项目二：图像分类问题

史明磊 2020011245
清华大学自动化系

1 问题介绍

图像分类是计算机视觉中基本任务，也是图像检测、图像分割、物体跟踪、行为分析等其他高层视觉 任务的基础。本题尝试进行鸟类物种的分类，涉及的鸟类图像数据集包括 525 类，共计 84,635 张图像，我 们同时提供测试集 2625 张（每类 5 张图），以及验证集 2625 张（每类 5 张图），本数据质量较高，类别 样本均衡。图像储存在以类别命名的目录中，图像大小为 224*224。

图 1-1 数据集样本示例

本数据集属于细粒度分类问题。

2 实现方法

2.1 项目结构

- Project
    - dataset
    - output
    - plot
    - confusion_matrix_plot
    - main.py
    - network.py
    - vit.py
    - engine.py
    - losses.py
    - optimizer.py
    - plot.py

- cmd.sh

笔者将我的项目结构设计成了上面显示的文件树。每个文件夹或模块如下所示：

- **dataset**
  该文件夹储存了鸟类数据集。
- **output**
  该文件夹保存每个实验输出的一些日志信息和绘图，例如模型信息、超参数等。
- **plot**
  该文件夹存储绘制的图像信息。
- **confusion_matrix_plot**
  绘制混淆矩阵的代码包
- **main.py**

在 main.py 中，我们定义了模型、标准、优化器、训练过程、绘图函数和一些日志信息，是本次训练使用的主要 main 文件。该 main 文件把各个分离的模块如 network、engine、losses、optimizer 等集成起来进行使用，笔者推荐直接使用该文件进行训练和推理，具体使用方法见 README.md。

- **main_sml.py**

在 main_sml.py 中，我们定义了模型、标准、优化器、训练过程、绘图函数和一些日志信息，该 main 文件使用了各个分离的模块如 network、engine、losses、optimizer 等。

- **network.py**
  在 network.py 中，我们定义了 ResNet-sml 和 Vit 模型。
- **engine.py**

在 engine.py 中，我们定义了 train_one_epoch 函数和训练过程，并同时定义了 validate 和 test 函数。

- **losses.py**
  在 losses.py 中，我们定义了不同的标准函数，包括交叉熵损失和欧几里德损失。
- **optimizer.py**

在 optimizer.py 中，我们定义了不同的优化器，包括无动量 SGD、带动量 SGD、Adam、AdamW 和其他函数。

- **plot.py**
  在 plot.py 中，我们定义了绘制实验损失和准确率曲线的函数。
- **cmd.sh**
  这是实验中使用的命令指令。

2.2 模型设计

2.2.1 ResNet 解析

ResNet 是深度学习中非常著名的卷积神经网络模型，其名称来源于 “Residual Networks”（残差网络），由 Microsoft Research 在 2015 年提出，是 ImageNet 大规模视觉识别竞赛中最先进的模型之一。

ResNet 的创新思想在于将深层网络拆分为多个浅层网络，并在层之间添加了所谓的 shortcut（残差连接）。这些 shortcut 允许网络在拟合过程中学习直接拟合残差（即输入与网络输出之间的差异），从而克服了梯度消失的问题，使网络训练更容易收敛，同时提高了模

型的准确性和泛化能力。

ResNet 的一个核心模块是残差块（Residual Block）。

经典的残差块包含两个 3x3 卷积层，每层的输出经过 ReLU 激活函数，然后通过加法运算来获得块的输出。该块的输入信号被加到块的输出中，这样就可以在深层网络中进行信息传递和反向传播，减少了梯度消失的问题。该块的输出随后传递到下一层或下一个残差块中。

ResNet 依据网络深度的不同，分别被分为 ResNet-18、ResNet-34、ResNet-50、ResNet-101、ResNet-152 等不同版本，每个版本的网络深度和结构都不相同。最初的 ResNet 版本采用了传统的 VGG 网络结构以及一个全连接层，而后来的版本则引入了其他的特性，如全局平均池化层、批量标准化等，进一步提升了网络性能。本实验使用了 ResNet-18、ResNet-34 和笔者根据 ResNet 原理涉及的更小更少参数的 ResNet-sml 网络。

ResNet 模型的良好性能和可定制性，使之成为图像分类、物体检测、语义分割等计算机视觉任务模型的骨干网络（backbone），至今仍具有非常广泛的应用。

2.2.2 实际模型设计

- **ResNet-sml**

```python
class ResNet(nn.Module):
    def __init__(self, img_H=224, img_W=224, in_chans=3):
        super(ResNet, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(in_channels=in_chans, out_channels=128, kernel_size=5, stride=1, padding=2), nn.ReLU())
        self.maxpool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Sequential(nn.Conv2d(in_channels=128, out_channels=128, kernel_size=5, stride=1, padding=2), nn.ReLU())
        self.maxpool2 = nn.MaxPool2d(2, 2)
        # 由于MaxPooling降采样两次，特征图缩减为原图的 1/4 * 1/4
        self.fc1 = nn.Sequential(nn.Linear(in_features=128*int(img_H/4)*int(img_W/4), out_features=784), nn.Dropout(p=0.0), nn.ReLU())
        self.fc2 = nn.Sequential(nn.Linear(in_features=784, out_features=525), nn.Softmax(dim=1))
        self.weights_init_xavier()

    def weights_init_xavier(self):
        if isinstance(self, torch.nn.Linear) or isinstance(self, torch.nn.Conv2d):
            init.xavier_uniform_(self.weight)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool1(x)
        x = self.conv2(x) + x
        x = self.maxpool2(x)
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

图 2-1 笔者自建的超轻量级 ResNet 的结构图

- **ResNet-18 与 ResNet-34 的结构示意**

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 2-2 ResNet-18、ResNet-34、ResNet-50、ResNet-101、ResNet-152 的结构图

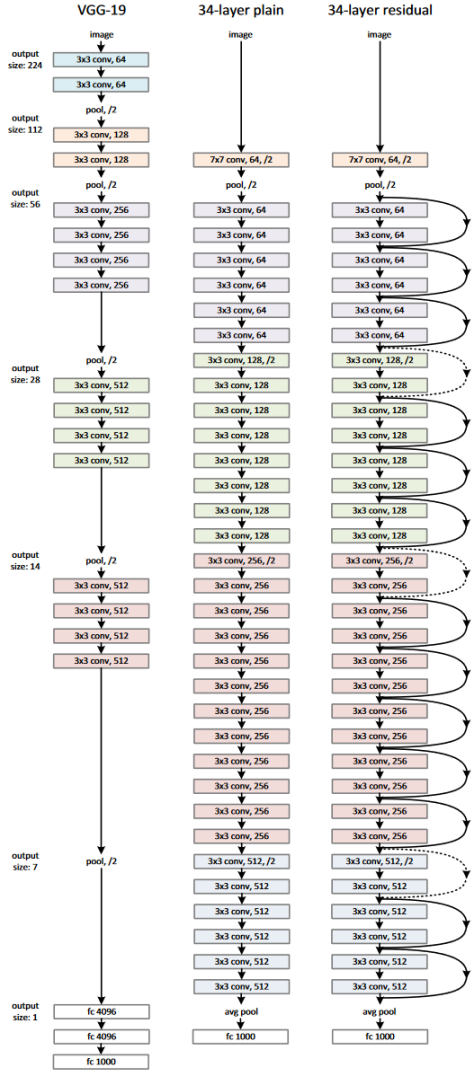

图 2-3 VGG-19、plain-34 的结构对比图

2.3 训练和测试

我们使用 cmd.sh 命令，并使用控制变量方法进行训练和测试实验

3 实验分析

3.1 超参数设置与训练结果

其中 A1~A6 是基础实验，而 B 和 C 组实验均为补充实验。B、C、D、E、F 组是在 A 组实验完成后经过分析得到改进后的参数设置结果。

表 3-1 超参数设置与训练结果

Tag	Model	Epochs	Learning rate	Weight decay	Dropout	Data Preprocess	Optimizer	Accuracy@1
A1	ResNet18	15	1e-01	1e-04	0.0	None	SGD	91.162%
A2	ResNet18	15	1e-01	0	0.0	None	SGD	90.133%
A3	ResNet18	15	1e-01	1e-04	0.1	None	SGD	90.514%
A4	ResNet18	15	1e-01	1e-04	0.2	None	SGD	88.457%
A5	ResNet18	15	1e-01	1e-04	0.0	None	AdamW	53.181%
A6	ResNet18	15	1e-01	0	0.0	None	AdamW	58.895%
B1	ResNet18	15	5e-02	1e-04	0.0	None	SGD	91.010%
B2	ResNet18	15	1e-02	1e-04	0.0	None	SGD	78.514%
B3	ResNet18	30	5e-02	1e-04	0.0	None	SGD	91.390%
B4	ResNet18	30	1e-02	1e-04	0.0	None	SGD	84.533%
B5	ResNet18	50	1e-01	1e-04	0.0	None	AdamW	72.876%
C3	ResNet34	15	1e-01	1e-04	0.0	None	SGD	90.362%
C4	ResNet34	30	1e-01	1e-04	0.0	None	SGD	90.705%
D1	ResNet18	15	1e-01	1e-04	0.0	AA1	SGD	80.648%
D2	ResNet18	15	1e-01	1e-04	0.0	AA2	SGD	86.286%
E1	ResNet18	15	1e-01	1e-04	0.0	删除 A*	SGD	91.670%
F1	ResNet-sml	50	1e-01	1e-04	0.0	None	SGD	1.752%

3.2 模型改进

3.2.1 输入数据处理

笔者设置了原始数据集形式（不进行处理）、数据扩增 1 形式、数据扩增 2、删减 A 开头的类别数据集形式。

（1）AA1 数据扩增 1 形式对应数据随机裁剪和水平翻转以及归一化处理：

```
train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))
```

图 3-1 数据扩增方法 1

（2）AA2 数据扩增 2 形式对应数据随机旋转和颜色抖动以及归一化处理：

```
train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomRotation(degrees=20),
        transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1),
        transforms.ToTensor(),
        normalize,
    ]))
```

图 3-2 数据扩增方法 2

（3）删除 A*指的是删除所有 A 开头的类别进行测试。

3.2.2 模型结构改进

模型从 ResNet-sml 到 ResNet18 再到 ResNet34,模型参数量依次扩大，对应于模型改进。多次使用 ResNet18 进行实验后，为了扩大参数量，提高泛化能力，笔者使用了 ResNet34 进行两组实验。实验过程发现，ResNet34 在 15 轮左右时训练集上的精度已经达到了 100%，而测试集仍为90%左右，模型出现过拟合问题，调整后发现模型精度与ResNet-18性能相当。而 ResNet-sml 由于模型较小，欠拟合问题严重。

3.2.3 模型参数压缩

模型从 ResNet34 到 ResNet18 到 ResNet-sml，模型参数量依次减小，对应于模型压缩。

3.3 评价指标

3.3.1 评价指标结果

此处选择基础实验 A 组进行指标评价；评价函数使用 sklearn 中的函数进行计算。

多分类问题中，我们通常会使用微平均和宏平均来评估多分类模型的性能表现。下面分别解释这两个指标：

微平均 (micro-averaging)：它考虑了所有样本的预测结果，然后计算精确度、召回率和 F1 分数（F1 分数是精确率和召回率的调和平均值）：

$$Precision_{micro} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} TP_i + FP_i}$$

$$Recall_{micro} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} TP_i + FN_i}$$

$$F1_{micro} = \frac{2 \times Precision_{micro} \times Recall_{micro}}{Precision_{micro} + Recall_{micro}}$$

宏平均 (macro-averaging)：它考虑了所有样本的预测结果，然后计算每个类别的精确度、召回率和 F1 分数（F1 分数是精确率和召回率的调和平均值）：

$$Precision_{macro} = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FP_i}$$

$$Recall_{macro} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i+FN_i}$$

$$F1_{macro} = \frac{2\times Precision_{macro}\times Recall_{macro}}{Precision_{macro}+Recall_{macro}}$$

具体代码实现如下：

```
micro_f1 = metrics.f1_score(labels, predicted, average='micro')
micro_precision = metrics.precision_score(labels, predicted, average='micro')
micro_recall_score = metrics.recall_score(labels, predicted, average='micro')


macro_f1 = metrics.f1_score(labels, predicted, average='macro')
macro_precision = metrics.precision_score(labels, predicted, average='macro')
macro_recall_score = metrics.recall_score(labels, predicted, average='macro')
acc = metrics.accuracy_score(labels, predicted)

confusion_mat = metrics.confusion_matrix(labels, predicted)
np.savetxt('matrix.txt',  confusion_mat)

# print("The test accuracy is {}.\n".format(batch_test_acc))
print("Micro_f1 {}, Micro_precision {}, Micro_recall_score {}, Acc {}".format(micro_f1, micro_precision, micro_recall_score, acc))
print("Macro_f1 {}, Macro_precision {}, Macro_recall_score {}, Acc {}".format(macro_f1, macro_precision, macro_recall_score, acc))
```

图 3-3 微观指标和宏观指标计算法

以下是 A 组实验的基本评价指标：

表 3-2 微观指标

Tag	F1-Score-Micro	Precision-Micro	Recall_Score-Micro
A1	0.912	0.911	0.912
A2	0.895	0.895	0.895
A3	0.905	0.905	0.905
A4	0.856	0.856	0.856
A5	0.532	0.531	0.532
A6	0.589	0.588	0.589

表 3-3 宏观指标

Tag	F1-Score-Macro	Precision-Macro	Recall_Score-Macro
A1	0.910	0.924	0.912
A2	0.893	0.910	0.895
A3	0.904	0.920	0.905
A4	0.854	0.888	0.856
A5	0.516	0.584	0.532
A6	0.575	0.640	0.589

3.3.2 指标含义解读

针对 6 个不同的标签 A1~A6。根据结果，先就微观结果进行以下的分析：

F1-Score-Micro：它是精度和召回率的加权平均值，因此可以对模型的总体性能进行评估。整个模型的 F1-Score 平均值为 0.76（即 5 个标签的 F1-Score 的平均值），说明模型的整体性能尚可。其中，标签 A1~A34 的 F1-Score 较高，说明模型对于这几个标签的识别准确度较高，标签 A5、A6 的 F1-Score 则较低，说明模型在这几个标签的识别上有较大的

提升空间。

Precision-Micro：用来评估模型的准确性标签 A1~A4 的 Precision-Micro 更高，这意味着模型在预测这几个标签为正例时准确性更高。但是 A5、A6 准确性较低。

Recall_Score-Micro：用来评估模型的召回率。标签 A1~A4 的 Recall_Score-Micro 更高，这同样意味着模型在这几个标签上的召回率更高。但是 A5、A6 召回率较低。

说明需要针对标实验 A5、A6 进行调整以提高准确率和召回率。

同理，我们对宏观结果进行分析能够得到同样的结论，说明多分类模型的宏观指标和微观指标均能够很好地评价模型的性能。

选取 A1 作为示例，绘制其混淆矩阵，由于类别众多，此处只抽样选择前 10 类别、第 420 到第 430 类别分别进行绘制混淆矩阵作为示例。

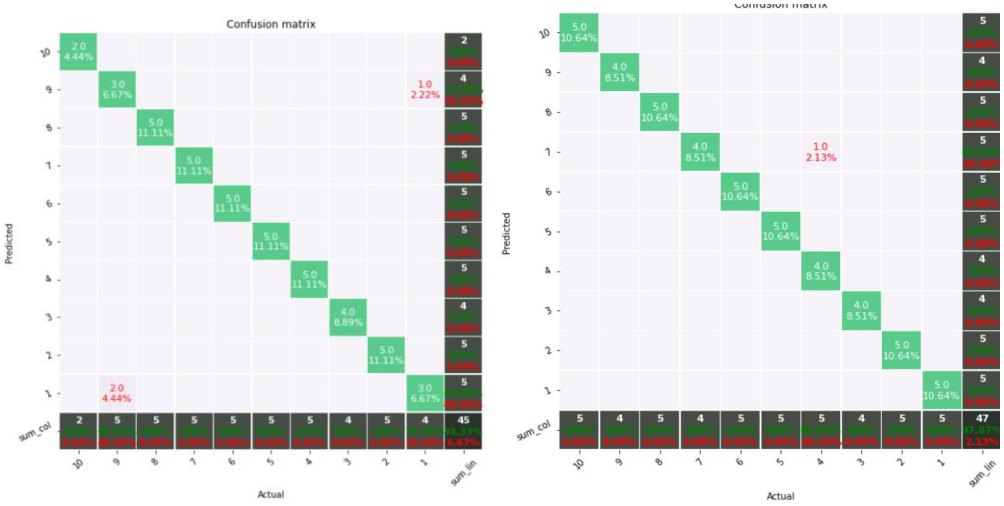

图 3-4 A1 前 10 类别、第 420 到第 430 类别混淆矩阵

观察混淆矩阵可以发现，A1 模型混淆矩阵对角线上基本上为该类别中所有目标值，非对角线上的元素很少，说明出错率低。实际上 A1 模型的准确率为 91.162%，误差相对较小

3.4 结果分析

固定参数和设置

- **数据集 Dataset**

  训练时使用提供的数据集中所有的训练数据进行训练。

  测试时使用提供的数据集中所有的验证数据进行测试。

- **批数量大小 Batch Size**

  实验将批数量大小固定为 256。

- **实验训练设备 Device**

  NVIDIA GeForce RTX 3090

- **损失函数 Criterion**

  由于我们面对的是分类问题，因此我们使用交叉熵损失函数 CrossEntropy Loss Function。

变化的参数和设置。

- **实验基础参数为：**

Epochs 15, Learning Rate 1e-01, Weight Decay 1e-04 Dropout 0.0 Optimizer SGD

- **最大轮次 Max Epoch**

在实验中，实验采用了 15 轮次为标准轮次。实验结果证明，在参数合适的情况下，15 轮时模型准确率已经能够达到 90%+。为了验证模型是否能够具有更强的泛化能力和准确率，也设置了 30 轮次的实验。

- **学习率 Learning Rate**

在实验中，我们设置了不同的学习率：1e-01、5e-02、1e-02。 我们可以发现，模型学习率为 0.1 时表现良好，其他参数合适的情况下，ResNet18 准确率已经可以达到 90%+，具体参见实验 Tag A1 A2 A3。

对比 A1 和 B1 发现 Learning Rate 为 1e-01 和 5e-02 时准确率差距不大（91.162% VS 91.010%），模型性能相当。

对比 A1 和 B2 发现 Learning Rate 为 1e-01 时准确率显著高于 1e-02 的情况，说明了 Learning Rate 过小时会使得模型收敛速度减慢，甚至无法获得最优解。为进一步验证说明，笔者设置了 Learning Rate 1e-02, 30 轮次的实验 C1，最终发现在 24 轮次时及之后训练集上 Acc@1 准确率已达到 100%，而测试集准确率稳定在 84.533%左右，说明模型过拟合严重，同时无法达到更好的泛化效果。

- **正则化系数 Weight Decay**

在实验中，我们设定了两种 Weight Decay：0 和 1e-04。

对比实验 A1 和 A2，能够发现 Weight Decay 为 1e-04 时能够略微减轻模型过拟合情况，使得模型准确率从 90.133%上升到了 91.162%。

- **失活率 Dropout**

在实验中，Dropout 添加在模型线性层之前，我们设定了三种 Dropout：0、0.1 和 0.2。

对比 A1 和 A3，能够发现 Dropout 为 0.1 时会使得模型略微欠拟合。

对比 A1 和 A4，能够发现 Dropout 为 0.2 时也会使得模型略微欠拟合。

分析结果说明了 Dropout 的设置在该参数条件下，不会提升模型性能，反而阻碍了模型拟合能力。

- **优化器 Optimizer**

在实验中，我们尝试使用 SGD 和 AdamW 作为优化器。

SGD 与 AdamW 组间对比 A1A2A3A4 与 A5A6，发现 15 轮次时通常 SGD 收敛更快，能够达到更高的准确率。AdamW 组内对比 A5A6 发现，不加 Weight decay 时，同轮次情况下准确率更高，说明此时模型仍处于欠拟合状态，指示我们要提高训练轮次，因此补充设置了 C2 实验。

- **模型结构**

在实验基础参数设置下，对比 A1、C3、F1 我们能够发现 ResNet18 和 ResNet34 性能显著好于 ResNet-sml。参见 E1，ResNet-sml 在经过 50 轮次训练后，准确率能够达到 1.752%，

比随机猜测 0.19%略高，但相比 ResNet18 和 ResNet34 而言，准确率差很多。

对比 A1、B3、C3、C4，得出结论在本数据集条件下，ResNet18 和 ResNet34 性能相当。

- **数据扩增**

  在实验基础参数设置下，对比 A1、D1、D2 发现数据扩增增大了训练识别难度，使得 D1、D2 模型准确率显著低于数据未扩增时的 A1 实验。同时通过训练过程也能够发现，尽管训练难度提高，但数据扩增相当于增大了数据集，可以减轻过拟合。但细粒度分类过程中，数据随机裁剪、颜色变换等问题可能会使得原有鸟类图片的关键信息丢失使得训练难度扩大。

- **参数调节经验**

  深度学习模型调参显著影响模型的性能和泛化能力，因此要根据场景选择合适参数。

  1. 选择适当的超参数，包括学习率、权重衰减，数量和大小的卷积核，池化大小等。这些超参数的选择对模型的性能和训练速度有很大影响。
  2. 使用合适的数据增强方法，例如镜像、随机裁剪、随机旋转等。越多的数据增强方法可以使模型能够学习更多的特征，以及更好的泛化能力。调整数据增强方法往往需要根据具体的数据集和问题进行实验。
  3. 对模型的结构进行适当的修改，并尝试不同的模型架构。例如，在卷积神经网络中添加更多的卷积层、全连接层、调整池化层的大小等。
  4. 打开或关闭 BatchNorm、Dropout 等技巧。这些技巧一般能够显著提高模型的性能，但打开或关闭它们会对训练和验证结果产生巨大的影响。因此需要经过实验调整。
  5. 使用合适的优化器，例如 SGD 、AdamW 等。初始学习率和学习率调整策略也非常重要，可以使用学习率衰减或者学习率随时间下降的策略进行调整。

  虽然调参是一种经验性的过程，但仍有一些基本原则需要遵循。最好的方法是尝试尽可能多的参数组合，并记录每次尝试的结果。根据实验结果，调整比重，并进行调整，从而找到最佳模型。

4 可解释性问题探索

4.1 CAM 可视化

CAM 可视化结果基于 A1 实验的最优结果。

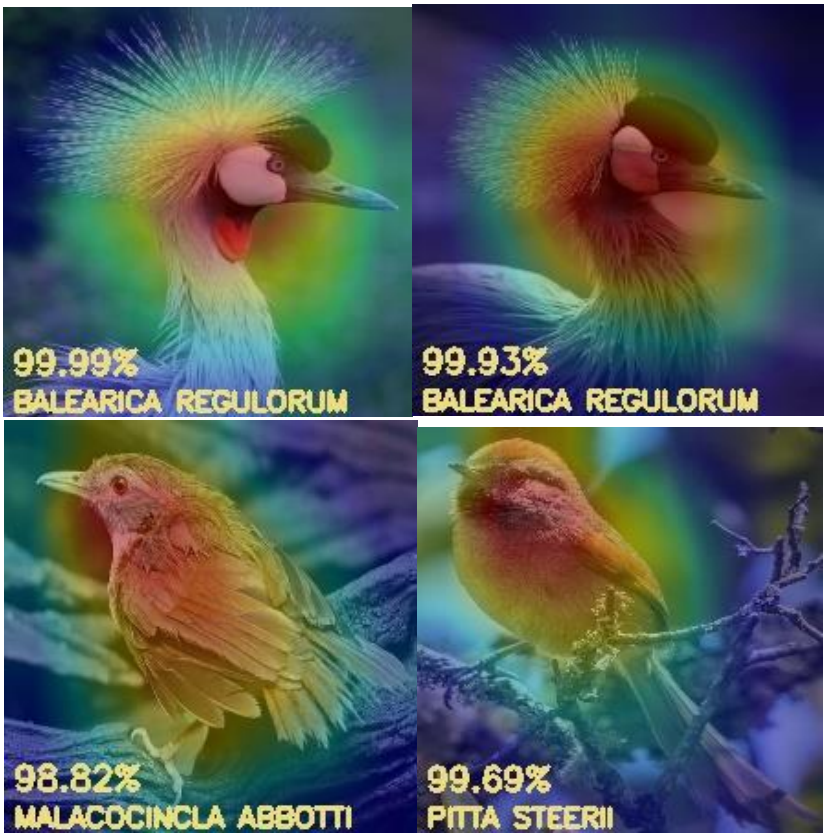

图 4-4 A1 随机选取的四张测试集鸟类的 CAM 图

4.1.1 CAM 原理解析

类激活图可以显示模型在训练过程中，权重或重心在何处、如何转移，分类模型是根据哪一部分的特征进行判别的。

CAM 是模仿人类识别物体的过程，随着模型的迭代，找到相关任务的关键部位。CAM 列激活图由两部分加权构成：原图+特征图。其中特征图是通过：最后全连接层的参数（权重矩阵 W）与最后输出的特征图集合对应相乘再相加（重叠）而形成，即显示模型是依据哪些特征图进行决策分类的。[1]

4.1.2 模型决策过程解释

CAM 图可以帮助分析网络的预测结果和注意力区域，从而可以解释模型的决策过程。例如，CAM 图可以揭示模型在图像分类时使用的特定视觉特征和重要区域，可以指导我们对收集更好的数据或进行更优的数据增强来优化模型性能。

从 CAM 图中可以看出，识别鸟类数据时，图形会优先关注鸟类的头部、和躯干部。这和人类理解图片的行为是相同的，对鸟类进行识别时，人类也会首先关注鸟类的头部和躯干部的信息，头部具有鸟喙、鸟冠、眼睛和形状等信息；躯干具有羽毛形状、颜色、躯干形状等信息，这些信息都能够帮助我们理解鸟类并进行分类。

同时 CAM 图可以在计算机视觉领域中对分类结果进行解释和可视化，使我们更好地理解深度学习模型的决策过程。

[1] 类激活图（CAM）原理详解

4.2 类别删减测试

笔者删除了所有 A 开头的类别进行测试。参见实验 E1 准确率为 91.670%，在删减掉 52 个 A 开头的类别后，发现准确率相比较 A1 而言并没有显著提升或下降。说明模型没有类别偏好性，也说明了不同类别难度相当，数据集质量很高。

4.3 数据集类别均衡度测试

修改数据集类别的均衡度是解决分类任务中类别不平衡的一种方法。在进行分类任务时，通常会出现某些类别的样本数远远大于其他类别的情况。这样容易导致一些类别被模型误分类或者被忽略，从而影响模型的性能和泛化能力。例如，如果存在一个二分类数据集，正类样本数为 1000 个，负类样本数量为 100 个，这就是一个不平衡的数据集。在这种情况下，模型可能会更偏向于预测正类，而忽略负类。因此，我们可以通过调整数据集的均衡度来解决这个问题。可以采用以下两种方法：

1.过采样方法。在过采样方法中，我们可以通过增加少量类别的样本来平衡样本数量。具体来说，复制一些少量类别的样本，或者根据少量类别的分布进行随机生成一些新的样本。一般情况下，这些生成的样本会基于现有的数据样本进行，例如通过旋转、缩放或者添加高斯噪声来增强样本。

2.欠采样方法。欠采样是另一种减少不均衡数据集的方法，其中从多数类样本中随机选择一些样本。这可以通过删除大量类别的样本，以便达到与其他类别相同的样本数量，并使得样本分布更加均衡。在欠采样中，通常随机删除大量类别的样本，以便达到与其他类别相同的样本数量，并使得样本分布更加均衡。

本实验中，训练集中

$$\frac{84635}{525} \approx 161$$

平均每类中有 161 张鸟类图片，而实际上，绝大部分类别鸟数量在 100~200 之间浮动，少数数据多于 200 张。说明该数据集是一个数据集较为均衡的情形，是一个优质数据集。笔者经过调整后进行实验，发现该数据集数据均衡度略微改变时不会影响整体训练结果和性能。

5 结论

实验中，我们使用了三种模型与多种参数设置，最终发现该数据集下模型过小不能够取得很好的细粒度分类效果、模型过大会使得参数冗余或是过拟合且并不能取得很好的提升效果，因而选择一个合适大小的模型设置合适的参数尤为重要。本实验中 ResNet18 的基础参数设置就是一个很好的 baseline。

为了缓解过拟合问题，我们可以采用数据扩增、增加 Weight Decay、Dropout、减少模型参数、设置早停等方法。

为了缓解欠拟合问题，我们可以采用增加模型参数、改进模型结构、增加训练轮次等方

法。

为了探索模型的可解释性问题，我们可以采用 CAM 图的方法进行可视化理解。

同时本实验也设置了类别删减测试、数据集类别均衡度测试等实验。

通过这次实验，笔者掌握了基本的图片分类训练方法、模型构建、改进策略、评价体系和可解释性分析，增强了对计算机视觉的理解，同时也对计算机视觉的前沿内容进行了阅读和理解，如 ResNet、Vit 和 CAM 方法等。

6 致谢

感谢这次作业提供的思路和数据集！通过练习，我熟练使用了 pytorch 实现 CNN 和 ResNet 模型，也阅读了若干篇基础模型的论文，同时学会了如何构建使用结构化文件组织的项目。

7 参考文献

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[3] Zhou, Bolei, et al. “Learning deep features for discriminative localization.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.