大块纹理，针对同一样本的特征图中有一些外表几乎相同，如图2-10 (a)；ResNet18的浅层特征与三层CNN非常相似，只不过经过深度卷积以后，特征图细节更加丰富，重复的图也更少，如图2-10 (b)；这两种模型的优点是相对另外两种来说特征更加直观，它们的卷积核有的在“看”图片的亮区，有的在“看”图片的暗区，只不过三层CNN层数少、感受野小，只能把相邻低级边缘、色块整合为粗纹理，在得出类别时需要大块区域取平均，而ResNet18在中层就开始捕捉更细粒度组合，并且逐渐压缩通道将其聚合，在模型深层只激活了关键像素，这种深度特征提取能力使其准确率提升了10%。MobileNetV3和EfficientNetB0由于深度分离卷积每个通道学习一个独立滤波器，提取到了更细的高频纹理，如图2-10 (c)、2-10 (d)，其中EfficientNetB0表现出对色块边缘的敏感性，它的准确率比MobileNetV3高出15%，与ResNet18接近，这表明该数据集的判别信息很可能跨尺度分布，准确率与纹理粒度的粗细没有必然关系。



1. 三层CNN特征图



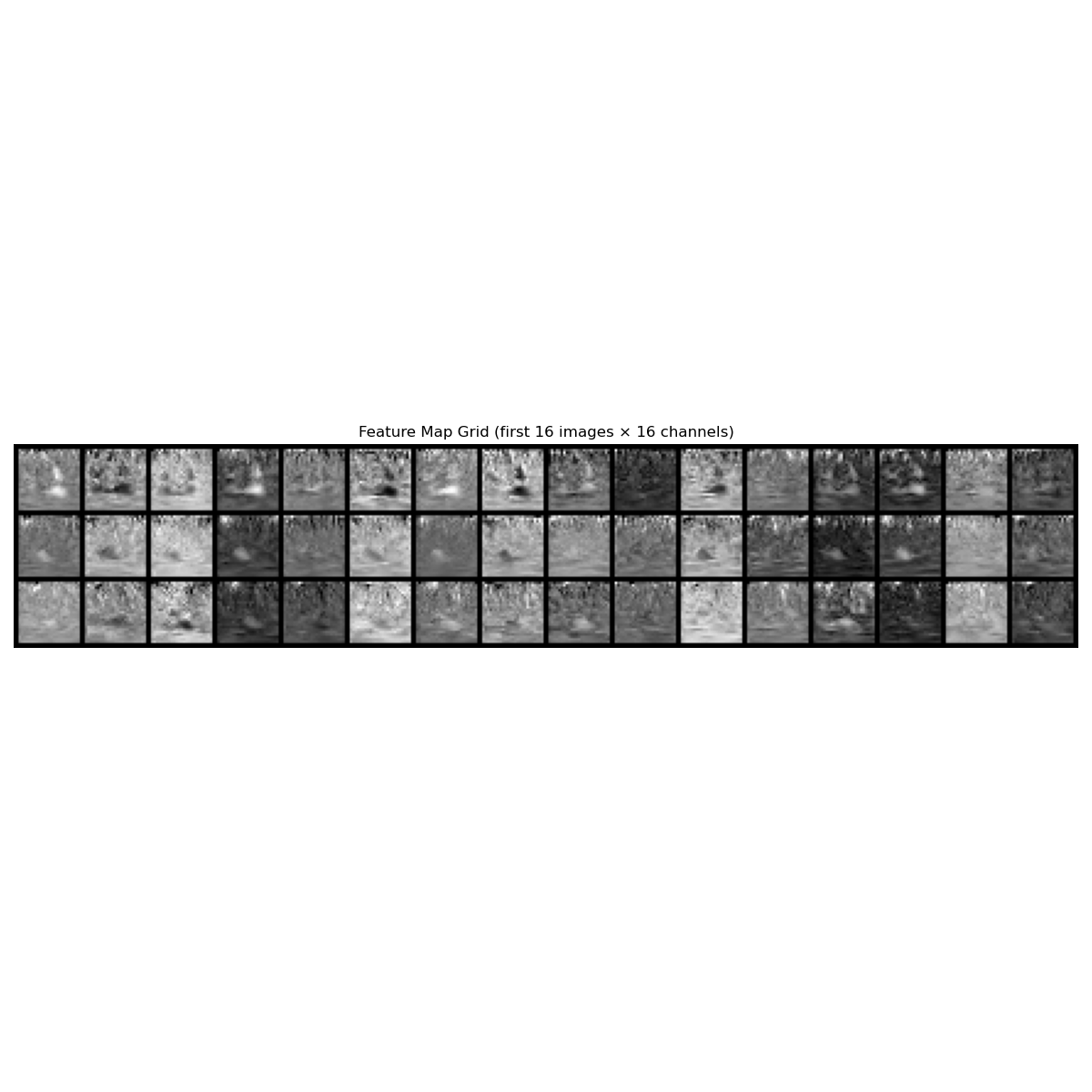
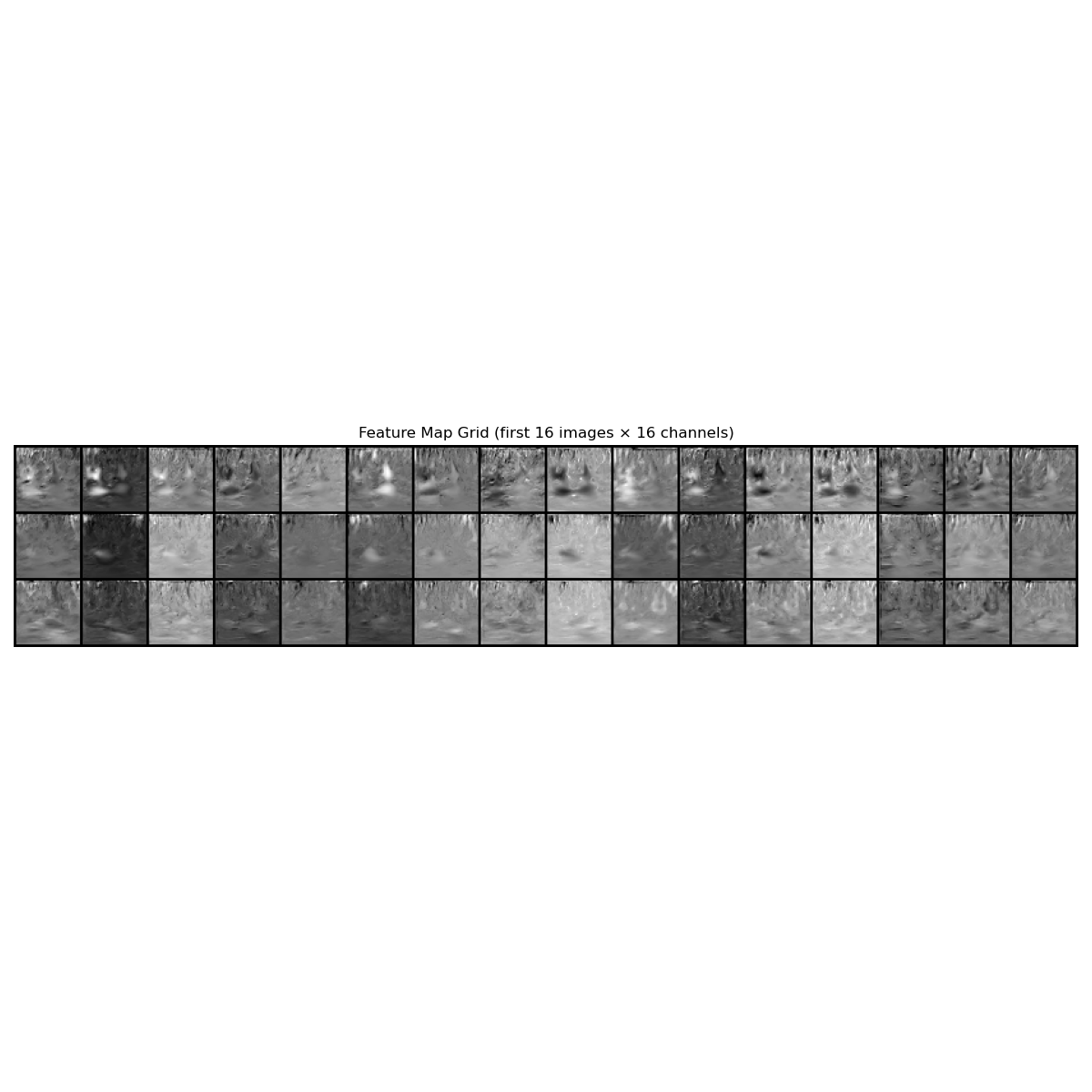
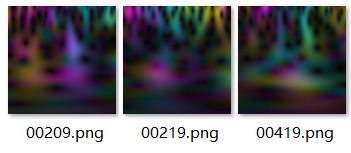
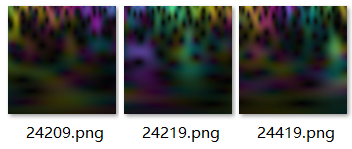
1. ResNet18特征图
2. MobileNetV3Small特征图
3. EfficitenNetB0特征图

图2-10 模型卷积层前16通道特征

为得知每种模型的决策路径，分别作这些样本在正确类别下的Grad-CAM来可视化模型决策的关键区域，如图2-11所示，其中左侧名称下方括号里是分辨率。

原图

(200\*200)

三层CNN

(50\*50)

ResNet18

(25\*25)

MobileNetV3

(25\*25)

EfficientNetB0

(50\*50)

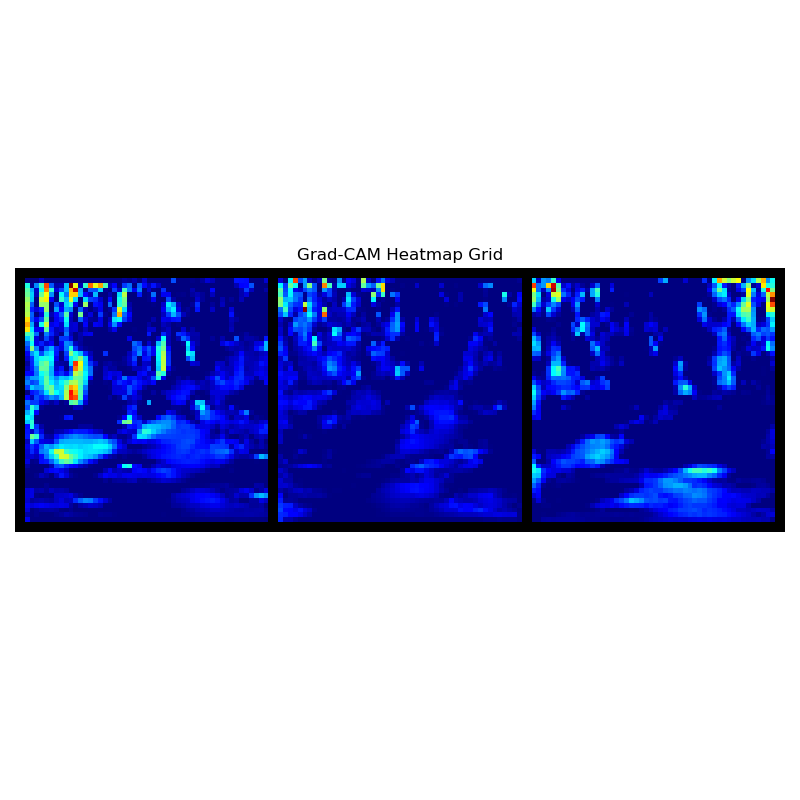
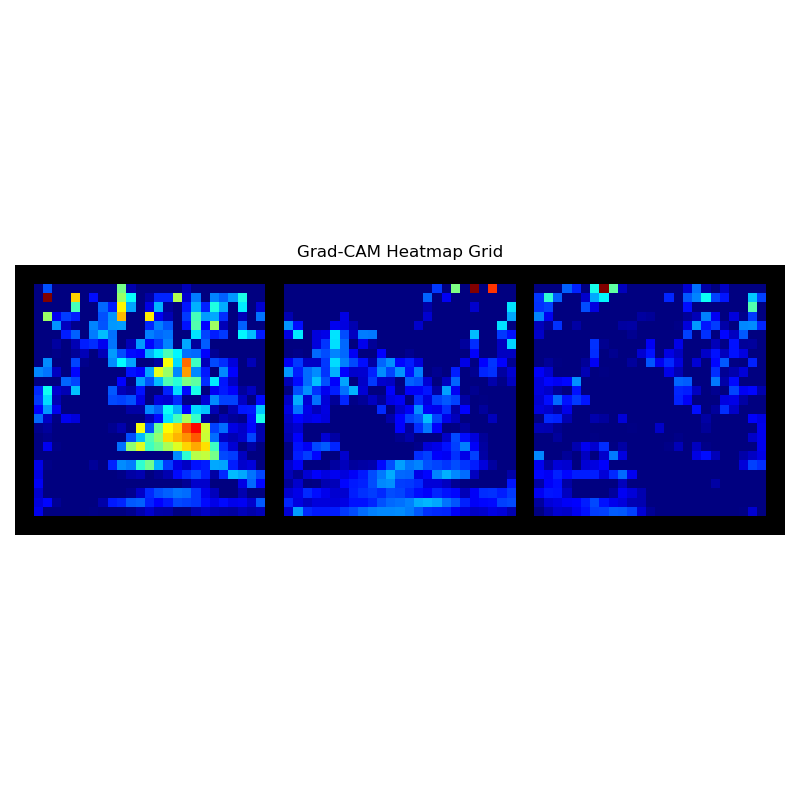
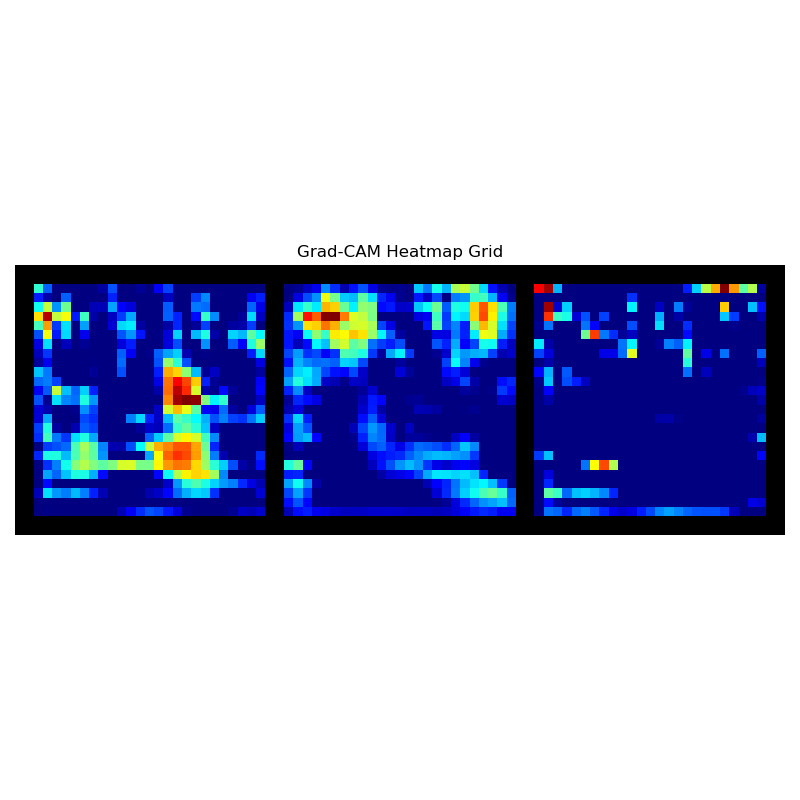
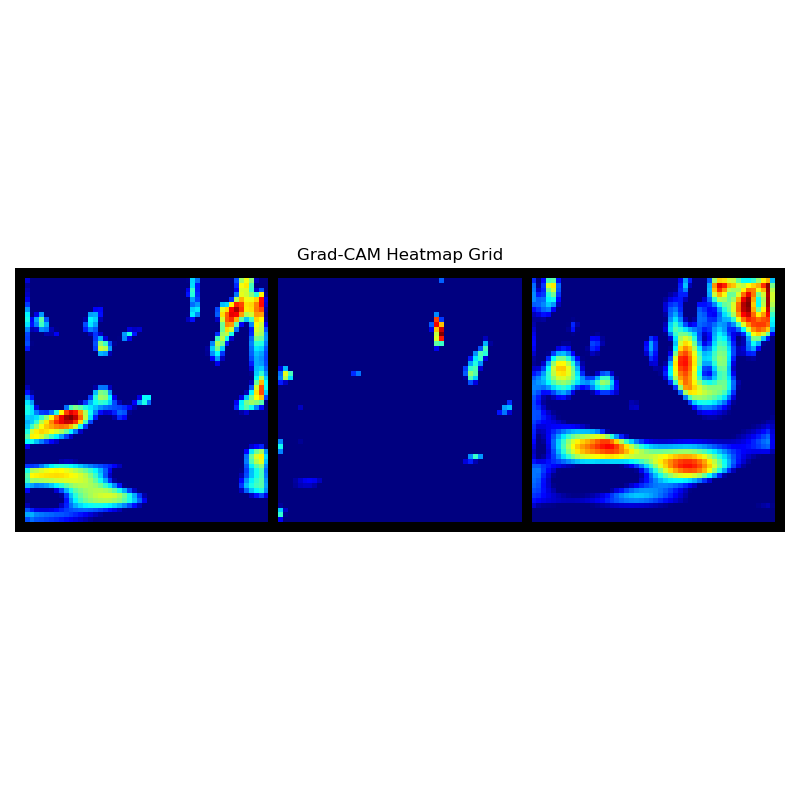
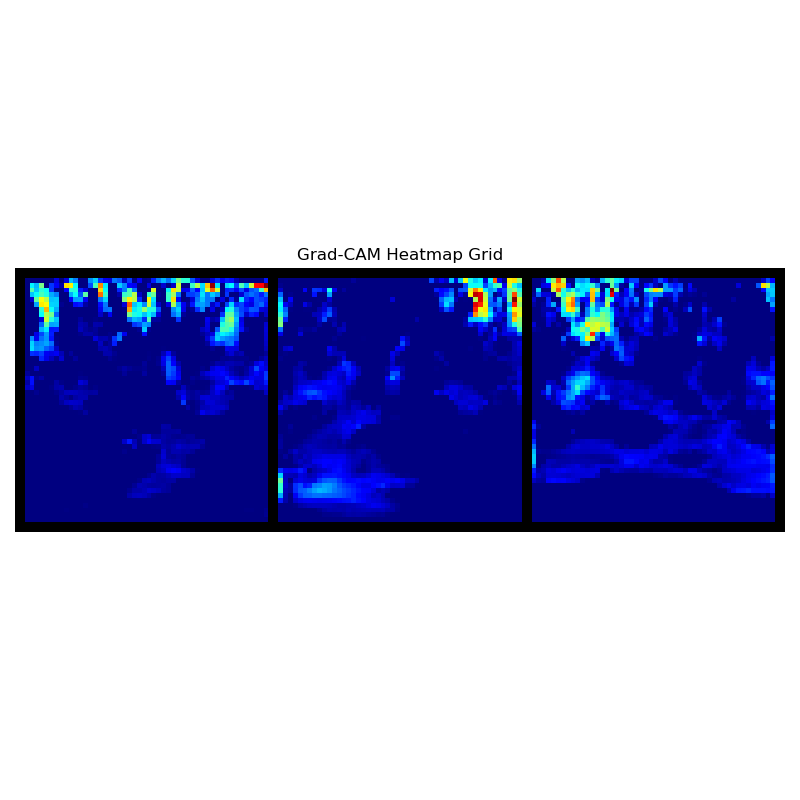
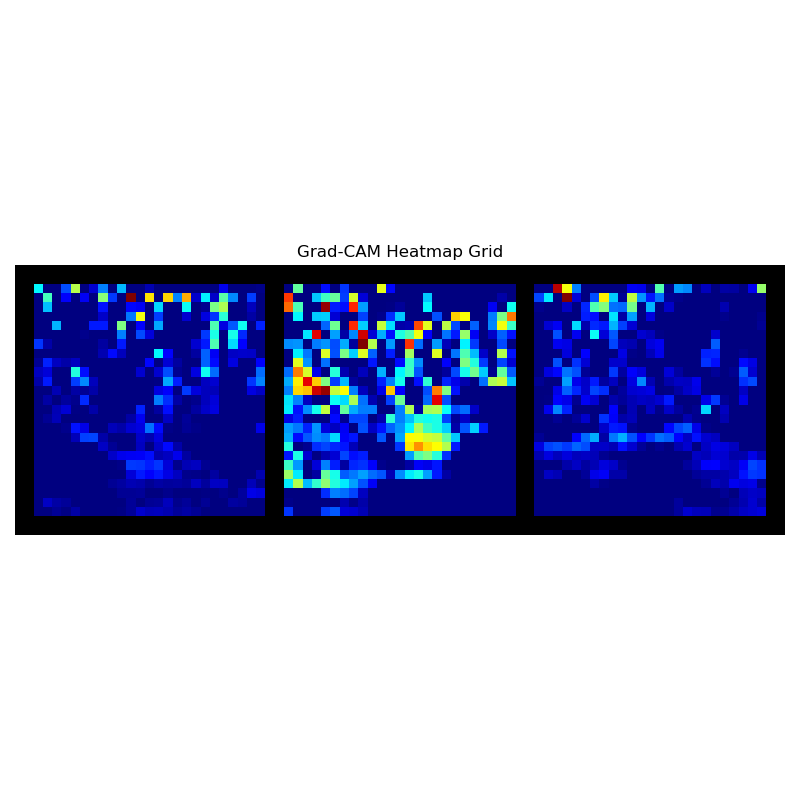
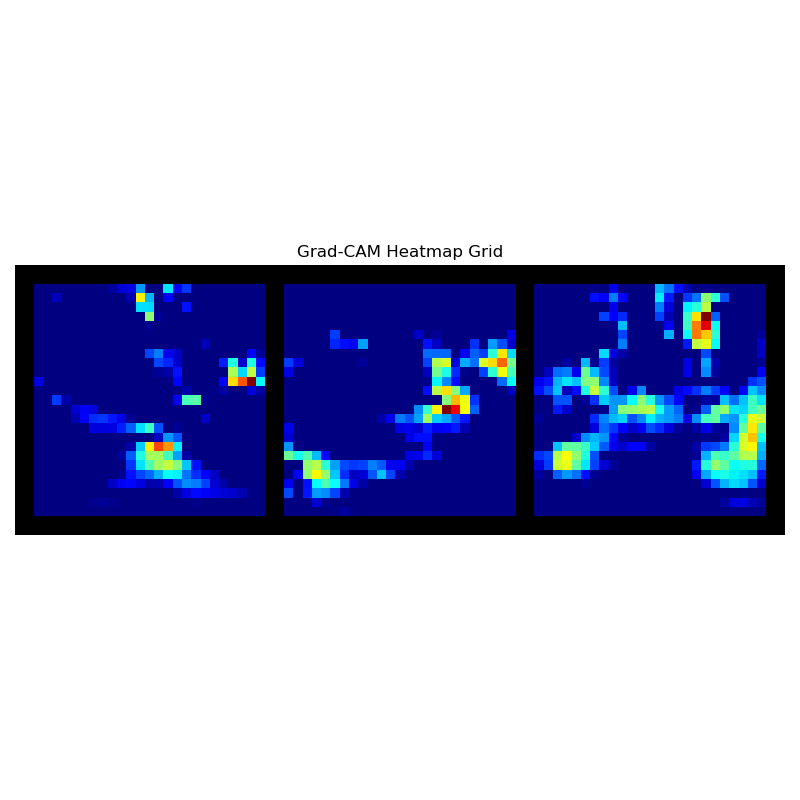
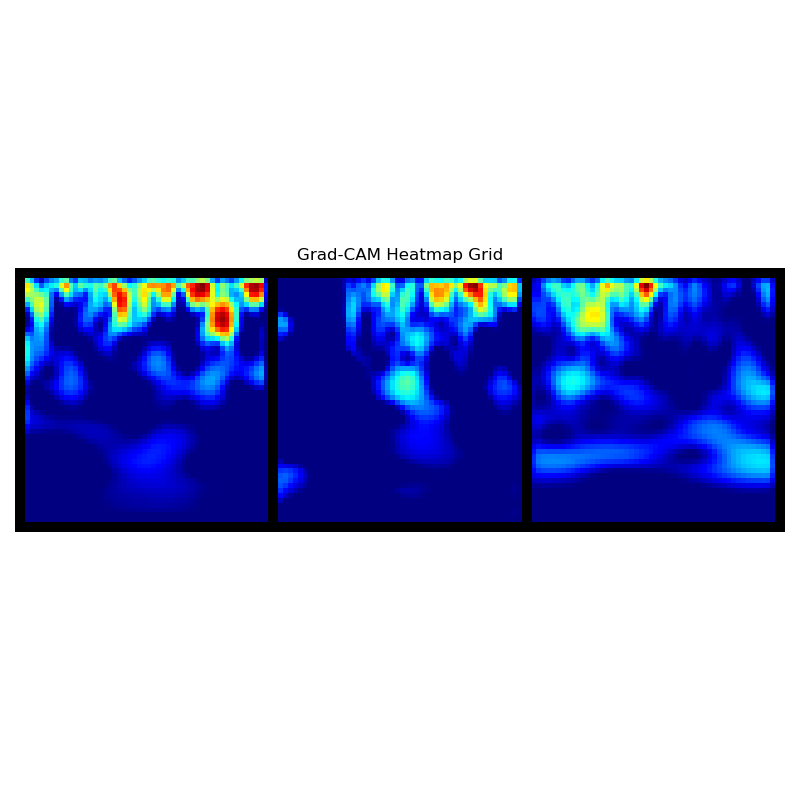


图2-11 4种模型的Grad-CAM可视化

虽然每组热力图的分辨率由于所在层的卷积通道数不同而存在差别，但是仍然能够看出它们与卷积特征图表现出了类似的纹理样式，即三层CNN和ResNet18纹理方向连贯、大块，随原图亮区呈现条带、网格状，MobileNetV3和EfficientNetB0 热区呈斑点状，激活区域更加“破碎”。图2-11显示不同模型对同一样本的关注区域非常不同，进一步支持了该分类任务存在多条潜在决策路径的推断，不同模型可在不同尺度、不同粒度上进行特征组合来对同一样本进行预测。然而这也从侧面印证了最初目测的结论，即这些样本本身缺乏直观且通用的语义特征或判别依据。该数据集似乎无法依赖人类可直接感知的局部颜色、纹理等视觉模式进行识别，只能寄希望于模型凭借其自身的结构优势和强大的特征提取能力，从看似无序的像素中挖掘出具有判别力的抽象表示或潜在关联。

### 特征固有维度

为了选出更适用于该数据集的模型结构，需要评估每种模型所学特征的信息紧凑性，从而在保留原始数据关键判别信息的前提下，优先选择能够使用更低自由度来表示特征信息的模型。信息紧凑程度可以通过特征的固有维度（Intrinsic Dimensionality）来量化。更低的固有维度意味着模型更高效地剔除了数据中的冗余和噪声，将关键信息压缩至更少的复杂度。此外，尽管模型在当前数据集上的分类准确率并不高，但是通过观察t-SNE图中各类别特征的聚类情况和边界清晰度也可以一定程度上判断它们是否内聚性强、具有判别力。

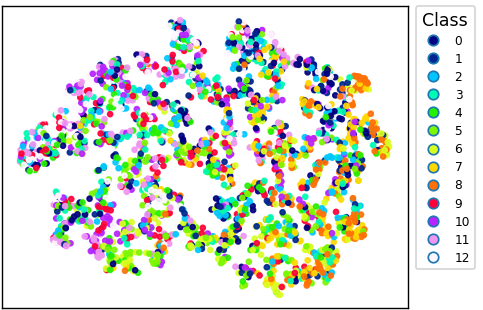
将所有数据输入multimodule实验中保存的模型来提取它们的原始特征。批次大小取决于可用显存，取一个适当的值使其迭代若干次可覆盖到所有样本。将数据迭代器的默认训练比例设为1，在命令行中运行python ext\_features.py --cfg .config/raitio08\_md1.JSON --batch\_size 2340将配置中的模型最后一层特征的原始Tensor和降维后的Tensor保存为.pt文件，修改配置换其它模型，ResNet18取layer4最后一层，MobileNetV3和EfficientNetB0取倒残差出口。运行python plot\_result.py --exp imgs --mode tsne --model\_name CNNModel --batch\_size 2340对其保存的特征进行T-SNE降维可视化，CNNModel替换为其它模型同理。得到4种模型的特征t-SNE可视化结果如图6-12所示。

图6-2显示4种模型在特征空间中构建的类别分布存在差别。三层CNN的 T-SNE图是“一锅粥”，暗示其高维边界非常复杂；MobileNetV3和ResNet18的聚类紧凑，由于训练比例取了80%，这些紧密聚集的特征其实主要来自于模型在训练阶段已充分记忆的样本，而对于其它错分和未能有效归类的特征，它们倾向于聚集在特征空间的中心区域或各类别簇向中心延伸的一侧，可能是模型中的线性层把向量重新旋转以适配交叉熵导致簇内个体被挤回中间重新求全局判别(通俗来说就是分错的样本被打回去)，不过更大的可能是模型由于没在训练集里见过而根本就不理解这些样本的内在模式，因为无法作出预测而将它们堆积在中心的模糊区域(意思是模型不“认识”，就给它另找个地方“搁起来”而不是放进某个标签“箱子”里)。与之不同的是，EfficientNetB0未出现类似的“摆烂”——它总是尝试将特征向量靠向最近的类别簇，使样本归入某种类别。推测这是由于MBConv的强烈平均池化使其倾向全局语义，这让EfficientNet具有更强的泛化倾向，因此推测它能在后续数据清洗环节中筛选出更具代表性的样本。而ResNet18的聚类模式更像是依赖于对训练样本的记忆来猜测未知数据，这可能就它虽然提升了准确率但损失函数并未显著降低的深层原因——它在内部优化过程中并未真正学到更本质的判别信息，只不过通过更个性化的记忆机制来强行拟合训练数据(通俗来说就是“死记硬背”)。值得注意的是，EfficientNetB0对个别sub-Class中的不同Magnitude簇形成了条状排列，暗示其可能具有向中间态Magnitude迁移的潜力。

图片包含 图表

AI 生成的内容可能不正确。图表

AI 生成的内容可能不正确。图表, 散点图, 气泡图

AI 生成的内容可能不正确。(a) 三层CNN (b) MobileNetV3

(c) ResNet18 (d) EfficientNetB0

图6-12每种模型提取特的征可视化结果

为了进一步得知每种模型压缩信息的效率，需要对它们所提取的特征固有维度进行比较。本实验采用了三种不同的特征维度计算方法，其结果如表6-1所示。

表6-1 多模型任务中的不同模型对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模型 | 特征维度 | MLE维度 | PCA-95% | TwoNN |
| 三层CNN | 64 | 3.36 | 3 | 3.56 |
| MobileNetV3 | 96 | 10.38 | 17 | 13.54 |
| ResNet18 | 512 | 10.52 | 30 | 14.45 |
| EfficientNetB0 | 320 | 7.59 | 8 | 9.81 |

由表6-1可知，4种模型得到的MLE 维度与 Two-NN接近，可见估计稳定，线性与非线性维度量级一致；所有模型的可分维度均远小于各类别的样本数量(180)，说明数据集在当前特征表示下数据量充足。同时观察到每种模型的直接特征维度(即表里的第2列，等于最后一层的卷积通道数)远高于对应的固有维度，然而除了三层CNN外其它模型在最后一层判别空间中都形成了清晰分明的类别簇，这表明t-SNE降维可视化本身不是导致类别特征在2D图中看起来搅在一块的原因，而是三层CNN在高维空间中的决策边界本身就非常复杂。三层CNN由于层数少、提取不到深度特征，尽管其原始特征有64维，但PCA-95%只有3，也就是说95%的信息都可压缩至3个自由度，这就解释了为什么它的卷积通道存在大量重复的特征图，同时也进一步印证了该数据集在粗纹理层面难以有效聚类，采用深度神经网络进行多层抽象特征学习是有必要的。

此外所有模型的固有维度普遍不高，说明该数据集的关键判别信息在15个维度以内即可表达，继续增加模型宽度/深度主要带来冗余度或鲁棒性，而非新的自由度。值得注意的是，EfficientNetB0的固有维度低于ResNet18，这意味着在准确率相近的情况下，EfficientNetB0能够将表征信息压缩至更低维度(意思是这个数据集对EfficientNet来说更“简单”)。在计算互信息(Mutual Information)时，低固有维度对于降低计算复杂度具有重要意义。因此虽然ResNet18性能发挥稳定、特征图直观、聚类性强，但是综合考虑决定使用EfficientNet模型来进行后续环节。

### 模型容量测试

容量测试环节旨在确定模型拟合给定数据集所需的最小有效参数量或层数，是评估数据集难度的核心环节，通过系统地调整模型容量来观察其对性能的影响，可以量化数据集的内在复杂性。当模型性能达到饱和或边际收益递减时表明此时的模型容量已经达到了数据集的难度阈值，选择“性价比”高的模型，有助于在充分发挥其学习能力、有效捕捉数据中的复杂模式的同时避免不必要的资源浪费，实现计算效率与模型性能的最佳平衡。文献[1]提出了一种EfficientNet缩放公式，只需调整一个参数即可使其按深度-宽度-分辨率成比例扩张，得到不同版本的EfficientNet，本研究取=1.2，=1.1。

在JSON中设置"name": "capacity"，通过"model\_args"将传入，取训练集比例为0.8，在trainer.py中修改保存CSV文件的代码，将参数添加到文件名中；运行python trainer.py --cfg capacity\_0.JSON capacity\_n1.JSON capacity\_n2.JSON capacity\_p1.JSON capacity\_p2.JSON capacity\_p3.JSON，训练结束后记录每种组合的最佳准确率。如果弹出警告：Initializing zero-element tensors is a no-op，意味着有的层因为参数太小而被削没了，查找发现都是倒残差层中的SE注意力通道，不涉及卷积层。本实验所采用的EfficientNet型号及其参数量、最高准确率如表6-2所示，其中=0为标准EfficientNetB0。

表6-2 不同模型容量时的最高准确率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| phi | 层数 | Conv | 参数量 | Best\_Acc | Time(min) |
| -2 | 185 | 32 | 864833 | 38.03% | 8.92 |
| -1 | 185 | 32 | 1234943 | 37.82% | 9.15 |
| 0 | 287 | 50 | 3255661 | 44.66% | 9.71 |
| 1 | 287 | 50 | 4637276 | 42.95% | 10.32 |
| 2 | 287 | 50 | 6623991 | 45.09% | 11.31 |
| 3 | 304 | 53 | 10450321 | 45.94% | 12.31 |

表6-2中的结果显示，增加模型深度能够提升分类效率，直到=0开始产生边际效应。当<0时准确率崩塌，表明模型容量开始不足以适应任务要求；然而同样深度下增加宽度不能提高准确率。此外发现训练时间随着模型参数量增加而变长，表明多出来的参数的确参与了更复杂模式的学习，尽管这种复杂度并未转化为同等比例的性能提升，由此推测该数据集的难度主要来自深层次的抽象特征，而非多样的纹理。运行python plot\_result.py --exp capacity --mode f1，得到每种参数时的准确率曲线如图6-13所示。

图6-13结果基本符合前面分析的规律，不过为了微小的性能收益而继续增加模型容量并非最优策略。由表6-1可知它所提取的特征维度，Tensor形状为[2340, 320]，MLE固有维度7.59，PCA-95% 维度为9，TwoNN为9.81，9/320约为2.81%，已经远超过一般分类任务10%~30%的信息冗余度要求，表明此时无论是数据量还是模型维度数都已相当充足，既不会因为类内样本过少、覆盖不了自由度而无法形成判别空间，也不是因为模型能力不足而导致数据集存在模型看不到的维度“盲区”，只不过由于数据特征在这些维度内表达的信息比较丰富，所以模型无法通过少量样本学习到完整的决策边界。

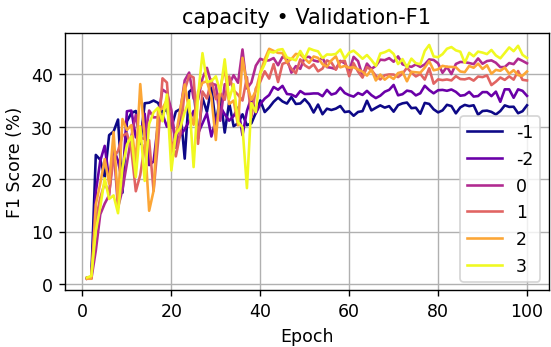


图6-13 不同容量模型的准确率曲线

模型容量测试的结果表明，经典EfficientNetB0可以适应该数据集的难度。关于固有维度与可分性的关系，可通过以下两个例子来理解。例如在一个分类任务中，样本的类别由颜色、形状、透明度、轨迹、出现时间这5个方面决定，然而每个类别仅有4个样本，此时模型难以学习到所有的维度关联，由于数据量不足以覆盖所有判别维度导致其不可分。但是这不意味着只要样本充足就必然可分，例如在另一个分类任务中，样本的类别只由颜色决定，一类偏红，一类偏绿，每类各拥有100个样本。此时其特征的自由度最多就3，即RGB三通道，数据量从维度覆盖角度看完全充足，但如果此时测试集出现一个紫色样本，虽然未引入新自由度，模型也因该样本落在了类别边界的模糊区域而无法准确预测。本研究需要重点解决第二个例子的问题。

## 数据噪声与可分性验证

数据噪声既可以是数据质量问题，也可作为衡量数据集难度的指标之一。在一个样本所包含的所有信息中，除了与类别相关的可泛化模式外，还有与标签无关的个体特征、随机误差，甚至与标签相悖的错误信息。模型本身无法区分噪声和真实的模式，可能在训练过程中努力去拟合这些噪声点，使其决策边界或预测函数变得复杂、扭曲，甚至失去对真实底层数据分布的捕捉能力，即“过拟合”。过拟合与数据本身、训练轮数、模型容量等都有关系，数据噪声太大会掩盖有用信号，同时如果模型容量过大、结构过于复杂，或者批次/学习率过小，也会诱导模型倾向于过度学习噪声而引起过拟合。选择适当的模型结构可以使数据噪声对方差的贡献落在正常范围，这就是在数据难度环节中选择性能-容量拐点处模型的意义。

通常来说，引入正则化项，如Dropout、权重衰减(L2正则)能够减小方差，使模型趋向平滑，可在一定程度上抑制过拟合，然而这只有在信噪比高或者噪声随机性强的时候作用才比较明显，对于低频、系统性甚至与真实信号高度耦合的噪声，正则强度的作用就非常有限。本研究使用的数据集有用信号相对较弱，甚至数据子集1中的类别在数据质量环节中已经因为噪声大到掩盖了有用信号而被放弃；此外由于特征过于抽象、无法通过目测识别，因此特征工程也很难进行，有必要采用基于信息审计的数据清洗方法。文献[2]提到了一种记忆-遗忘统计方法，即记录每个样本第一次被模型预测正确的 epoch，从未被记住或记住后又频繁遗忘意味着高噪声；文献[3]提到了查找持久高损样本的方法，把测试 loss 按 epoch 累积，并在达到一定程度时通过早停来避免过拟合噪声；文献[4]提到了通过交叉验证概率矩阵计算问题标签概率(label-issue probability)。本研究将结合已有方法，通过交叉验证来计算所有样本的正确率和验证损失，识别数据集中的高噪声样本。

### 基于交叉验证的渐进式清洗

为了精准识别数据子集2中的高噪声样本，本研究采用了一种基于交叉验证的策略，将数据集分成10等份，每次验证迭代中取8份做训练集，2份做测试集，直到10份中的任意2份组合都划入一次测试集为止；此时所有样本就可以得到9次预测情况及其损失变化。通过分析每个样本在9次中的预测表现，即可将样本分成以下3类：

1. 高噪声样本：如果某些样本在9次预测中均表现出持续的错误预测或高损失，在不能修改数据标注的情况下，有充分理由认为它们是噪声样本，应该考虑将其移除。
2. 高质量样本：如果某些样本在9次测试中表现出较高的准确率或低损失，则表明这些样本极有可能包含其所在类别的典型模式，应该重点保留。
3. 低代表性样本：鉴于数据集质量环节中，该数据集虽然难以达到很高的准确率，但是却表现出很好的稳定性，无论怎么划分训练/测试集准确率都接近，故合理推测，比起大量噪声样本，更大的可能性是该数据集中存在大量准确率一般的样本，它们不能算作噪声但是代表性不足，能否被准确预测很大程度上依赖训练集中的特定模式。这类样本可予以保留，以期在未来工作中可通过更先进的方法来挑战其分类效果。

根据交叉验证中的排列组合，总共需要进行45次收敛测试。考虑到模型在训练到一定轮数时准确率不再增加，反而容易发生过拟合，需要设置早停。由图2-9可知EfficientNet的准确率变化很大，连续15个epoch没有增加时可以认为模型的性能已经到达上限。在JSON中设置"name": "k\_fold"，添加参数"patience": 15；修改trainer.py，在Trainer类fit函数中判断最佳准确率的分支处加一个else，使其连续15个epoch没有更新最佳准确率时触发早停。从数据加载器脚本中获取完整数据集，用sklearn的StratifiedKFold按标签拆成10等份，用itertools组合、for循环解包，新建Dataloader来覆盖Trainer的数据迭代器，调用Trainer类进行训练，在每折收敛后进行一次测试，得到该fold全部测试集样本的正确次数、在正确标签上的概率和loss，在全部收敛结束后把全部2340个样本文件名、标签和9次作为测试样本时的各项预测表现保存到一个CSV文件中；该程序保存为kpair\_cv.py。为防止不同折数时Trainer保存的epoch信息互相覆盖，需要将折数添加到文件名里；在Trainer类的初始化函数中添加形参current\_fold，以便在创建实例时将折数传入。运行python kpair\_cv.py --cfg k\_fold.JSON --k 10进行交叉验证，训练结束后即可在k\_fold路径下看到保存的文件。

为直观量化整体可分性，计算9次预测结果与正确类别下的概率相乘后的平均得分，作得分段-样本数量直方图，整体结果和除原型外的具体类别结果如图6-14，6-15所示。结果显示数据子集2中超过 20% 的样本在全部 9 次交叉验证中一次都没有被预测正确，属于“噪声样本”。它们广泛分布于不同的工况中，表明它们的低可分性与特定工况没有必然联系。得分较高的样本数量非常有限，超过半数的类几乎没有高质量样本。这样的结果在一定程度上解释了整体准确率不高的原因，可利用paircv\_k10\_base.csv中保存的信息进一步筛选数据。

图表, 直方图

AI 生成的内容可能不正确。

图6-14 整体得分-样本数条形图

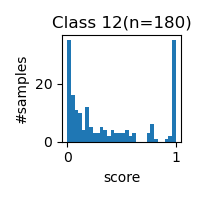
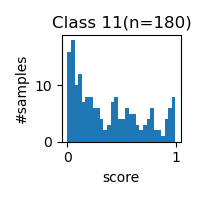
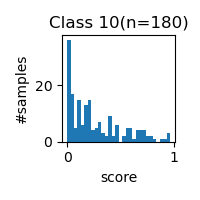
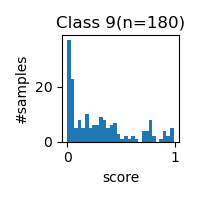
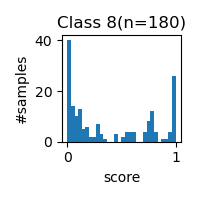
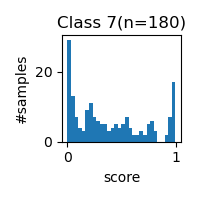
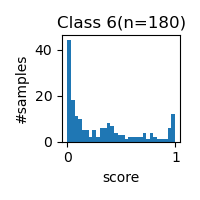
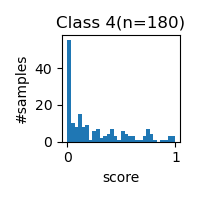
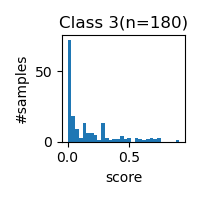
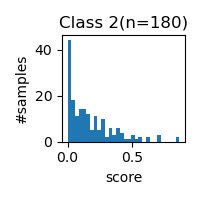
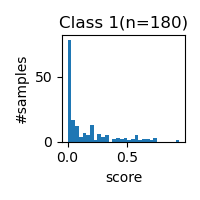
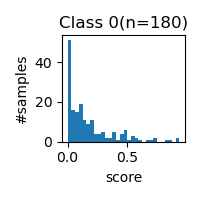


图6-15 每个类别得分-样本数条形图(除原型外)

然而这并不意味着必须删除大量样本、只保留得分高的那一小撮，因为删掉噪声样本有助于提升整体的准确率和其它样本的预测效果。本研究决定先从噪声最高的部分样本开始摘除，尝试屏蔽不同的预测正确次数、正确类别的预测概率、loss组合，直到试出保留最多可分样本的组合为止，该程序保存为kpair\_drop.py，运行python kpair\_drop.py --cfg k\_fold.JSON --k 10训练。经过多次尝试，本研究发现并不是筛选条件越严格准确率就一定越高，可见这些样本之间存在复杂的代表性关系，即使暂时被打“低分”的样本也可能在类内“凝聚力”中有着想不到的贡献。为了尽可能不“错杀”潜在高质量样本，有必要采用“渐进式”数据清洗策略。现根据每次准确率提升的阈值确定摘除条件，每一轮的具体操作及其结果如下。

第一轮：去掉paircv\_k10\_base.csv中除原型类以外所有9次的prob都在30%以下的样本，此时剩余2127 / 2340个，每个类别样本数均不低于140；其交叉验证结果保存为paircv\_k10\_round1.csv，本轮清洗完准确率上升5%左右，开始超过50%。

第二轮：去掉paircv\_k10\_ round1.csv中除原型类以外所有9次正确数与预测概率的乘积在0.6以下的样本，此时剩余1898 / 2127个，每个类别样本数均不低于110；其交叉验证结果保存为paircv\_k10\_round2.csv，本轮清洗完准确率继续上升，逼近60%。

第三轮：这一轮开始不再保护原型类。去掉paircv\_k10\_ round2.csv中所有9的loss之和在42以上，或所有9次正确数与预测概率的乘积在0.3以下，或所有9次的prob都在20%以下的样本，此时剩余1664 / 1898个，除了两个退磁40%的类只剩78，97以外其它类的样本数均不低于110；其交叉验证结果保存为paircv\_k10\_round3.csv，本轮清洗完准确率上升7%左右，介于58%到68%之间。

第四轮：考虑到准确率升高以后拟合变慢，而且收敛不稳定，可能前期跳到很高的值，这一轮开始将早停patience改为20。去掉paircv\_k10\_ round3.csv中所有9的loss之和在40以上，或所有9次正确数与预测概率的乘积在0.4以下，或所有9次的prob都在30%以下的样本，此时剩余1516 / 1664个，除了两个退磁40%的类只剩69，84以外其它类的样本数均不低于100；其交叉验证结果保存为paircv\_k10\_round4.csv，清洗完准确率大部分稳定在65%附近，个别超过70%。

第五轮：观察到训练准确率收敛变慢，从这一轮开始将学习率调度器的γ从0.1改为0.2，并取消早停。去掉paircv\_k10\_ round4.csv中所有9的prob都在60%以下的样本，此时剩余1444 / 1516个，每个类别样本数均不低于60；其交叉验证结果保存为paircv\_k10\_round5.csv，清洗完准确率整体逼近70%。

第六轮：去掉paircv\_k10\_ round5.csv中所有9的loss之和在36以上，或所有9次正确数与预测概率的乘积在0.3以下，或所有9次的prob都在50%以下的样本，此时剩余1385 /1444个，每个类别样本数均不低于50；其交叉验证结果保存为paircv\_k10\_round6.csv，清洗完准确率稳定在70%附近。本轮清洗后所得1385个样本记为数据子集2-A，数据子集2中的剩余样本记为数据子集2-B。

6轮清洗结束后，再次作得分条形图，如图6-16、6-17所示。图中结果表明剩下的数据已经比之前干净许多，包括原型在内的一半类别可分性已经相当好。

图表, 直方图

AI 生成的内容可能不正确。

图6-14 清洗后的整体得分-样本数条形图

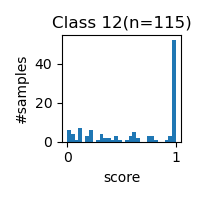
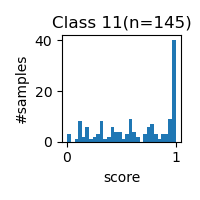
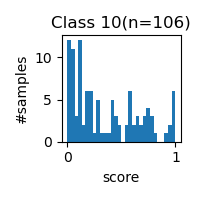
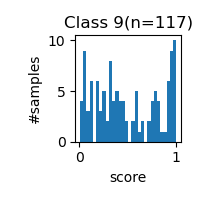
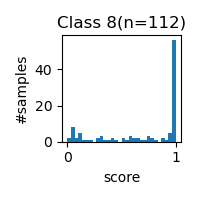
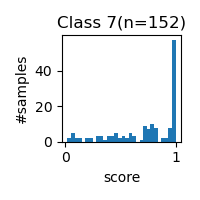
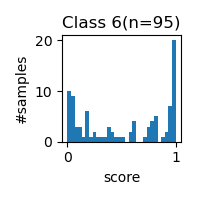
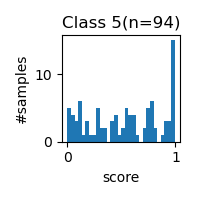
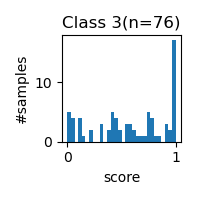
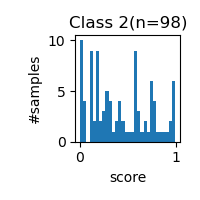
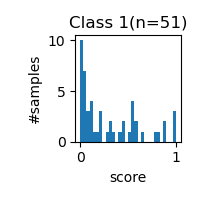
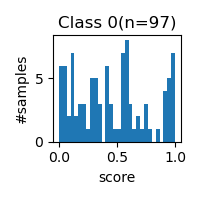


图6-15 清洗后每个类别得分-样本数条形图(除原型外)

实际上当最后一轮清洗结束后，仍有30个样本一次也没有预测正确，尽管这30个样本在前几轮的清洗中总是有被正确预测的情况。最后两轮清洗都没有太大收益，只是下限更稳定而已，说明此时在剩余数据中很可能已经没有了真正意义上的噪声样本：即使一次都没有被成功识别，也不代表它在预测其它样本的任务中没有起到任何作用。考虑到此时样本已经剩得不多，而且随着数据量越来越少，收敛曲线也更加不稳定，如果继续通过严格筛选来迫使准确率提高，那么很有可能导致类内数据分布偏移(Distribution Shift)，甚至每个类只剩下少数极端样本。因此不建议继续清洗，而是要先从原理上验证数据集的可分性。

### 贝叶斯下界与K-NN算法

用机器学习的方法对数据进行有效识别的前提是数据本身必须包含可学习、可分的模式。机器学习的核心是让模型通过可见的数据样本学习一个特征分布，使其逼近数据的真实分布；通常模型拟合与之间的吻合度越高，其预测效率和泛化能力就越强，这就是深度学习的基本原理。然而即使拥有完整的数据分布知识、使用了理论上最优的分类器（贝叶斯分类器），也可能由于其真分布本身的重叠而无法突破一个理论上可达的最低错误率，即贝叶斯误差率(Bayes Error Rate)：

依据数据处理不等式，任何复杂模型所能从数据中提取到的特征信息的判别能力都不可能超过原始数据与标签之间固有的互信息。如果数据本身的区分度就存在统计学上限，那么无论采用多么先进的模型架构、多么强大的算法，投入多么高的资源，都不可能凭空挖出数据里原本不存在的判别信息而使其分类效率“无中生有”地突破贝叶斯误差率。例如，当或p(label∣features)≈uniform时，其，那么任何方法都不能使acc>1-。 因此本研究认为，提升机器学习在分类任务中效果的关键在于量化其贝叶斯误差造成的上限1-以及当前准确率与1-的距离，而非冲着100%去盲目升级模型或算法。

然而在真实的任务场景中，无法直接计算——能够完全表示数据的全部特征信息、完全过滤无关噪声的完美特征空间不可得，即数据真分布不可知。因为现实中没有无穷多个样本或者足够充分的先验知识，只能借助有限的模型、有限的样本去逐渐逼近贝叶斯分类器。考虑到本研究中使用的数据集，没有其它类似数据可供学习先验知识；没有原理级别的可泛化性能够“实锤”；没有肉眼可见的区分度来做“特征工程”，是一个完全由数据驱动(data-centric)的应用场景，所以只能暂时将准确率作为指标来反推当前特征空间的可分性，从而借助K近邻算法(KNN)的渐近性质来估算一个近似上限。

KNN是一种用于分类的简单机器学习算法。在预测一个未知样本时，KNN分类器会考察其K个最近样本的类别，将其归类为多数邻居所属的类别；在数据规模足够大的前提下，通过选择合适的k值，KNN算法能够提供对理论上最低错误率的有效估计。尽管实际上样本数量有限， KNN 精度受方差影响(约)，但本研究中的数据规模在2k量级，仍然可以为评估分类任务的固有难度提供一个实际可操作的基准参考。

然而，“度量”的前提是存在一个表征空间（Feature Space），如果没有这个可度量空间的定义，那么所谓的位置和距离就无从谈起。原始数据作为图片，虽然存在直接的像素空间，但是像素空间中的样本分布与当前任务之间通常是错位或无关的(否则就根本不需要训练深度模型，直接拿随机森林或SVM切割就行了)；因此需要先让模型学习一个任务相关的特征空间，然后基于这个特征空间来进行测距和计数。然而这就有一个问题：模型是用数据训练出来的；而判别可分性又要求被观测样本的位置没有被记忆塑造——否则它会与标签之间建立某种个性化、强关联映射而导致评估失真(通俗来讲就是它都背过了，再认出来有什么好奇怪的)。所以这就是一个悖论，它要求同一批数据，既要当裁判又要当选手。为解决这一问题，本研究采用k折交叉验证的折外提取特征(out‑of‑fold)策略，把数据分成k份，每次用k‑1 份训练模型，然后对其余样本提取特征，如此循环k次即可得到全部数据的泛化表征。该程序保存为kfold\_trainer.py。尽管由于每fold中训练集不同会导致k个特征空间存在差异，但是这不意味着折数越多表示越精确——虽然k越大时训练比例越高，表征更接近“全量模型”，但是由于每折模型分开训练，不同折之间的方差会随折数增多而被放大。根据经验通常取5~10 折。

本研究取k=5，用EfficientNetB0分别对数据子集2、数据子集2-A、完整数据集运行交叉验证，类别数分别为13、13、26；为了保证结果的可再现性，采用保存好的模型对折外样本提取特征。在trainer中保存每fold准确率达到最高时的模型参数，在kfold\_trainer.py中将所有样本的测试flag保存为CSV表格，当某个样本在某一fold中被划分为测试集时，其对应的flag为1，否则为0；运行python kfold\_trainer.py --cfg k\_fold.JSON --k 5进行交叉验证，运行结束后可在result/k\_fold路径下看到相应的文件。然后从保存下来的CSV文件获取训练/测试集划分情况，将所有样本分别输入前面训练好的k个模型得到特征，用其中的测试集特征用于评估，训练集特征用于对齐。其伪算法表如算法2所示。该程序保存为Kfold\_features.py。

|  |  |
| --- | --- |
| 算法2: 提取特征+对齐主程序 | |
| 输入: 每fold训练好的模型(.pth)、保存训练Flag的表格(.csv) | |
| 输出: 对齐后的特征文件(.npz), 元数据文件(.csv) | |
| 1: | 初始化：模型、配置与输出目录 |
| 2: | **for**每个数据集**do** |
| 3: | 加载模型与测试Flag |
| 4: | **for** 每折 **do** |
| 5: | 创建训练+测试集索引 |
| 6: | 创建训练+测试集的数据迭代器 |
| 7: | 加载模型权重 |
| 8: | 注册前向钩子到目标层 |
| 9: | **for** 数据迭代器里的每个批次 **do** |
| 10: | 前向获取特征且进行平均池化 |
| 11: | 保存样本特征、标签、折号、训练/测试Flag |
| 12: | **end for** |
| 13: | 移除钩子 |
| 14: | **end for** |
| 15: | 合并特征 |
| 16: | **折间对齐**(见算法3) |
| 17: | 合并保存所有特征 |
| 18: | **end for** |

这样就用每折训练好的模型提取到了全部样本的特征，每个样本保存了5次，其中4次为训练集，1次测试集。之所以需要跨折对齐，是因为模型学习到的表征空间压根就不是个稳定的东西，即使是dropout随机性、mini‑batch采样顺序等微小扰动都会被累积梯度和权重旋转(尤其是BatchNorm统计和Adam动量)放大而导致特征子空间发生漂移，因此虽然固定了随机种子且每折有60%的训练样本重合，但是其直接提取的类别簇质心距离仍远不及每折之间的漂移/旋转。而这就是每折需要保存全部样本特征的原因——需要依赖它们的重叠部分进行跨折对齐。折间对齐的伪算法表如算法3所示。

|  |  |
| --- | --- |
| 算法3: 基于训练质心的折间对齐 | |
| 输入: 特征、标签、折号、训练/测试Flag | |
| 输出: 对齐后的特征矩阵 aligned | |
| 1: | 初始化：折数编号、特征容器 |
| 2: | **for**每个类别c **do** |
| 3: | 找到该折中的训练样本 |
| 4: | 计算质心 |
| 5: | **end for** |
| 6: | **for** 每个非参考折 **do** |
| 7: | **for**每个类别c **do** |
| 8: | 找到该折中的训练样本 |
| 9: | 计算质心 |
| 10: | **end for** |
| 11: | 找到在两折中都有足够训练样本的有效类别 valid |
| 12: | **if** valid类别数 >= 2 **then** |
| 13: | 用正交Procrustes计算旋转矩阵 R |
| 14: | 对当前折的所有特征应用该旋转 |
| 15: | **else** |
| 16: | 跳过旋转 |
| 17: | **end for** |
| 18: | **return** 对齐后的特征矩阵 |

运行python Kfold\_features.py --cfg k\_fold.JSON提取特征，结束后可在实验路径下看到每个数据集的npz文件，文件内所有来自测试集的特征Tensor就是该数据集在EfficientNetB0模型表征空间下的raw分布。可以用它们来计算KNN准确率。

KNN的以下三个性质使其能够被用于估计贝叶斯误差率。

1. 局部性：KNN只考虑观察点x附近的K个样本，使其能够通过选取合适的邻居数K而无视决策边界的复杂性和远处噪声样本的干扰。
2. 非参数性：KNN不对数据分布做任何先验假设，因此能够逼近任意复杂的条件概率分布P(Y|X)，这正是贝叶斯分类器所需要的。
3. 渐近收敛性：随着样本总数N增多，每个点x附近的K个最近邻会越来越接近该点的真实条件分布，从而KNN的预测会趋近于贝叶斯决策规则argmax P(y|x)；因此当N→∞且k随样本数适当增长(k→∞但k/n→0)时，KNN分类器会收敛到贝叶斯最优分类器()。

加载保存了每个数据集特征的npz文件，取其中来自测试集的部分计算KNN准确率及置信区间(排除自身)，同时进行t-SNE可视化；此外为了查看数据子集2A相较于数据子集2而言有没有洗歪/洗过头，对它们跨域正交对齐后的特征进行度量，计算质心距离、FID类似度量、最大均值差异(MMD)和类内方差；该程序保存为KNN\_measures.py。直接运行，得到每个数据集的样本数N、邻居数K、准确率、95%置信区间如表6-3所示，dataset2和2A的度量结果如表6-4所示。

表6-3 每种数据集的KNN准确率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据集 | N | K | Train\_Acc | Test\_Acc | Test\_95%CI |
| all | 4680 | 120 | 77.71% | 24.66% | (77.11,78.30) |
| subset2 | 2340 | 90 | 95.84% | 44.06% | (42.06,46.08) |
| 2A | 1385 | 30 | 99.62% | 70.25% | (67.79,72.60) |

表6-4 清洗前后的数据子集所有类别度量

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| class | 样本数 | L2距离 | Cos距离 | FID\_like | MMD\_RBF | 2方差 | 2A方差 |
| 0 | 127/180 | 2.137 | 0.019 | 14.627 | 0.06 | 24.34 | 27.676 |
| 1 | 51/180 | 2.874 | 0.048 | 125.633 | 0.11 | 144.594 | 27.258 |
| 2 | 98/180 | 3.394 | 0.04 | 209.349 | 0.122 | 254.139 | 35.701 |
| 3 | 76/180 | 5.814 | 0.073 | 163.257 | 0.25 | 162.728 | 38.315 |
| 4 | 127/180 | 2.137 | 0.02 | 14.627 | 0.06 | 24.34 | 27.676 |
| 5 | 94/180 | 3.019 | 0.087 | 17.506 | 0.117 | 20.812 | 25.591 |
| 6 | 95/180 | 2.613 | 0.053 | 25.664 | 0.104 | 26.422 | 56.982 |
| 7 | 152/180 | 1.97 | 0.025 | 12.066 | 0.064 | 20.775 | 24.34 |
| 8 | 112/180 | 4.285 | 0.026 | 39.558 | 0.12 | 35.017 | 48.249 |
| 9 | 117/180 | 2.49 | 0.029 | 37.963 | 0.081 | 52.543 | 21.967 |
| 10 | 106/180 | 1.924 | 0.034 | 64.771 | 0.066 | 96.576 | 29.594 |
| 11 | 145/180 | 1.91 | 0.039 | 282.131 | 0.099 | 378.114 | 64.112 |
| 12 | 115/180 | 4.382 | 0.055 | 262.972 | 0.096 | 254.214 | 218.359 |

表6-3结果显示在对齐后的表征空间内，模型训练集已可驱动77~100%的近邻纯度。其中数据子集2和2A的训练集准确率均达到95%以上，证明对齐措施有效，能够把旋转误差压到可忽略；考虑到保存的模型已经充分记忆了训练集，推测完整集低至77.71%和subset2的略低的现象是由于低质量噪声样本对类别簇分布产生干扰而增大对齐方差所致。在此基础上，全量测试集的最优可分性分别为 24.7 %，44.06%，70.25%，接近模型直接分类效率，该结果说明模型当前分类器已逼近表征-条件Bayes极限，进一步提升需效果要更强的特征提取器。三个数据集的t-SNT可视化特征图如图6-15所示。

表6-4结果显示，由subset2清洗后得到的数据子集2A在不同类别上发生了不同程度的偏移。其中Class 3质心位移最大，欧式距离>5；Class12虽然其它指标良好，Fréchet 距离却异常地大，说明它们的协方差结构存在显著变异；Class 1 2 3 10 11的方差收缩超60%，有可能清洗过度；除此之外其余7个类别表现良好，可视为“域对齐完成”。然而这样的变化也是2A相较于subset2准确率能够提升的原因，退磁故障类的可分性本身就不高，为了提升效率不得不筛掉大量样本，导致清洗后的方差明显收窄，类别3有可能被分类器当成一个新类。由图6-15可知清洗后的决策边界变清晰，开始靠近「可分」的标准，表明本环节的数据清洗是有效的。

图表, 散点图

AI 生成的内容可能不正确。图表, 散点图

AI 生成的内容可能不正确。图表, 散点图

AI 生成的内容可能不正确。

(a) 完整集 (b) 数据子集2 (c) 数据子集2A

图6-15 三种数据集在EfficientNet表征空间下的原始分布

值得强调的是，本节中的结论并不作为该数据集可分性上限的最终结论。因为用于估算KNN准确率的特征本身就来自一个深度学习模型，KNN分类器的作用仅仅是取代了模型的分类头，它所导出的贝叶斯误差上限其实是基于该模型的理论上限。正如本日志4.1节所说，如果采用CNN结构以外的特征提取器，还有可能得到更加有效的表征。因此本环节的最大意义在于为模型能力的上限提供了一个理论解释，旨在解决从“我做不出来”得出“它不可行”的问题。

## 结论及建议

### 总结&展望

本章验证了一个包含永磁同步电机故障信息的26类别4680样本200\*200 pixel \*3通道png图片集在分类任务中的数据集质量和难度，根据其质量评估结果将整个偏心大类、轴承故障和退磁的10%、20%摘除，用剩下部分(数据子集2)继续评估了数据集难度，检查了它在不同CNN架构的神经网络模型下的可解释性，根据其难度评估结果选择了EfficientNetB0进行了基于交叉验证的数据清洗，得到数据子集2A；最后通过K近邻算法估计了完整数据集及其不同程度清洗后的数据子集的可分性上界，分别为24.66%、44.06%、70.25%；经过同标签度量和t-SNT可视化，发现本章采用的数据清洗策略基本有效，然而由于数据质量过低，在摘除50%类别、70%以上样本的情况下仅将上限提升至70%。

原始信号来自有限元仿真，每个类别对应的模型之间仅有故障部分的区别，已最大程度上排除了随机噪声、设备伪影等因素可能造成的信号干扰或标签泄露；而且数据已经对齐，所有样本均来自相应故障类型的典型工况，除了轴承故障由于Flux2D中看不见轴承而只能通过动/静偏心与摩擦力来模拟以外，偏心和退磁故障的有限元模型均严格遵循该种故障的定义。观察原始数据，可以看到每种工况下的三相电流的确已经随着故障种类和程度的变化表现出了细微的差别，而且这些变化已经被预处理步骤提取到了图片中。小波变换本身并不制造特征，它只是电流信号从时域到时频域的重新排版。因此可以认为数据集的采集和预处理流程已经尽可能还原了故障类别的直接影响。考虑到24.66%的准确率上界过低，根据本研究中的局限性，推测最可能的原因有以下三点：

1. CNN学习能力有限。~~可能CNN架构并不适合提取时频能量图，如果采用特定的GCN或直接学习原始时间序列也许能挖出新的东西~~(已经试过了，几乎没用。GCN部分代码见“Validation\_GCN”，如果没有可解释先验，这个数据集几乎没救了，与其调模型架构还不如从有限元仿真原理端入手找找公式)。
2. 仿真精度过低。由于预算有限，采集数据用的有限元模型网格比较粗糙。增加网格密度或许能抑制原始数据中的噪声，但减小采样时间/仿真步长应该没什么用。
3. 以非侵入式电流信号为指标进行永磁同步电机故障诊断的可行性本身存疑。现有的研究成果大多基于平稳工况，而本研究使用的数据集，每个类别包含了5种相差较大的不同工况。为了完成故障诊断，可能需要换成温度或振动信号。

### 日志复盘

本日志按照结论需要和条件准备顺序展开实验，具体步骤和实验条件已经记录在日志中，所使用的脚本都是Demo级，模型都是经典架构，所记录的实验现象也都经过了反复检验，基本可以排除偶然误差干扰，想必即使没有源代码也可以成功复现。每个环节的局限性都已经当场写在了文字说明中。日志中提到的大部分脚本采用命令行运行，是为了把代码和配置分离开，以免在不同实验中忘记改回参数。由于实验项目较多，而且很多脚本功能只用了一次，因此没有为所有实验都整理好可以直接运行的管理接口，复现时仍需改动脚本/配置/文件路径(有些在脚本注释部分)。本日志记录的实验结果均来自该环节论述的限定条件，已经尽量规避了数据泄露问题。将在整理好后上传至GitHub。

### 研究定位(作者的自言自语)

本研究服务于学位论文《基于元学习的永磁同步电机故障诊断》，这个题目本身就已经假设了该方法的可行性，因此本章的数据集评估结果给后续工作带来了很大压力。通常来说元学习并不旨在提升准确率，它甚至是为了牺牲一部分准确率去换节省样本数量。它的原理和迁移学习类似，都是依赖先验知识来加快新任务的学习(先验知识的作用，就比如在识别手写字的任务中，一个字长得既像1又像7，由于边界模糊而难以辨识；但如果引入“个人书写习惯”这个先验，知道了写这个字的人习惯把7的横写得很短，而1就只写个竖，那么就能更容易确定这是1还是7)，因此它发挥作用的前提是需要找到一个能为目标任务带来有用信息的元任务。然而考虑到该数据集的可分性上限和抽象程度，既不可能把它用作元任务，也不适合把它用作目标任务。其实之所以采用深度卷积网络，是因为本人毕业论文挂的项目里写的是这个方法。然而现在看来为了能毕业，只能另寻出路了。

# 参考文献

1. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
2. Memorization in deep learning: A survey，arXiv:2406.03880v1 [cs.LG] 06 Jun 2024
3. Early Stopping Against Label Noise Without Validation Data，arXiv:2502.07551v1 [cs.LG] 11 Feb 2025
4. [Label Errors and Confident Learning · Introduction to Data-Centric AI](https://dcai.csail.mit.edu/2024/label-errors/)

# 附录：Python神经网络模型搭建日志

## 机器学习基本概念

本文主要介绍在数据集验证环节中为了强调现象及结论而没有过多提到的关于神经网络模型的详细说明。一个神经网络模型可以视为一个系统，即给它输入某种信息就会产生特定的输出。与一般被“写死”的系统不同的是，神经网络模型可以根据其实际的输出与你所期望的输出之间的误差(loss)来调整自身构造，使其输出逐渐逼近你所期望的输出，也就是“机器学习”；目前的人工神经网络模型以分层结构为主，输入信息经过逐层处理以后得到输出；每一层的每个节点为固定的函数，不同层之间的节点靠权重连接，模型通过调整这些权重来修正输出，从而逼近给定数据。然而真正让机器学习有用的是模型对从未见过的数据的输出，即“泛化能力”；让训练好的模型在新数据上的输出更加逼近它在训练数据上的效果，就是机器学习的目标。

本研究基于Python第三方库Pytorch搭建人工神经网络模型，按照习惯的做法，把每个模型分别创建为Class，即Pytorch中nn.Module的子类，此时它们的实例就可以拥有你在模型类中定义的结构，而且具备nn.Module的属性和功能。每个模型类必须包含初始化方法\_\_init\_\_()和前向传播方法forward()；在初始化里使用super()调用它的超类，使其继承nn.Module的方法，然后创建这个模型里能用的东西，比可用的层、参数等；前向传播方法就相当于“传递函数”，其形参代表模型的输入，返回值代表模型的输出，在调用模型实例时调用该方法。简而言之，创建实例时填入的是初始化方法的实参，即“构造函数”的参数；调用实例时填入的是前向传播的实参，即模型“系统”的输入，通常只接受Tensor类型。此外可以根据需要在模型类中定义其它方法，如初始化方法等。

nn模型是通过预测值(模型输出)与真实值(给定输出)之间的误差来训练的，误差会从输出层开始逐层往前计算，即loss反向传播，两层之间的误差变化率被称为“梯度”。每经过一次输入输出和反向传播，通常是输入一批(batch)数据以后，模型就会更新一次权重，更新的幅度取决于学习率大小；训练集所有样本都输入一次即可完成一轮(epoch)训练。为了将数据分批输入模型，需要借助数据加载器。它一般通过创建DataLoader()类的实例来实现，该类可将整个数据集dataset分成batch\_size大小的切片堆叠，因此它的实例相当于一个Tensor列表，每个元素可用next(iter())迭代器或者for…in…语法逐个取出，从而在循环中逐个输入模型。如此经过许多轮训练，模型即可收敛成你想要的“系统”。nn模型的前向传播和反向传播可以在cpu或gpu上进行，通过.to()方法可将模型放在相应设备。

## 基础CNN模型

在数据集质量环节使用的最基础的“已知可靠模型”为一个三层CNN网络，将200\*200 pixel \* 3通道图像转化为Tensor后直接输入。如表3-1所示，其第一层为一个2D卷积层，输入3通道，与图像维度一致；卷积核大小为3，步长为1，即一个3\*3的矩阵从输入的200\*200\*3矩阵前两个维度的左上角3\*3个元素开始，逐个与之相乘(卷积)，每乘完得到1个值，然后挪1个元素继续相乘，直到覆盖整个输入矩阵；最后将这些乘积按照权重拼接起来得到卷积层的输出，即特征图。每个卷积核都会产生一个特征图，输出16个通道，意味着该层有16个卷积核。第2、3层卷积层同理。卷积操作可以提取图像的局部特征，每个卷积层后面有通道注意力模块，包含两个普通线性层和激活函数，给卷积层之间又增加了可以训练的权重；reLU函数的作用是将负值部分截断为0，通常用于给模型增加非线性映射；池化层为二维最大池化，池化核为2，步长为2，即从输入数据的前两个维度的每2\*2个元素中取一个最大值(池化)，然后挪2个元素继续取值，因此池化后数据的前两个维度长度减半。经过三次卷积-通道注意力-reLU激活-最大池化后，tensor形状变为25\*25\*64，进入分类头。该分类头先将tensor展平为1维向量，然后经过线性映射、ReLU激活、Dropout随机丢弃后映射为26个类别标签，于是就得到了表3-3中的输入输出形状。

## ResNet18模型

普通CNN通常将每一层的输出直接作为下一层的输入，当层数较少时，理论上层数越多，模型能学习到更复杂的特征，从而表现更好，然而当网络达到一定深度(通常是20层左右)时，其性能反而会随着层数增加而退化，在前向/反向传播中容易发生意外，无法将特征信息/训练误差有效地传回输出/前层。ResNet(残差网络)通过引入残差连接来解决这些问题，它允许信息跳过一个或多个层，直接将输入层与更深的层连接，从而解决了梯度消失的问题。并且对于一个多层+跳跃连接的结构，即残差块而言，它的多层堆叠部分只需要学习输入与输出之间的差异，即残差函数，而非完整映射，因此加快了收敛速度。ResNet的出现极大提升了深度神经网络学习能力的上限，使得训练上百层甚至上千层的神经网络成为可能。

在数据集难度的多模型基准测试中使用到的ResNet18包括主干部分和4个含有残差块的层，输入Tensor先传入卷积层->归一化->ReLU激活->最大池化，即主干部分；然后进入残差块BasicBlock的实例。BasicBlock本身也是个nn模型，在\_make\_layer()里打包成主网络的子模块，即nn.Sequential对象，以便在layer1,2,3,4中调用；其参数\*layers为一个装有构造好的残差块的列表。在BasicBlock的前向传播中，用identity变量将输入Tensor暂存，然后与经过多层处理后的Tensor相加，得输出out，实现残差连接；downsample用来对齐identity与out维度。最后残差块的输出Tensor经过平均池化后进入分类头，得到ResNet18模型的最终输出。

## MobileNetV3模型

MobileNetV3Small是一种专门为移动设备和嵌入式设备等资源受限环境设计的轻量级卷积神经网络，它与ResNet都有残差连接，然而MobileNet的残差块是先通过卷积操作将输入Tensor扩展到更高的维度(通常是输入通道数的 6 倍左右)后，进行深度可分离卷积，然后再通过投影卷积将通道数压缩回输出通道数。这种结构被称为“倒残差块”。与标准卷积不同，深度分离卷积的每个卷积核只作用于一个输入通道，该方法能够显著降低模型的参数量和计算量，是MobileNet能够快速收敛的主要原因。

本测试使用到的MobileNetV3包括主干部分和11个倒残差块，输入Tensor先传入卷积层->归一化-> Hardswish激活，然后进入倒残差块，即InvertedResidual实例的堆叠层。在InvertedResidual的前向传播中，只有步长为1且输入输出维度相匹配时才会进行残差连接，返回输入Tensor与倒残差块中多层网络的输出之和，否则直接返回多层网络的输出。最后倒残差块的输出Tensor经过卷积->归一化-> Hardswish激活->平均池化后进入分类头，得到MobileNetV3模型的最终输出。

## EfficientNetB0模型

EfficientNetB0是一种在计算效率和模型性能之间取得良好平衡的深度学习模型，它与MobileNetV3都有倒残差块，然而EfficientNet中的残差连接有可能被DropConnect随机丢弃(不同于Dropout只丢弃单个神经元)，此时该模块的输出将仅剩下输入。同时它的参数针对复合缩放而特别设计，从而在不同EfficientNet版本中能够按深度-宽度-分辨率成比例扩张，使得EfficientNet系列相较于MobileNet更能灵活地调整其规模以适应不同的性能/资源要求。其核心结构作为倒残差模块的一个优化版本，被称为移动翻转瓶颈卷积(MBConv)模块。

本测试使用到的EfficientNetB0包括主干部分和18个MBConv模块，输入Tensor先传入卷积层->归一化->SiLU激活，然后进入MBConv模块，即MBConvBlock实例的堆叠层。在MBConvBlock的前向传播中，只有步长为1且输入输出维度相匹配时才会进行残差连接，否则直接返回多层网络的输出。当进行残差连接时，默认返回输入Tensor与多层网络的输出之和，然而多层网络的输出有20%概率被丢弃，替换成相同形状的0矩阵，此时将直接返回输入x。最后MBConv的输出Tensor经过卷积->归一化-> SiLU激活->平均池化后进入分类头，得到EfficientNetB0模型的最终输出。

## 模型保存、加载和特征提取

在训练nn模型时可以随时保存训练的结果，以便直接加载使用。由于不同数据集/训练方式/初始状态/随机条件/会得到不同的模型状态，通常这些状态并不容易达到，因此为了再现模型某一时刻的状态，需要将它保存在.pth文件中。Pytorch的torch.save()方法支持保存整个模型和仅保存参数字典，前者保存的是模型类的实例，是包含了模型结构在内的所有信息的完整“系统”，加载时需访问它的模型类，适用于需要精确再现的情况；后者保存的是模型中所有可学习参数(包括权重和偏置)的字典形式，加载时需要重新定义模型结构(即创建新的实例)，这使得保存下来的参数可以应用在不同结构的新模型上，并且在不同 PyTorch 版本之间具有更好的兼容性。

在数据集难度环节，为了使模型针对特定输入样本所产生的内部结果具有再现性，在训练程序中保存了某epoch时模型的参数字典state\_dict()，并在提取特征的程序中调用.load\_state\_dict()将其加载到与训练时结构完全相同的新建实例上。该实例的返回值为模型的最终输出，为了得到模型内部结果，即某一层的输出，需注册钩子，此时只要让模型发生一次前向传播，即可将该层的输入/输出取出来。在用于提取特征的脚本中对模型的目标层调用.register\_forward\_hook(output)，将模型设为eval()模式，输入所需数据，无需返回值即可自动将该层的输出拿到output里。