

《人工智能原理》项目 2 报告
ButterflyNet 图像分类模型的设计与优化

目录

1 前言	4
1.1 任务描述	4
1.2 实验任务	4
2 工程架构设计	4
2.1 配置驱动的实验管理	5
2.2 模型工厂与结构解耦	5
2.3 训练引擎与回调机制	5
2.4 数据管线与可视化分析模块	5
2.5 复现性与日志管理	6
3 数据预处理	6
3.1 数据集划分	6
3.2 数据增强	6
3.3 效果对比	6
4 模型迭代与演进	7
4.1 原型验证: ButterflyMyNet	7
4.1.1 结构	7
4.1.2 测试	8
4.2 深度与稳定性增强: ButterflyMyNetPro	9
4.2.1 测试	10
4.3 标准化与模块化——ButterflyVGG	11
4.3.1 结构	11
4.3.2 测试	12
4.4 残差学习——ButterflyResNet	13
4.4.1 结构	13
4.4.2 测试	15
4.5 模型压缩与轻量化——ButterflyResNet (Tiny)	16
4.5.1 结构	16
4.5.2 测试	17
5 训练策略	19
6 模型测试与性能评估	19
6.1 测试方法与评估指标	19
6.2 测试结果与解释	19
6.2.1 训练过程分析	19
6.2.2 测试集性能评估	20
6.2.3 混淆矩阵分析	20

7 模型可解释性分析 (Grad-CAM)	20
7.1 算法原理	20
7.2 实现流程	21
7.3 结果展示与分析	21
8 讨论与展望	23
8.1 遇到的问题与解决方案	23
8.2 不足与展望	23
8.3 总结	24

摘要

本项目旨在解决包含 50 个类别、4479 张图像的蝴蝶物种分类问题。项目基于 PyTorch 深度学习框架，设计并实现了一个完整的深度学习工程系统，完成了全部必做和绝大多数选做任务。首先构建了基础的浅层卷积网络；随后增加深度、引入批归一化和 Dropout 设计了增强版模型；进而引入 VGG 模块化设计思想构建了深层网络，并复现了基于残差学习的 ResNet 架构。采用了准确率、精确率、召回率及 F1-Score 等多维度指标，还利用 Grad-CAM 技术对模型进行了可视化解释性分析。最终在测试集达到准确率 89% 左右，且可视化结果验证了模型决策依据的合理性。

关键词： 图像分类；卷积神经网络；ResNet；Grad-CAM；模型可解释性

1 前言

1.1 任务描述

本次大作业是基于蝴蝶图像数据集（包括 50 类海洋生物，共4479张图像）的蝴蝶图片多分类任务。

本次任务中使用 Pytorch 框架设计训练流程，使用 wandb 平台记录实验数据，尝试：自己搭建简单的卷积神经网络，改进简单神经网络，复现经典卷积神经网络等多种方式完成本次分类任务。还尝试了不同的学习率策略、超参数选择、优化器选择等训练技巧。

1.2 实验任务

任务1

- 设计一个卷积神经网络模型来解决该分类问题；见 Sec 4.1, 4.2
- 采用一个深度学习框架来实现设计的模型；见 Sec 2
- 提出方法并测试模型；见 Sec 6.1
- 提出指标，解释并评价模型。见 Sec 6.2

任务2（选）

- 输入数据的处理；见 Sec 3
- 训练中各种超参数的设置；见 Sec 5
- 优化器的选择；见 Sec 5
- 模型结构的改进；见 Sec 4.3, 4.4
- 设计方法对模型参数进行压缩；见 Sec 4.5
- 设计可视化方法，验证模型的可靠性，并尝试对模型决策过程进行解释。见 Sec 7

2 工程架构设计

为保证代码的可维护性、扩展性以及实验结果的可复现性，本项目采用了配置驱动 + 模块化 + 工厂模式的设计思路。

2.1 配置驱动的实验管理

项目的所有超参数（学习率、Batch Size、网络结构参数等）和路径配置均采用 YAML 文件进行集中管理，实现了代码与参数的分离。

- 统一配置入口：通过 `config/base_config.yaml` 提供默认设置，包括数据路径、数据增强策略、模型结构、优化器与学习率调度器、训练轮数、早停策略、日志与可视化等。
- 增量实验配置：在 `config/experiments/` 目录下为不同实验维护差异配置，运行时由 `config_parser.py` 负责依次加载基础配置与实验配置和命令行 `key=value` 覆盖。
- 强校验机制：配置解析器对输入尺寸、数据增强参数、训练/评估超参数进行类型与范围检查，避免由于配置错误导致的隐性 bug。

2.2 模型工厂与结构解耦

在模型层面采用模型工厂模式，将具体网络结构与训练流程解耦。

- 模型注册与创建：在 `model_factory.py` 中维护全局模型注册表，利用装饰器 `@register_model` 将不同模型统一注册；通过 `get_model` 方法动态实例化模型。
- 结构参数配置化：以 `vgg_net.py` 为例，通过传入 `kwargs` 中的卷积层通道数、池化层位置、残差块堆叠深度等结构信息，通过配置驱动快速调整网络结构。
- 归一化与激活工厂：通过 `get_normalization` 和内部激活工厂函数，根据配置选择 BatchNorm、LayerNorm、GroupNorm 或无归一化，以及 ReLU、GELU 等激活函数，便于在不同正则化与激活组合之间切换。

2.3 训练引擎与回调机制

核心训练流程由 `Trainer` 类统一管理，并通过回调机制增强可扩展性。

- 训练主循环：`Trainer` 负责构建优化器、学习率调度器、损失函数和数据加载器，并在 `fit()` 中执行标准的 epoch/batch 训练与验证流程，同时累积指标并记录历史曲线。
- 回调机制解耦附加功能：在 `callbacks.py` 中定义了若干回调类，包括检查点保存、早停、学习率调度等；在训练主循环中触发回调钩子 `on_epoch_end`，实现功能的模块化扩展。

2.4 数据管线与可视化分析模块

数据加载和结果分析部分也进行模块化设计，以支持更丰富的数据预处理和后续分析需求。

- 数据管线模块化：`transforms.py` 按配置动态构建训练/验证/测试阶段的 `torchvision.transforms` 数据预处理与按需开启的增强流水线。
- 评估与分析解耦：`Evaluator` 负责在测试集上计算最终指标、生成分类报告和混淆矩阵，并调用 Grad-CAM 工具生成热力图，同时进行单图与按类别的可视化分析。

2.5 复现性与日志管理

在工具层面的设计保证实验可复现性和调试便利性。

- 随机性控制：`set_seed` 函数，在实验开始前统一设置 Python `random`、NumPy、PyTorch CPU 与 CUDA 的随机种子。
- 统一日志系统：`logger.py` 支持控制台与文件双通道输出，所有子模块共用同一个日志入口，便于集中排查问题。
- 权重与结果管理：`CheckpointManager` 负责按指标排序保存模型检查点，并在训练结束后通过导出仅包含模型权重的 `best_model.pth`，方便部署和后续评估。

3 数据预处理

3.1 数据集划分

采用独立测试集的测试方法，将原始数据集按照分层抽样的方式划分为训练集、验证集和测试集，比例设定为 70% : 15% : 15%，分别有 3135、672、672 张图片。分层抽样确保了每个子集中各类别的分布与总体一致，避免了因某些稀有类别在验证集中缺失而导致评估偏差。实验时在训练集上进行训练，每个 epoch 后在验证集上验证模型性能决定是否保留；实验结束后，在测试集上进行最终的评价和横向比较。

3.2 数据增强

为提升模型对不同图像变化的泛化能力，训练集采用了以下数据增强策略，相关参数通过配置文件进行调整：

- 随机放缩和裁剪：`RandomResizedCrop` 模拟不同距离和视角下的蝴蝶图像。
- 几何变换：随机水平翻转（`RandomHorizontalFlip`），模拟蝴蝶的不同朝向；随机旋转（`RandomRotation`），增强对角度变化的鲁棒性。
- 颜色抖动：`ColorJitter` 随机调整亮度、对比度和饱和度，模拟不同光照条件。
- 归一化：使用ImageNet数据集的均值和标准差对图像进行标准化处理，加速模型收敛。

最后所有图像被调整为 224×224 的输入尺寸。

3.3 效果对比

我用模型 ButterflyVGG 进行对比实验：分别使用经过数据增强和未经过数据增强的训练集进行训练，结果表明经过数据增强的模型在正确率等评价指标上均有更好的表现。从曲线可以看出，尽管最终都过拟合，但数据增强后的模型过拟合程度明显更低。这证实了数据增强的有效性。

表 1: 数据增强前后对比

数据增强	Acc	Precision	Recall	F1
启用	0.8698	0.8803	0.8685	0.8699
未启用	0.7991	0.8105	0.7902	0.7987

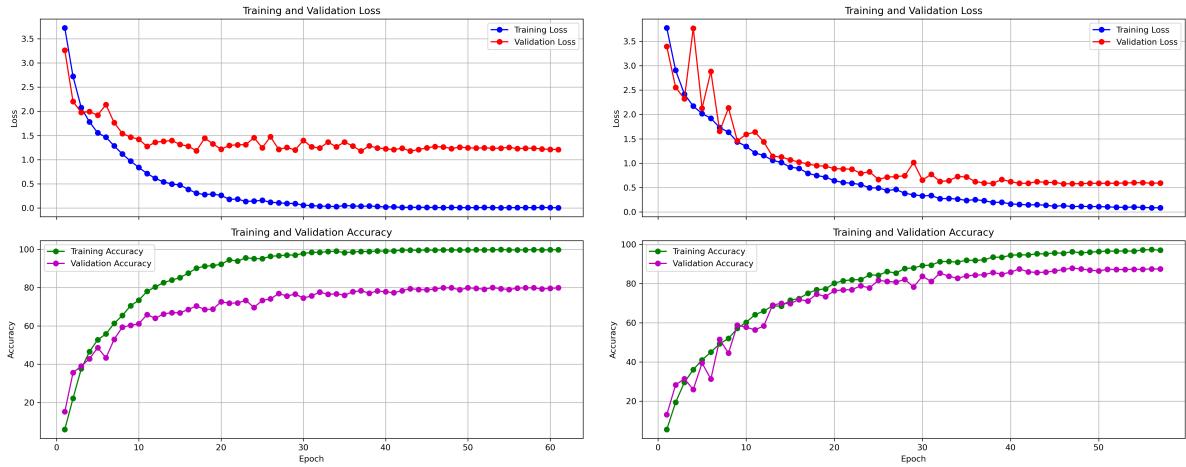


图 1: 数据增强前后训练曲线对比 (左: 未增强, 右: 增强)

4 模型迭代与演进

在探究蝴蝶图像分类任务的过程中，本项目并未直接采用单一模型，而是遵循由简入繁、由浅入深的原则，设计并实现了五个阶段的模型架构。

4.1 原型验证: ButterflyMyNet

4.1.1 结构

ButterflyMyNet 是一个极简的卷积神经网络，用于建立最基础的性能基线。它仅使用两个卷积层提取特征，并通过全连接层进行分类。

表 2: ButterflyMyNet 模型结构详解

层次	模块结构	输入尺寸 ($C \times H \times W$)	输出尺寸 ($C \times H \times W$)	核心目的
Input	-	$3 \times 224 \times 224$	$3 \times 224 \times 224$	图像输入
Conv Block 1	Conv2d(5×5) → MaxPool(2×2)	$3 \times 224 \times 224$	$10 \times 110 \times 110$	初步特征提取与降维
Conv Block 2	Conv2d(5×5) → MaxPool(2×2) → ReLU	$10 \times 110 \times 110$	$20 \times 53 \times 53$	提取局部特征
Flatten	Flatten	$20 \times 53 \times 53$	56180	特征展平
Classifier	Linear(56180 → 128) → ReLU	56180	128	映射到隐层空间
Output	Linear(128 → 50)	128	50	输出类别 Logits

- 模型结构: 非常精简，仅包含两个特征提取块。
 1. Layer 1: 5×5 卷积层 (输出通道10) → 2×2 最大池化。
 2. Layer 2: 5×5 卷积层 (输出通道20) → 2×2 最大池化 → ReLU激活。

3. 分类头: 展平层 (Flatten) → 全连接层 (128隐层单元) → ReLU → 全连接层 (输出50类)。

- 设计特点: 采用了较大的卷积核 (5×5) 以在浅层网络中获得较大的感受野。
- 局限性: 网络过浅, 特征提取能力极其有限; 未引入归一化和正则化手段, 训练容易震荡且极易过拟合。

4.1.2 测试

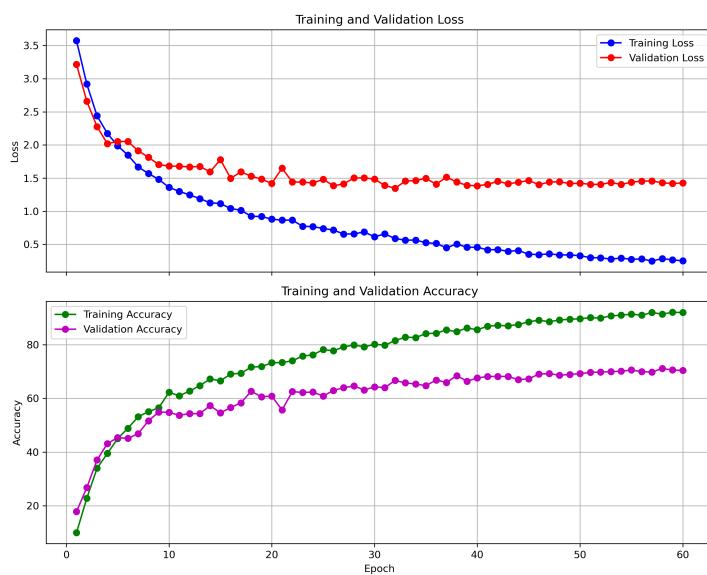


图 2: 训练曲线

```
{  
    "accuracy_top1": 0.7091,  
    "accuracy_top5": 0.91,  
    "precision_macro": 0.7213,  
    "recall_macro": 0.706,  
    "f1_score_macro": 0.7072,  
    "precision_micro": 0.7091,  
    "recall_micro": 0.7091,  
    "f1_score_micro": 0.7091,  
    "precision_weighted": 0.7184,  
    "recall_weighted": 0.7091,  
    "f1_score_weighted": 0.7079  
}
```

图 3: 测试集结果

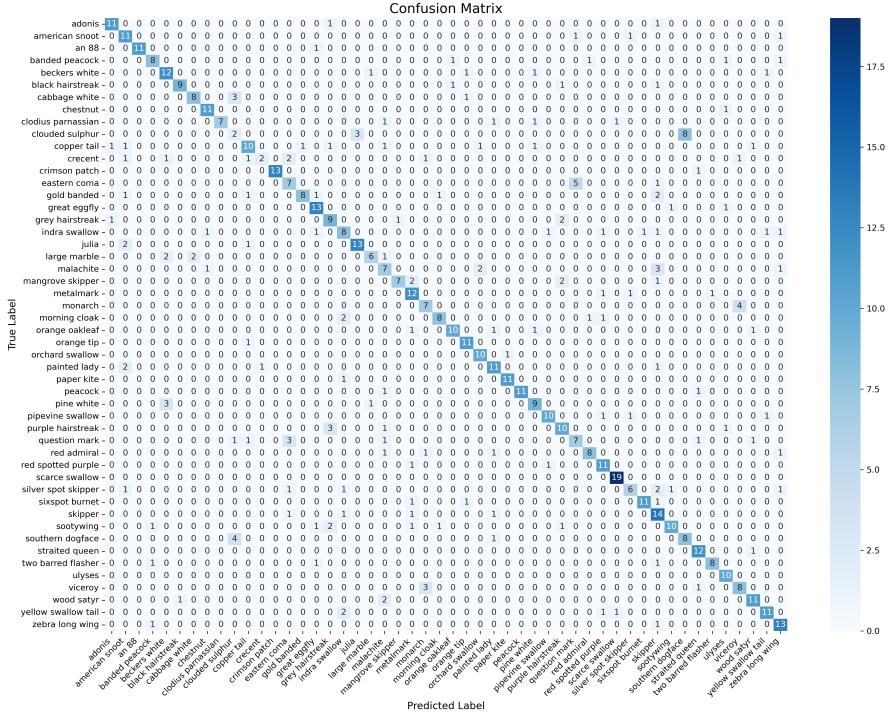


图 4: 混淆矩阵

上述结果表明: MyNet 结构过于简单, 即使在训练集上也没有很好地收敛, 在验证集上更是早早出现了过拟合的情况,

4.2 深度与稳定性增强: ButterflyMyNetPro

针对基础版模型特征提取能力不足的问题, 设计了增强版模型 ButterflyMyNetPro, 主要加深网络(4个卷积阶段)、引入批归一化和多尺度卷积核来提升性能。

表 3: ButterflyMyNetPro 模型结构详解

层次	模块结构	输入尺寸	输出尺寸	核心目的
Block 1	Conv(7×7 , s=2) \rightarrow BN \rightarrow ReLU \rightarrow MaxPool	$3 \times 224 \times 224$	$32 \times 56 \times 56$	大卷积核快速降维
Block 2	Conv(5×5) \rightarrow BN \rightarrow ReLU \rightarrow MaxPool	$32 \times 56 \times 56$	$64 \times 28 \times 28$	中层特征提取
Block 3	Conv(3×3) \rightarrow BN \rightarrow ReLU	$64 \times 28 \times 28$	$128 \times 28 \times 28$	增加非线性深度
Block 4	Conv(3×3) \rightarrow BN \rightarrow ReLU \rightarrow MaxPool	$128 \times 28 \times 28$	$128 \times 14 \times 14$	深层抽象特征提取
Classifier	Dropout \rightarrow Linear \rightarrow ReLU \rightarrow Dropout \rightarrow Linear	25088	50	正则化与分类

- 结构改进:

1. 加深网络: 将卷积层数量增加至4个阶段, 通道数逐层递增 ($32 \rightarrow 64 \rightarrow 128 \rightarrow 128$)。
2. 多尺度卷积核: 第一层采用 7×7 大卷积核配合步长2快速降低分辨率; 中间层混合使用 5×5 和 3×3 卷积核, 以丰富特征提取的尺度。
3. 引入批归一化: 在每个卷积层后、激活函数前加入BN层, 规范化特征分布, 加速收敛。
4. 正则化策略: 在分类头中引入了双重 Dropout, 比例分别为 0.3 和 0.2, 以缓解全连接层参数过多导致的过拟合问题。

- 效果：相比基础版，Pro 版在收敛速度和泛化能力上均有显著提升，但手动堆叠不同尺寸卷积核的方式缺乏系统性设计，难以进一步扩展深度；同时模型参数量大大增加，训练时间也相应增长。

4.2.1 测试

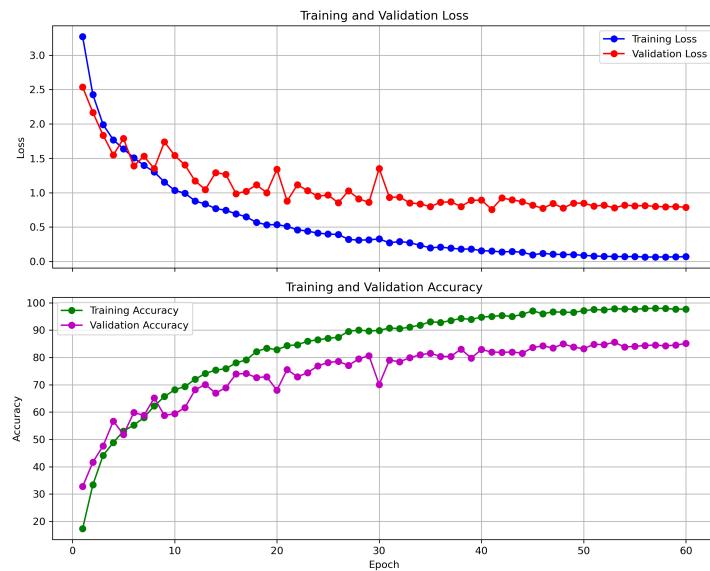


图 5: 训练曲线

```
{
    "accuracy_top1": 0.8557,
    "accuracy_top5": 0.9673,
    "precision_macro": 0.8634,
    "recall_macro": 0.8537,
    "f1_score_macro": 0.8554,
    "precision_micro": 0.8557,
    "recall_micro": 0.8557,
    "f1_score_micro": 0.8557,
    "precision_weighted": 0.8624,
    "recall_weighted": 0.8557,
    "f1_score_weighted": 0.856
}
```

图 6: 测试集结果

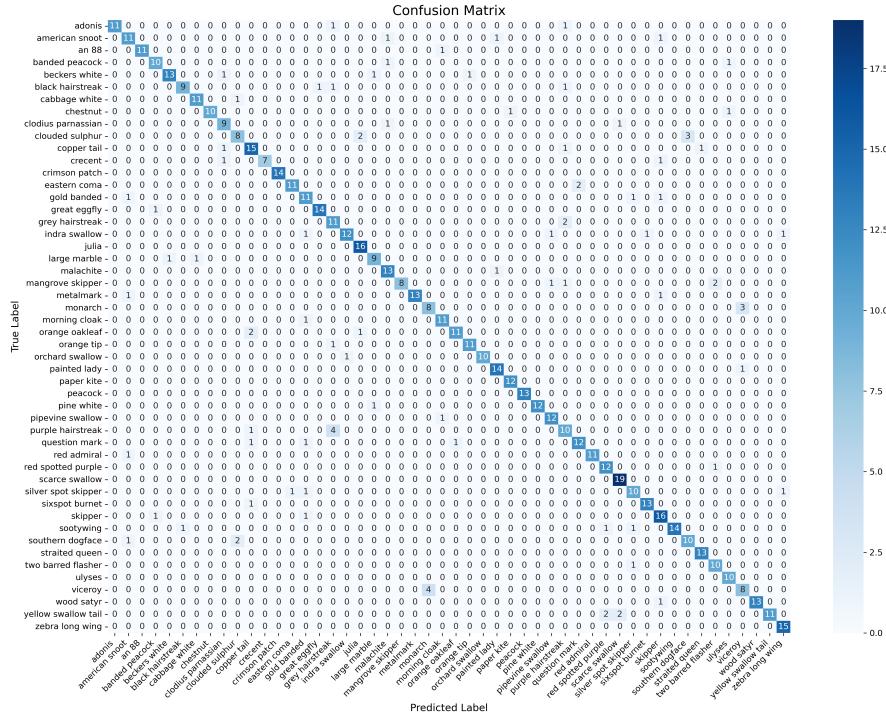


图 7: 混淆矩阵

上述结果表明，MyNetPro 在网络结构上复杂度提升明显，凭借更多的卷积层、BN 层和 Dropout 有更强的分辨能力，获得更好的表现，实现了任务要求的模型改进与提升。但我还想向 90% 的准确率努力，因此接下来又尝试复现两种经典模型。

4.3 标准化与模块化——ButterflyVGG

4.3.1 结构

VGG [1] 是 ILSVRC 2014 的相关工作，证明了增加网络的深度能够在一定程度上影响网络最终的性能。和之前的 SOTA 模型 AlexNet 相比，VGG 采用连续的几个 3×3 的卷积核代替 AlexNet 中的较大卷积核 ($11 \times 11, 7 \times 7, 5 \times 5$)。对于给定的感受野，采用堆积的小卷积核优于采用大的卷积核，因为多层非线性层可以增加网络深度来保证学习更复杂的模式，而且参数更少。

本实验借鉴 VGGNet 的设计思想，采用统一的 3×3 卷积核和模块化堆叠，引入自适应池化以适应不同尺寸输入，实现了 VGG11_small 和 VGG16_small 两种变体。本实验主要使用 VGG11_small 进行测试。

表 4: ButterflyVGG (VGG11_small) 模型结构详解

层次	模块结构	输入尺寸	输出尺寸	核心目的
Block 1	Conv(3×3 , 32) → BN → ReLU → MaxPool	$3 \times 224 \times 224$	$32 \times 112 \times 112$	浅层纹理特征
Block 2	Conv(3×3 , 64) → BN → ReLU → MaxPool	$32 \times 112 \times 112$	$64 \times 56 \times 56$	组合特征
Block 3	[Conv(3×3 , 128) → BN → ReLU] × 2 → MaxPool	$64 \times 56 \times 56$	$128 \times 28 \times 28$	中层语义特征
Block 4	[Conv(3×3 , 256) → BN → ReLU] × 2 → MaxPool	$128 \times 28 \times 28$	$256 \times 14 \times 14$	高层语义特征
Block 5	[Conv(3×3 , 256) → BN → ReLU] × 2 → MaxPool	$256 \times 14 \times 14$	$256 \times 7 \times 7$	进一步抽象
AvgPool	AdaptiveAvgPool2d(7, 7)	$256 \times 7 \times 7$	$256 \times 7 \times 7$	统一特征图尺寸
Classifier	(Linear → ReLU → Dropout) × 2 → Linear	12544	50	稠密分类层

- 核心设计思想:

- 统一卷积核: 全部替换为 3×3 小卷积核。根据感受野原理, 堆叠两个 3×3 卷积层的感受野等同于一个 5×5 卷积, 但参数量更少且非线性更强。
 - 模块化堆叠: 采用 Conv-Norm-Activation 的标准序列构建特征层。本实验主要使用了 VGG11_small 配置 (通道数序列: $32 \rightarrow 64 \rightarrow 128 \rightarrow 256$)。
 - 自适应池化: 在特征提取器末端引入 AdaptiveAvgPool2d((7, 7)), 无论输入图像尺寸如何变化, 均能输出固定尺寸的特征图, 增强了模型对输入尺寸的鲁棒性。
- 局限性: 全连接层参数量依然较大, 且随着网络进一步加深, 容易出现梯度消失问题。

4.3.2 测试

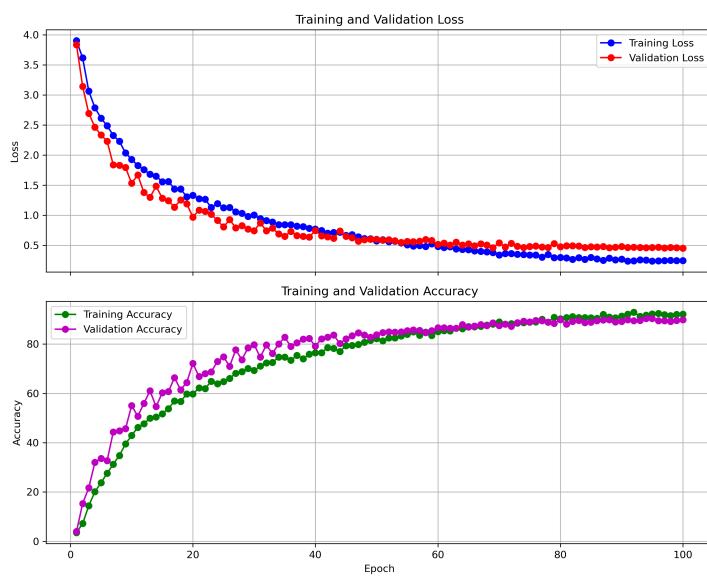


图 8: 训练曲线

```
{
    "accuracy_top1": 0.8981,
    "accuracy_top5": 0.9777,
    "precision_macro": 0.9052,
    "recall_macro": 0.8984,
    "f1_score_macro": 0.8975,
    "precision_micro": 0.8981,
    "recall_micro": 0.8981,
    "f1_score_micro": 0.8981,
    "precision_weighted": 0.906,
    "recall_weighted": 0.8981,
    "f1_score_weighted": 0.8977
}
```

图 9: 测试集结果

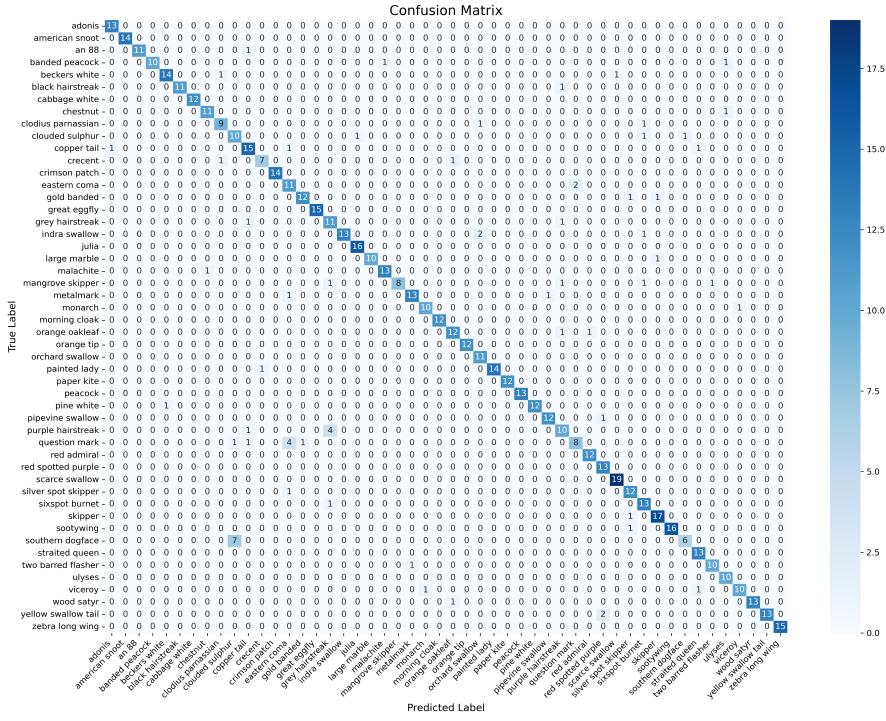


图 10: 混淆矩阵

上述结果表明，相比于之前的网络，VGG 模型的泛化性能更好，验证集和训练集的成功率已经十分接近，在 100 个 epoch 的训练后最终达到了 89% 的正确率。但 Value Loss 曲线表明仍然出现了过拟合问题，猜测是图片数量过少导致的，即使已经增强了数据，但是 4.5k 张图片还是过于困难。

4.4 残差学习——ButterflyResNet

4.4.1 结构

ResNet [2] 是由 Kaiming He 于 2016 年提出的残差网络结构，其重要思想残差缓解了深层网络的梯度消失和爆炸问题，同时由于学习的映射关系近似于恒等映射，使学习过程更加简单。

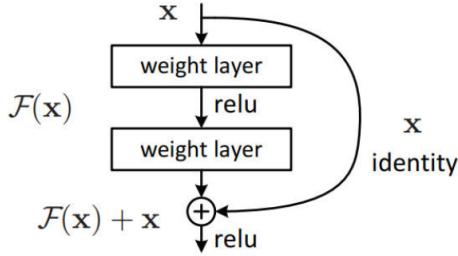


图 11: 残差块结构

本实验也尝试复现了 ResNet18_small 配置，其中通道数较标准 ResNet18 减半以适应数据集规模，并使用全局平均池化大幅减少参数。

表 5: ButterflyResNet (ResNet18_small) 模型结构详解

层次	模块结构	输入尺寸	输出尺寸	核心目的
Stem	Conv(7×7 , s=2) → BN → ReLU → MaxPool	$3 \times 224 \times 224$	$32 \times 56 \times 56$	快速降采样
Layer 1	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$ (BasicBlock)	$32 \times 56 \times 56$	$32 \times 56 \times 56$	残差特征提取
Layer 2	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$ (BasicBlock, s=2)	$32 \times 56 \times 56$	$64 \times 28 \times 28$	降维与特征加倍
Layer 3	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$ (BasicBlock, s=2)	$64 \times 28 \times 28$	$128 \times 14 \times 14$	深层特征提取
Layer 4	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$ (BasicBlock, s=2)	$128 \times 14 \times 14$	$256 \times 7 \times 7$	高维语义抽象
GAP Classifier	AdaptiveAvgPool2d(1, 1) Flatten → Dropout → Linear	$256 \times 7 \times 7$ 256	$256 \times 1 \times 1$ 50	全局空间信息聚合 轻量级分类

- 残差块：引入跨层连接，将输入 x 直接加到卷积输出上：

$$y = \sigma(F(x, \{W_i\}) + x)$$

其中 $F(x)$ 为残差映射（包含两个 3×3 卷积）， x 为恒等映射。这使得梯度能够无损地反向传播。

- 架构细节：

1. Stem层： 7×7 卷积 ($stride = 2$) + BN + ReLU + MaxPool，快速下采样。
2. 残差阶段：包含 4 个 Stage，每个 Stage 由 2 个 BasicBlock 组成。通道数依次为 32, 64, 128, 256。在 Stage 2, 3, 4 的第一个 Block 通过 $stride = 2$ 进行下采样。
3. 全局平均池化：不同于 VGG，在末端使用全局平均池化（输出 1×1 ），将特征图直接压缩为特征向量。这减少了参数量，并具有更好的抗过拟合能力。

4.4.2 测试

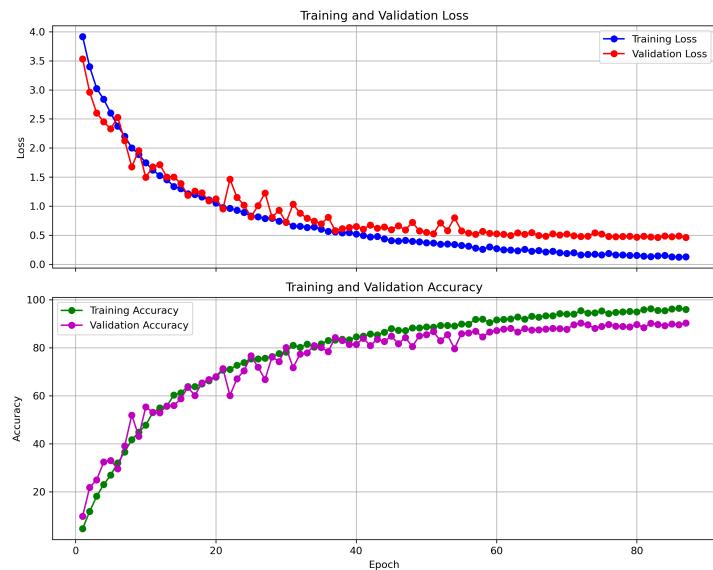


图 12: 训练曲线

```
{  
    "accuracy_top1": 0.8966,  
    "accuracy_top2": 0.9516,  
    "accuracy_top3": 0.9635,  
    "accuracy_top4": 0.9688,  
    "accuracy_top5": 0.9732,  
    "precision_macro": 0.9022,  
    "recall_macro": 0.8961,  
    "f1_score_macro": 0.8966,  
    "precision_micro": 0.8966,  
    "recall_micro": 0.8966,  
    "f1_score_micro": 0.8966,  
    "precision_weighted": 0.9026,  
    "recall_weighted": 0.8966,  
    "f1_score_weighted": 0.8972  
}
```

图 13: 测试集结果

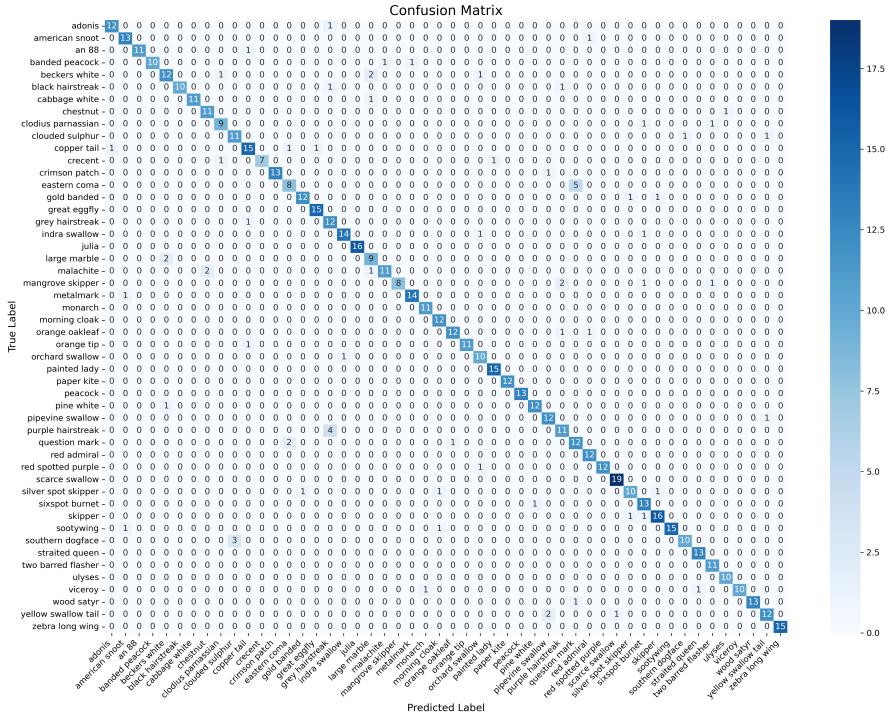


图 14: 混淆矩阵

总体结果来看，ResNet 的表现和 VGG 的表现大致相仿，正确率也达到了 89%，在 Recall 和 F1 上 VGG 表现更好，在 Precision 上则是 ResNet 表现更好；同时 ResNet 最优模型的训练只需要大约 80 epochs，收敛速度更快。不过从训练曲线来看，ResNet 验证集的 Loss 曲线出现比 VGG 还明显的震荡，可能是由于数据集规模较小所致也可能是本人复现的模型减少通道数、或实现有问题导致的。

4.5 模型压缩与轻量化——ButterflyResNet (Tiny)

4.5.1 结构

在实际部署场景中，计算资源往往受限。为了进一步探索模型性能与参数量之间的平衡，本人基于 ResNet 架构进行了结构化压缩，设计了 `ResNet10_tiny`。

- 压缩策略：**保持 `ResNet18_small` 的通道宽度（Base Planes=32）不变，将每个 Stage 的残差块数量从 2 个减少为 1 个（即 `layers=[1, 1, 1, 1]`）。这使网络的深度减半，降低了推理延迟。
- 结构特点：**尽管深度减少，但依然保留了下采样和跳跃连接，保证了基本的特征提取能力。

表 6: 模型压缩对比: ResNet10_tiny 结构详解

层次	模块结构	输入尺寸	输出尺寸	压缩与保留策略
Stem	Conv(7×7 , s=2) → BN → ...	3×224^2	32×56^2	保留: 保持初始感受野
Layer 1	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 1$	32×56^2	32×56^2	压缩: 块数由2减为1
Layer 2	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 1 \text{ (s=2)}$	32×56^2	64×28^2	压缩: 保留下采样功能
Layer 3	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 1 \text{ (s=2)}$	64×28^2	128×14^2	压缩: 减少高维计算量
Layer 4	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 1 \text{ (s=2)}$	128×14^2	256×7^2	压缩: 减少高维计算量
GAP	AdaptiveAvgPool2d(1, 1)	256×7^2	256×1^2	保留: 维持参数效率
Classifier	Flatten → Dropout → Linear	256	50	保留: 分类维度不变

4.5.2 测试

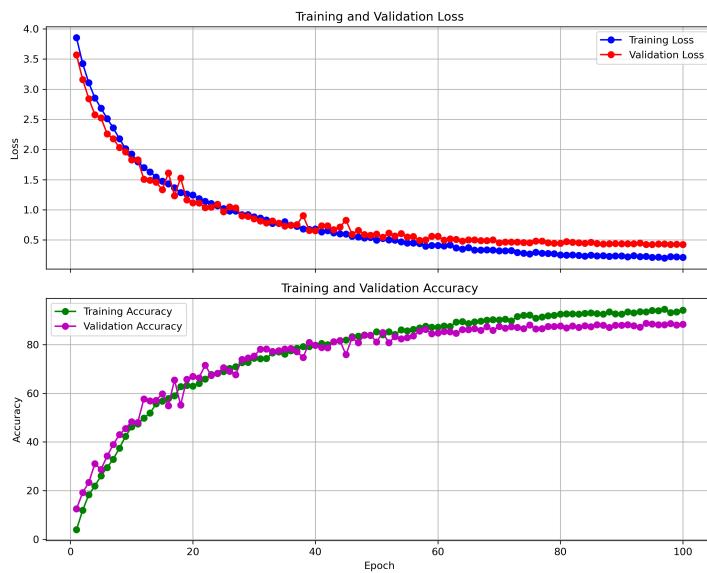


图 15: 训练曲线

```
{
    "accuracy_top1": 0.8899,
    "accuracy_top2": 0.9501,
    "accuracy_top3": 0.9635,
    "accuracy_top4": 0.971,
    "accuracy_top5": 0.9769,
    "precision_macro": 0.8958,
    "recall_macro": 0.8888,
    "f1_score_macro": 0.889,
    "precision_micro": 0.8899,
    "recall_micro": 0.8899,
    "f1_score_micro": 0.8899,
    "precision_weighted": 0.8953,
    "recall_weighted": 0.8899,
    "f1_score_weighted": 0.8896
}
```

图 16: 测试集结果

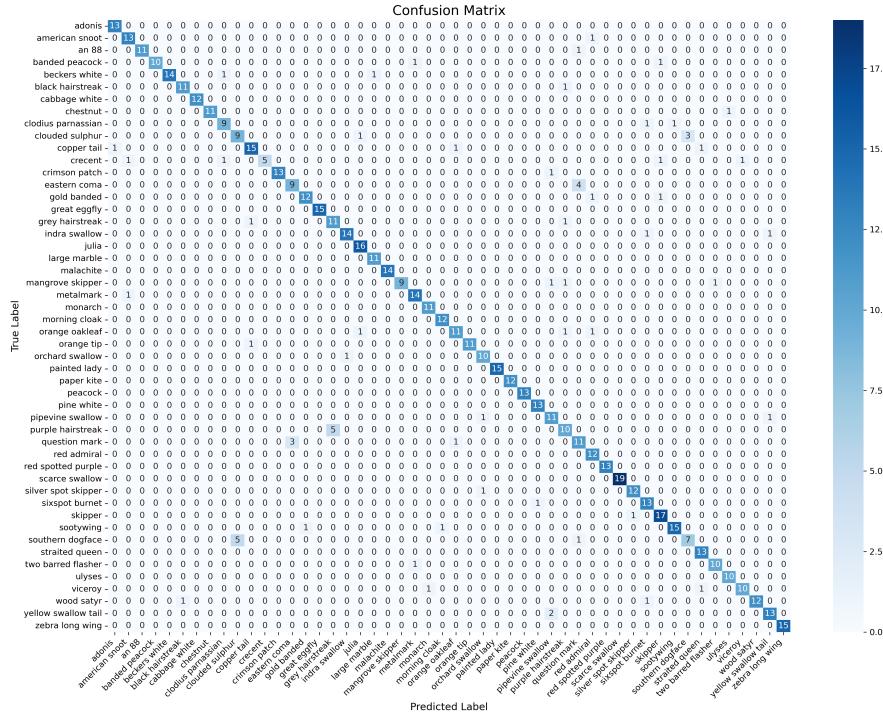


图 17: 混淆矩阵

与 4.4.2 中的结果对比表明，ResNet10_tiny 的参数量相比 ResNet18_small 大幅减小了约一半，但准确率仅下降了约 1%。这证明了通过结构化剪枝可以极大地优化模型效率，而只带来可接受的性能损失。

5 训练策略

- **优化器:** 改进实验中采用了 AdamW 优化器, 相比 SGD, AdamW 结合了动量法和自适应学习率, 并解耦了权重衰减, 在处理复杂纹理时表现更佳。
- **学习率调度:** 采用余弦退火策略 (Cosine Annealing), 并在训练初期进行预热 (Warmup), 随后学习率按余弦函数衰减, 有助于模型跳出局部最优解。
- **损失函数:** 交叉熵损失函数 (CrossEntropyLoss)。
- **超参数扫描:** 利用 wandb 的 sweep 功能, 调节学习率、批大小、权重衰减等超参数, 最终确定了较优的训练配置。

通过对大量实验结果的分析, 我总结出以下几点经验:

1. 在优化器方面, AdamW 普遍上表现的比 Adam 和 SGD 更好, 这是因为 AdamW 集合了 Momentum 和 RMSprop, 同时使用一阶动量和二阶动量, 实现自动调整的学习率, 大幅提升了训练速度和稳定性。 2. 在学习率调度方面, 余弦退火调度器 (Cosine Annealing) 结合预热 (Warmup) 策略表现最佳。预热阶段帮助模型在训练初期稳定收敛, 随后余弦退火逐渐降低学习率, 有助于提升最终性能。 3. 其他超参数方面都没有有表现出太强的规律, 因条件不同而各有优劣。

本实验为我提供了大量调参的机会, 增加了很多调参的经验。

6 模型测试与性能评估

6.1 测试方法与评估指标

本报告采用以下指标, 并重点关注宏平均 Macro-average 结果, 以同等对待每一个类别:

1. 准确率 (Accuracy): 正确分类的样本占总样本的比例。
2. 精确率 (Precision): $Precision = \frac{TP}{TP+FP}$, 衡量模型预测为正类的样本中有多少是真正的正类。
3. 召回率 (Recall): $Recall = \frac{TP}{TP+FN}$, 衡量真实的正类样本中有多少被模型成功找出。
4. F1-Score: 精确率和召回率的调和平均数, 综合反映分类性能:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

6.2 测试结果与解释

下面取 ButterflyMyNet 作为基线模型, 与 ButterflyVGG 进行对比说明。

6.2.1 训练过程分析

从训练曲线 2 和 8 中可以看出:

- **ButterflyMyNet**: 在训练早期 Loss 下降较快，但验证集 Loss 在第 20 个 Epoch 左右开始基本稳定，表现出明显的过拟合倾向。
- **ButterflyVGG**: 得益于 VGG 结构，训练集 Loss 下降相对平缓，且验证集 Loss 与训练集 Loss 差距较小，说明模型的泛化能力得到了显著增强。

6.2.2 测试集性能评估

两种模型在测试集上的最终性能如图 3 和 9 所示。相比于基线模型，ButterflyVGG 在各项指标上均有约 18% 的提升。这验证了深层网络在提取蝴蝶翅膀复杂纹理特征方面的优势。

6.2.3 混淆矩阵分析

两种模型的混淆矩阵如图 4 和 10 所示。通过对比可以发现，ButterflyVGG 的混淆矩阵整体上更接近对角线，说明分类错误的样本数量明显减少。

观察 VGG 网络的混淆矩阵 10，可见对角线区域颜色较深，说明大部分样本被正确分类。而观察非对角线元素也存在比较严重的混淆情况，如 southern dogface 和 copper tail 的混淆相当严重。

7 模型可解释性分析 (Grad-CAM)

为了打开深度学习的黑箱，验证模型是否依据合理的生物学特征（如翅膀花纹）而非背景噪声进行分类，利用 Grad-CAM (Gradient-weighted Class Activation Mapping) [3] 技术对模型进行了可视化。

7.1 算法原理

Grad-CAM 的核心思想是利用梯度流来计算目标卷积层中每个特征图对于特定类别的重要性。

假设卷积神经网络最后一层卷积层的特征图为 A^k (k 为通道索引)，模型对于类别 c 的原始评分 (Logits) 为 y^c 。Grad-CAM 的计算主要分为两步：

1. 计算神经元重要性权重 α_k^c : 通过计算类别评分 y^c 对特征图 A^k 的偏导数，并进行全局平均池化，得到第 k 个特征图对类别 c 的重要性权重：

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

其中 Z 为特征图的像素总数， (i, j) 为空间坐标。

2. 生成加权热力图 $L_{Grad-CAM}^c$: 将特征图 A^k 进行加权求和，并应用 ReLU 激活函数以剔除对类别 c 产生负面影响的特征，仅保留正相关的激活区域：

$$L_{Grad-CAM}^c = \text{ReLU} \left(\sum_k \alpha_k^c A^k \right)$$

7.2 实现流程

具体实现流程如下：

1. Hook Registration：利用 PyTorch 的 `register_forward_hook` 和 `register_backward_hook` 机制，在不修改模型源码的情况下，监控并截取最后一个卷积层的输出特征图和反向传播的梯度。
2. 前向传播与反向传播：
 - 输入图像，执行前向传播得到预测分数。
 - 选定目标类别（预测类别或真实类别），将其分数组置为标量，对模型执行 `backward()`，使梯度回传至目标卷积层。
3. 热力图生成：根据上述公式计算权重并生成粗糙的热力图。
4. 后处理与叠加：将生成的热力图上采样至与原图相同尺寸（ 224×224 ），进行归一化处理，并使用 OpenCV 的 `COLORMAP_JET` 伪彩色映射，最后以 $0.4 : 0.6$ 的透明度比例叠加在原始图像上。

7.3 结果展示与分析

使用表现最好的 ButterflyVGG 模型，对测试集中若干样本进行了 Grad-CAM 可视化，结果如图下：

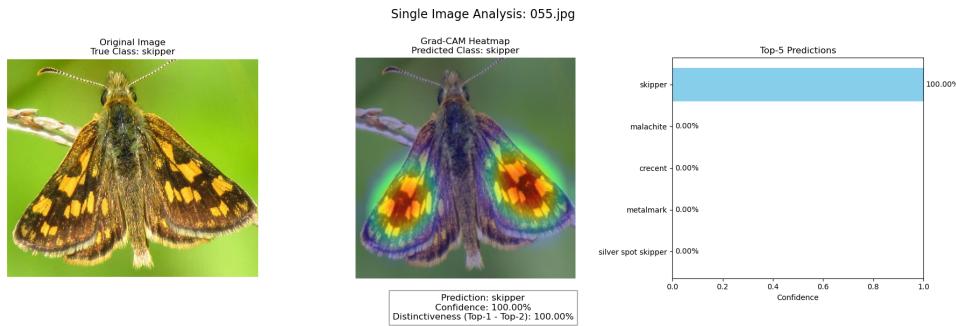


图 19: Grad-CAM 可视化结果与置信度

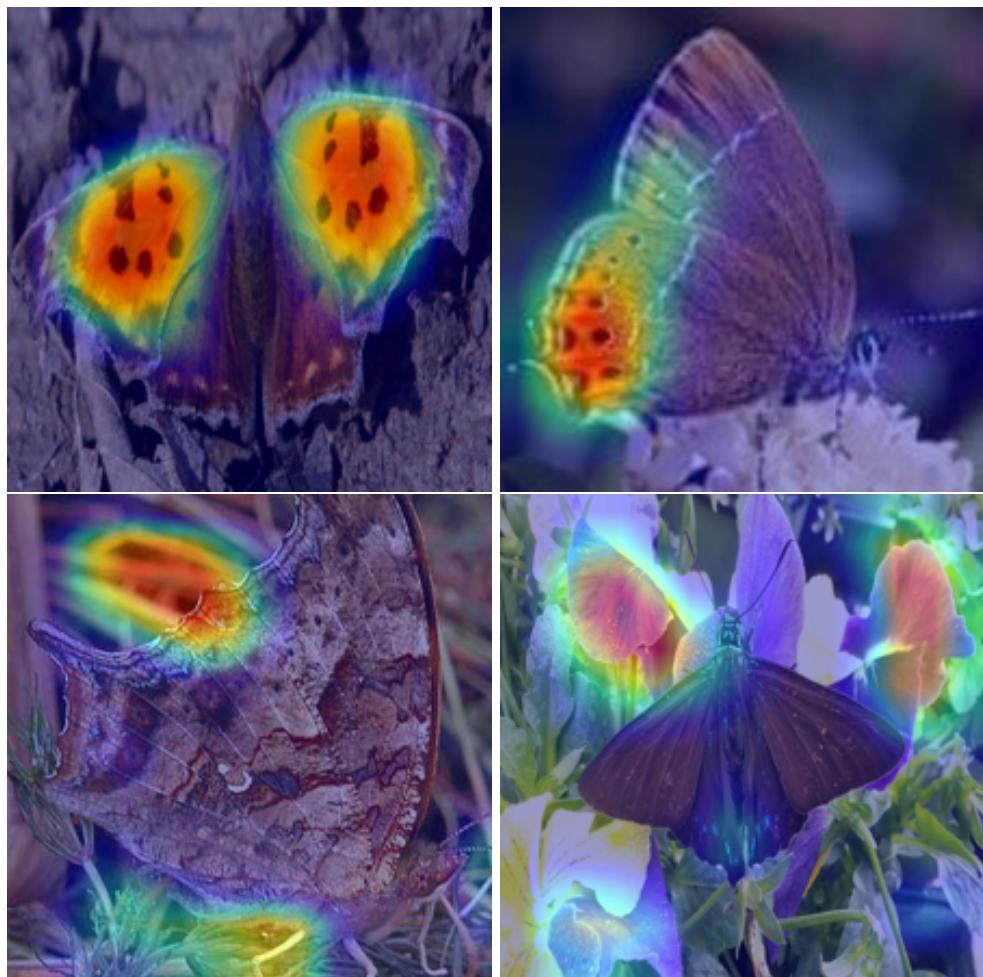


图 18: Grad-CAM 可视化结果（上：分类正确，下：分类错误）

从图 18 中可以观察到：

- 对于大多数样本，模型的热力图高亮区域主要集中在蝴蝶的翅膀纹理和边缘上，这与人分类蝴蝶的依据基本一致。
- 背景区域几乎没有被激活，说明模型成功学习到了主体特征，具有较好的鲁棒性。
- 对于分类错误的样本，热力图有时会分散到背景的纹理上，这说明需要进一步加强对于复杂

背景下的抗干扰训练。

总体来说，该模型可靠性强、决策过程清晰。

8 讨论与展望

8.1 遇到的问题与解决方案

- 工程框架搭建的复杂性：初期在少数几个文件完成所有功能，但这不利于后续扩展、超参数调节和模型切换。为此我重构了代码架构，引入工厂模式和配置驱动，将参数与代码逻辑分离。虽然搭建耗费心力，但极大地提升了后续实验迭代的效率，只需修改配置文件即可完成大部分实验，可更专注于模型设计与调参。
- 过拟合问题：最初我一味堆叠卷积层和增加参数量，导致模型在出现了严重的过拟合现象，训练集准确率接近 95% 而验证集仅为 50%。这时我意识到数据集样本量较少，平均每类仅约 70 张训练图。为此我采取了以下措施：
 1. 减少模型复杂度，设计更合理的网络结构。
 2. 引入 Dropout 层和 Batch Normalization 层，增强模型的正则化能力。
 3. 加入颜色抖动和旋转等变换，丰富训练样本的多样性。
 4. 采用 Early Stopping 策略，在验证集性能不再提升时提前终止训练。
- Grad-CAM 实现中的数据泄露：在初次实现 Grad-CAM 时，发现测试集准确率异常波动。经排查发现：在生成热力图时未将模型置为 `eval()` 模式，导致 Batch Normalization 层在处理单张测试图片时更新了模型的参数。在分析阶段显式调用 `model.eval()` 后解决，保证可视化准确性和模型参数的安全性。
- 数据准备时间过长：虽然使用了 Cuda 加速训练，但我注意到在一个 epoch 的 30 秒内，有约 20 秒的时间花在了数据加载上，大大拖慢了实验进度。疑惑的是我已经使用了 `num_workers=8` 和 `pin_memory=True`。最终我发现这是 windows 系统多线程的问题，在一个 epoch 之前创建的子进程会被销毁，导致每次都要重新创建进程池。将 `persistent workers` 参数设为 `True` 后，除了第一次 epoch 外，后续 epoch 的数据加载几乎不占用时间，极大提升了训练效率。

8.2 不足与展望

尽管模型性能有了显著提升，但仍有改进空间：

- 数据增强：当前的数据增强方法较为基础，未来可以尝试更复杂的技术，如 CutMix 或 MixUp，以进一步提升模型的泛化能力。
- 细粒度分类优化：对于拟态相似的蝴蝶，可以引入注意力机制或双线性池化）来捕捉更细微的局部特征。
- 预训练模型迁移：根据实验要求，本实验从头训练模型。实际上利用在 ImageNet 上预训练的 ResNet 或 EfficientNet 进行迁移学习，通常能以更少的训练代价获得更高的准确率。

8.3 总结

本次大作业完成过程中遇到了很多阻力。从最简单的卷积神经网络、增加深度与宽度、引入批次归一化和正则化、调整学习率与增加优化器、尝试不同学习率调度器、引入多种数据增强、疯狂调参，到复现经典网络结构、实现 Grad-CAM 可视化、模型剪枝与压缩，每一步都充满了挑战。但通过不断尝试与调试，最终成功设计并训练出了性能较好的模型。遗憾的是，由于时间有限未能尝试更多先进的架构和技术。

同时也有很多新的收获和体会：学习并实验了各种各样的经典网络结构、使用 wandb 的 sweep 功能扫描超参数进行比较、实验了可视化卷积网络决策过程，对卷积神经网络有了更深入的理解，对深度学习的流程也更加熟悉。

参考文献

1. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
3. Selvaraju, R. R., Cogswell, M., Das, A., et al. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision* (pp. 618-626).