# 大作业

## 几何变换p43

* **图像几何变换过程**

1. **先计算新图像的尺寸**
2. **根据几何变换的反变换，确定新图像中的每一点再原图像中的对应点。**
3. **按对应关系给新图像中各像素赋值。**

**若原图像中的点对应存在，直接赋值。**

**若原图像中对应的点坐标超出图像宽高范围，直接赋为背景色。**

**若原图像中对应的点坐标在图像宽高范围内，但坐标不是整数，则采用插值的方法计算。**

* **图像缩放**

**原图像中的点(x,y)进行缩放处理后，变换到点(x’,y’)，则缩放处理的矩阵可以表示为**

****

**根据该公式，我们可以知道变换后的点对应原图像中的点，从而实现图像缩放。**

**实现代码：**

1. function new\_img = bilinear\_resize(f, kx, ky)
2. % 函数功能: 使用双线性插值对图像进行缩放
3. % 输入参数:
4. %   f  - 输入图像 (可以是灰度图像或彩色图像)
5. %   kx - 水平方向缩放因子
6. %   ky - 垂直方向缩放因子
7. % 输出:
8. %   new\_img - 缩放后的图像
10. % 获取原图像的大小
11. [h, w, c] = size(f);  % c为通道数（如果是灰度图，c=1）
13. % 缩放后的图像大小
14. new\_h = ceil(h \* ky);
15. new\_w = ceil(w \* kx);
17. % 初始化放大后的图像
18. new\_img = zeros(new\_h, new\_w, c);
20. % 对每个颜色通道分别进行双线性插值操作
21. **for** ch = 1:c
22. **for** y\_new = 1:new\_h
23. **for** x\_new = 1:new\_w
24. % 计算在原图像中的位置
25. x\_orig = (x\_new - 1) / kx;
26. y\_orig = (y\_new - 1) / ky;
28. % 如果该点在原图像范围内，则进行插值，否则设置为背景色0
29. **if** x\_orig >= 0 && x\_orig < w && y\_orig >= 0 && y\_orig < h
30. % 找到原图像中的四个邻近像素的坐标
31. x = floor(x\_orig) + 1;
32. y = floor(y\_orig) + 1;
34. % 计算插值的权重
35. a = x\_orig - (x - 1);  % 水平方向的权重
36. b = y\_orig - (y - 1);  % 垂直方向的权重
38. % 如果在图像范围内，使用四个像素进行插值
39. **if** x < w && y < h
40. % 获取四个邻近像素的值
41. f11 = f(y, x, ch);         % 左上角
42. f12 = f(y + 1, x, ch);     % 左下角
43. f21 = f(y, x + 1, ch);     % 右上角
44. f22 = f(y + 1, x + 1, ch); % 右下角
45. % 计算插值后的像素值
46. new\_img(y\_new, x\_new, ch) = (1 - a) \* (1 - b) \* f11 + ...
47. a \* (1 - b) \* f21 + ...
48. (1 - a) \* b \* f12 + ...
49. a \* b \* f22;
50. % 如果超出图像宽度范围，则在垂直方向进行插值
51. elseif x >= w && y < h
52. new\_img(y\_new, x\_new, ch) = f(y, w, ch) + ...
53. b \* (f(y + 1, w, ch) - f(y, w, ch));
54. % 如果超出图像高度范围，则在水平方向进行插值
55. elseif x < w && y >= h
56. new\_img(y\_new, x\_new, ch) = f(h, x, ch) + ...
57. a \* (f(h, x + 1, ch) - f(h, x, ch));
58. % 如果x和y都超出图像范围，则直接使用右下角的像素值
59. **else**
60. new\_img(y\_new, x\_new, ch) = f(h, w, ch);
61. end
62. **else**
63. % 超出原图像范围时，设置背景色
64. new\_img(y\_new, x\_new, ch) = 0;
65. end
66. end
67. end
68. end
70. % 将结果转换为8位图像
71. new\_img = uint8(new\_img);
72. end

* **图像旋转**

**原图像中点(x,y)绕原点逆时针旋转θ角后的对应点(x’,y’)可以表示为**

****

**绕原点旋转的反变换为**

****

**实现代码：**

1. function new\_img = rotate\_image(f, theta)
2. % 函数功能: 使用双线性插值法对图像进行逆时针旋转
3. % 输入参数:
4. %   f      - 输入图像
5. %   theta  - 旋转角度（弧度制）
6. % 输出:
7. %   new\_img - 旋转后的图像
9. % 获取原图像的大小
10. [h, w, c] = size(f);
12. % 步骤一：先确定旋转后的图像大小
13. x = [0 w-1 w-1 0];
14. y = [0 0 h-1 h-1];
15. x\_new = x\*cos(theta) + y\*sin(theta);
16. y\_new = -x\*sin(theta) + y\*cos(theta);
18. % 找新图像的大小
19. minx = min(x\_new);
20. miny = min(y\_new);
21. maxx = max(x\_new);
22. maxy = max(y\_new);
23. H = ceil(maxy - miny + 1);
24. W = ceil(maxx - minx + 1);
26. % 初始化新图像
27. new\_img = zeros(H, W, c);
29. % 对每个通道分别进行处理
30. **for** ch = 1:c
31. **for** newx = 1:W
32. **for** newy = 1:H
33. % 平移变换
34. x\_temp = newx - 1 + minx;
35. y\_temp = newy - 1 + miny;
37. % 反变换
38. oldx = x\_temp\*cos(theta) - y\_temp\*sin(theta);
39. oldy = x\_temp\*sin(theta) + y\_temp\*cos(theta);
41. % 判断是否在原图像范围内
42. **if** oldx < 0 || oldy < 0 || oldx >= w || oldy >= h
43. new\_img(newy, newx, ch) = 0;
44. **else**
45. x = floor(oldx) + 1;
46. y = floor(oldy) + 1;
47. a = oldx - floor(oldx);
48. b = oldy - floor(oldy);
50. % 双线性插值
51. **if** x < w && y < h
52. f11 = f(y, x, ch);        % 左上角
53. f12 = f(y + 1, x, ch);    % 左下角
54. f21 = f(y, x + 1, ch);    % 右上角
55. f22 = f(y + 1, x + 1, ch);% 右下角
56. new\_img(newy, newx, ch) = (1 - a) \* (1 - b) \* f11 + ...
57. a \* (1 - b) \* f21 + ...
58. (1 - a) \* b \* f12 + ...
59. a \* b \* f22;
60. elseif x >= w && y < h
61. new\_img(newy, newx, ch) = f(y, w, ch) + b \* (f(y + 1, w, ch) - f(y, w, ch));
62. elseif x < w && y >= h
63. new\_img(newy, newx, ch) = f(h, x, ch) + a \* (f(h, x + 1, ch) - f(h, x, ch));
64. **else**
65. new\_img(newy, newx, ch) = f(h, w, ch);
66. end
67. end
68. end
69. end
70. end
71. % 将结果转换为uint8格式
72. new\_img = uint8(new\_img);
73. end

* **图像错切**

**图像的错切变换是平面景物在投影平面上的非垂直投影。错切变换使图像中的图形产生扭变。这种扭变只在水平或者垂直方向产生时，分别称为水平方向错切和垂直方向错切。**

**设原图像中的点(x,y)进行错切变换后，变换到点(x’,y’)，则错切变换的矩阵表达式为：**

****

**实现代码：**

1. function new\_img = shear\_image(f, shear\_factor\_xy, shear\_factor\_yx)
2. % 函数功能: 对图像同时进行水平和垂直方向的错切变换
3. % 输入参数:
4. %   f               - 输入图像
5. %   shear\_factor\_xy - 水平方向错切因子
6. %   shear\_factor\_yx - 垂直方向错切因子
7. % 输出:
8. %   new\_img         - 变换后的图像
10. % 获取原图像的大小
11. [h, w, c] = size(f);
13. % 水平和垂直方向的错切变换矩阵
14. T = [1 shear\_factor\_xy; shear\_factor\_yx 1];
16. % 计算变换后图像的范围
17. corners = [1, 1, w, w; 1, h, 1, h];  % 原图像四个角的坐标
18. new\_corners = T \* corners;           % 变换后的四个角坐标
20. minx = min(new\_corners(1, :));
21. maxx = max(new\_corners(1, :));
22. miny = min(new\_corners(2, :));
23. maxy = max(new\_corners(2, :));
25. % 计算新图像的大小
26. new\_w = ceil(maxx - minx);
27. new\_h = ceil(maxy - miny);
29. % 初始化新图像
30. new\_img = zeros(new\_h, new\_w, c);
32. % 对每个通道分别处理
33. **for** ch = 1:c
34. **for** newx = 1:new\_w
35. **for** newy = 1:new\_h
36. % 反向变换到原图坐标
37. pos = inv(T) \* [(newx + minx - 1); (newy + miny - 1)];
38. oldx = pos(1);
39. oldy = pos(2);
41. % 判断是否在原图像范围内
42. **if** oldx >= 1 && oldx <= w && oldy >= 1 && oldy <= h
43. % 双线性插值
44. x1 = floor(oldx);
45. x2 = ceil(oldx);
46. y1 = floor(oldy);
47. y2 = ceil(oldy);
49. % 计算插值权重
50. a = oldx - x1;
51. b = oldy - y1;
53. % 获取邻近像素的值
54. **if** x1 < 1, x1 = 1; end
55. **if** y1 < 1, y1 = 1; end
56. **if** x2 > w, x2 = w; end
57. **if** y2 > h, y2 = h; end
59. f11 = **double**(f(y1, x1, ch));  % 左上角
60. f12 = **double**(f(y2, x1, ch));  % 左下角
61. f21 = **double**(f(y1, x2, ch));  % 右上角
62. f22 = **double**(f(y2, x2, ch));  % 右下角
64. % 插值计算
65. new\_img(newy, newx, ch) = (1 - a) \* (1 - b) \* f11 + ...
66. a \* (1 - b) \* f21 + ...
67. (1 - a) \* b \* f12 + ...
68. a \* b \* f22;
69. **else**
70. % 超出原图范围设为背景色0
71. new\_img(newy, newx, ch) = 0;
72. end
73. end
74. end
75. end
77. % 将结果转换为uint8格式
78. new\_img = uint8(new\_img);
79. end

* **图像镜像**

**设图像的分辨率为M×N，采用像素坐标系，图像镜像变换矩阵公式如下所示，镜像就是左右、上下或者对角对换。**

**水平镜像：**

**垂直镜像：**

**对角镜像：**

**实现代码：**

1. function new\_img = mirror\_image(f, mode)
2. % 函数功能: 对图像进行镜像变换
3. % 输入参数:
4. %   f    - 输入图像
5. %   mode - 选择镜像模式, 'horizontal' 水平镜像，'vertical' 垂直镜像, 'diagonal' 对角镜像
6. % 输出:
7. %   new\_img - 变换后的图像
9. [h, w, c] = size(f);  % 获取原图像大小
11. % 初始化新图像
12. new\_img = zeros(h, w, c);
14. **switch** mode
15. **case** 'horizontal'
16. % 水平镜像（沿着垂直轴翻转）
17. **for** y = 1:h
18. **for** x = 1:w
19. new\_img(y, x, :) = f(y, w - x + 1, :);
20. end
21. end
23. **case** 'vertical'
24. % 垂直镜像（沿着水平轴翻转）
25. **for** y = 1:h
26. **for** x = 1:w
27. new\_img(y, x, :) = f(h - y + 1, x, :);
28. end
29. end
31. **case** 'diagonal'
32. % 对角镜像（沿着对角线翻转）
33. % 假设输入是方阵，若不是，则以左上角矩阵为主
34. **for** y = 1:min(h, w)
35. **for** x = 1:min(h, w)
36. new\_img(x, y, :) = f(y, x, :);  % 交换行列位置
37. end
38. end
40. otherwise
41. error('未知的镜像模式. 请选择 horizontal, vertical 或 diagonal.');
42. end
44. % 将新图像转换为uint8类型
45. new\_img = uint8(new\_img);
46. end

## 直方图均衡化p116

**直方图修正法的基本原理就是通过构造灰度级变换函数改造原图像的直方图，使变换后的图像的直方图达到一定要求。**

**直方图均衡化是采用灰度级r的累积分布函数作为变换函数的直方图修正法。给出一幅数字图像，共有L个灰度级，总像素个数为N，其中，第j级灰度对应的像素数为。对该图像进行灰度直方图统计，则图像进行直方图均衡化处理的变换函数T（r）为**



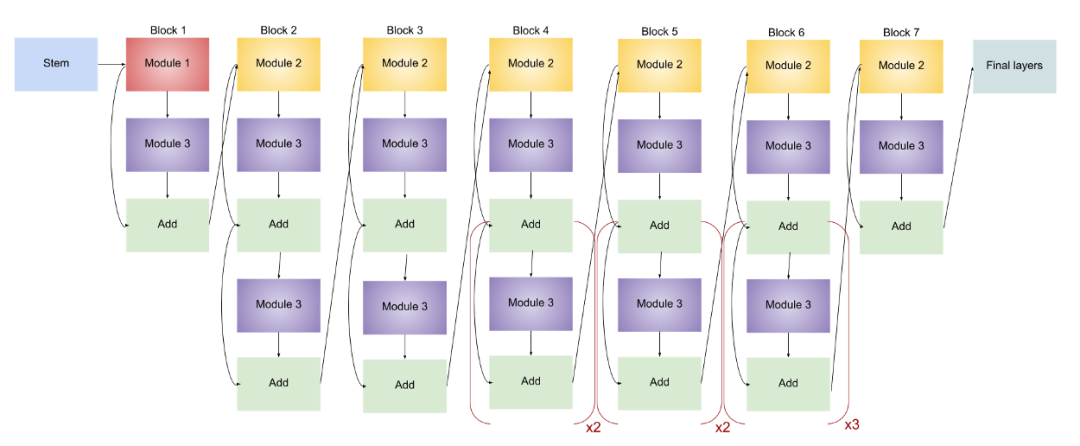
**对一幅数字图像进行直方图均衡化处理的算法步骤如下：**

1. **统计原始图像的直方图。**
2. **计算新的灰度级。**
3. **修正sk为合理的灰度级。**
4. **计算新的直方图。**
5. **用处理后的新灰度代理处理前的灰度，生成新图像。**

## 基于EfficientNet-B2模型的鸟类分类任务

### EfficientNet模型

EfficientNet是由Google Brain提出的一种高效的卷积神经网络（CNN）架构，其核心思想是在网络的深度、宽度和输入分辨率这三方面同时进行复合缩放，从而实现更高的计算效率和更好的精度。EfficientNet的提出是为了应对计算资源和模型精度之间的权衡问题，通过创新的模型设计在减少计算成本的同时提升模型的性能。



**EfficientNet的主要创新之一是复合缩放方法。传统的神经网络设计通常在深度、宽度和输入分辨率上独立地进行调整，而EfficientNet通过复合缩放将这三个维度进行有机结合，创建了一个可以在计算资源与精度之间平衡的方案。复合缩放的目标是根据模型的初始版本来优化网络的深度、宽度和分辨率，找到最佳的缩放系数，避免传统方法中可能出现的参数过多或过少的情况。**

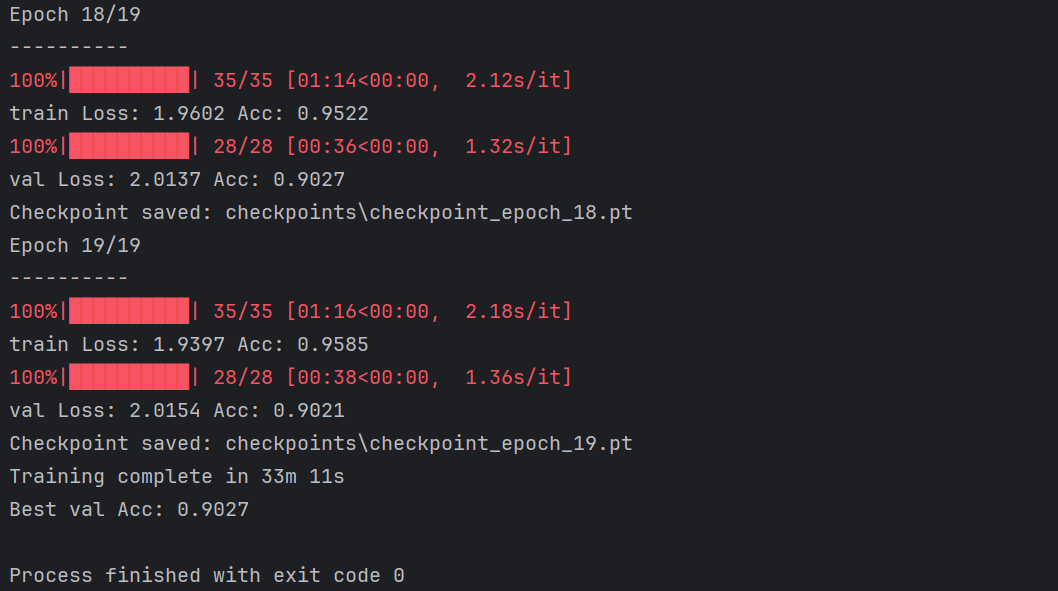
**相比于传统的CNN架构，EfficientNet在相同计算资源下能够提供更好的精度。它能够在减少计算量和参数量的同时提高准确率，从而减少计算开销。**

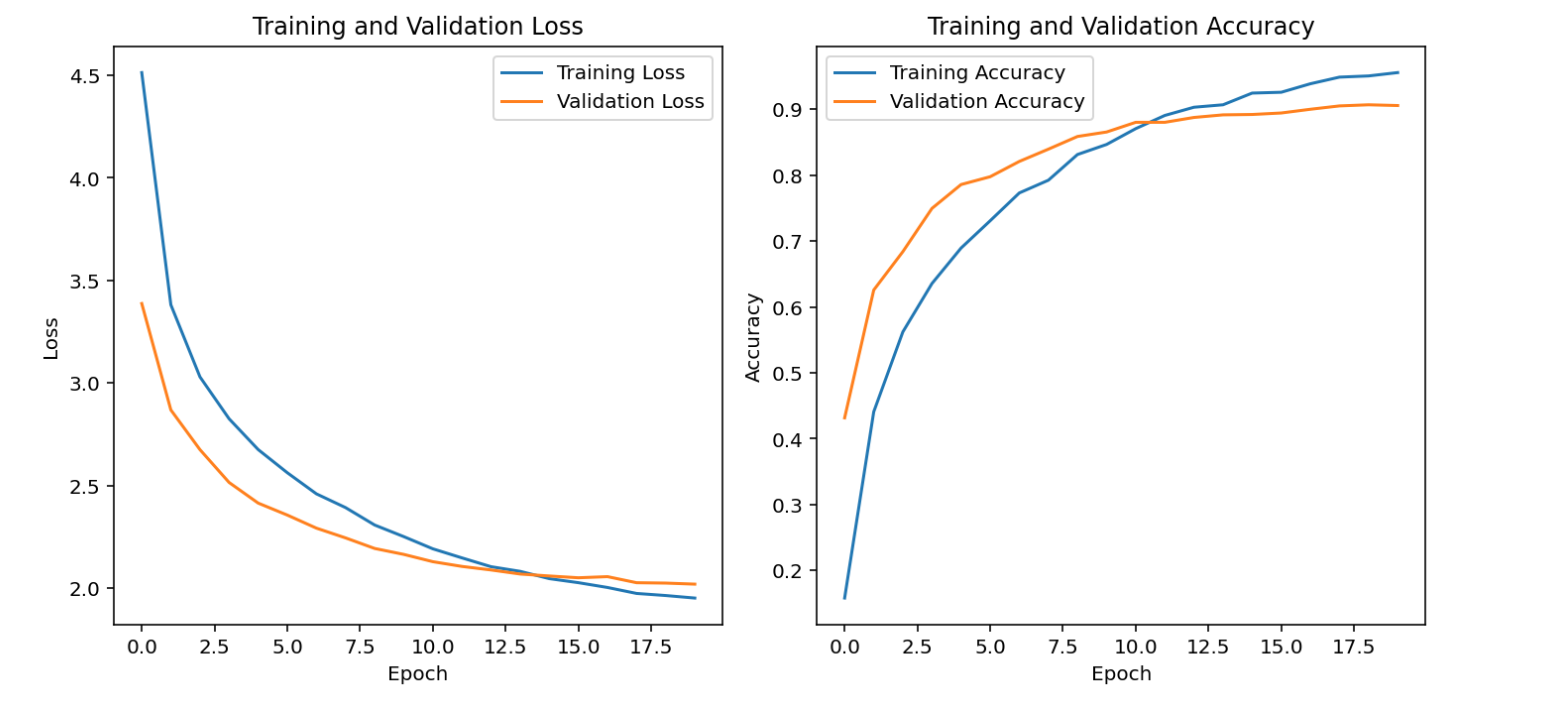
### 训练模型

模型架构选择预训练的EfficientNet-B2模型。EfficientNet是一种高效的卷积神经网络架构，通过复合缩放方法同时优化网络的深度、宽度和分辨率，使其在计算效率和精度之间取得平衡。在本任务中，我们冻结了EfficientNet-B2的预训练权重，仅训练其分类头部分，以适应我们的特定任务。

数据预处理是深度学习中提高模型性能的关键步骤。在本分类任务中，我们对训练集应用了数据增强技术，包括随机水平翻转、随机垂直翻转和颜色扰动等。验证集和测试集则进行了标准化处理，以减少训练数据偏差对模型性能的影响。

训练过程中采用了AdamW优化器，它能够有效地控制学习率并减少过拟合的风险。同时，使用了带标签平滑的交叉熵损失函数，标签平滑能够让模型对类别不确定性进行建模，避免过于自信的预测。为了避免模型过拟合，我们还对全连接层应用了Dropout正则化。





## 基于Resnet18模型的鸟类分类任务

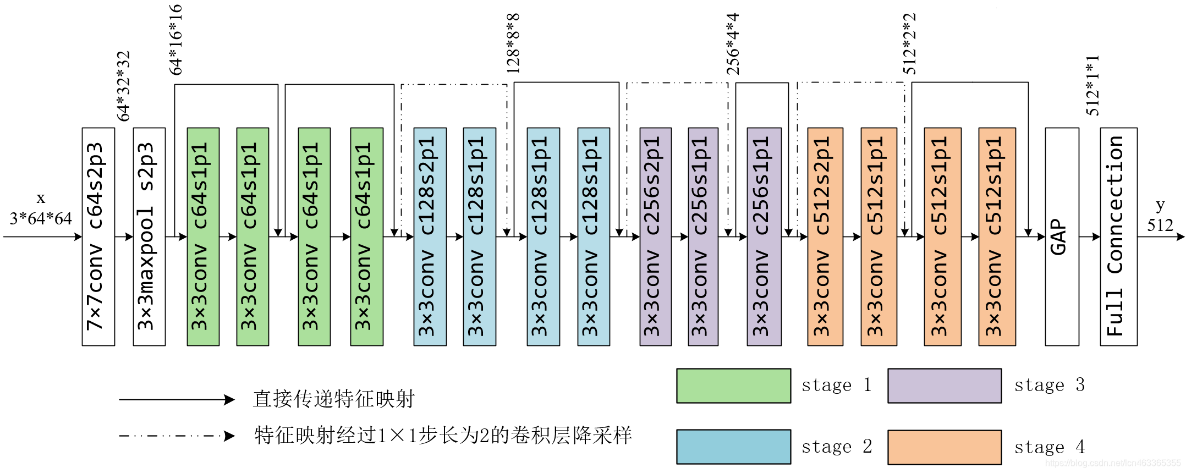
### Resnet模型

残差神经网络(ResNet)是由微软研究院的何恺明、张祥雨、任少卿、孙剑等人提出的。ResNet 在2015 年的ILSVRC（ImageNet Large Scale Visual Recognition Challenge）中取得了冠军。残差神经网络的主要贡献是发现了“退化现象（Degradation）”，并针对退化现象发明了 “快捷连接（Shortcut connection）”，极大的消除了深度过大的神经网络训练困难问题。其核心思想是通过添加额外的连接来解决深度神经网络训练中的梯度消失和梯度爆炸等问题，从而允许构建非常深的神经网络。

ResNet通过引入快捷连接，允许某一层的输出直接跳过一个或多个层，连接到后续层的输入。这样做的好处是，即使某些层不做任何有意义的变换，它们仍然可以传递之前层的信息，而不会对梯度产生过多的损失。这可以用一个公式来表示：



ResNet18的基本含义是，网络的基本架构是ResNet，网络的深度是18层。但是这里的网络深度指的是网络的权重层，也就是包括池化，激活，线性层。而不包括批量化归一层，池化层。图—————就是一个ResNet18的基本网络架构，其中并未加入批量化归一和池化层。



### 训练模型

模型架构使用的是预训练的 ResNet-18 网络，并冻结卷积层的权重，只训练最后的全连接层。我们修改了全连接层的结构，通过增加层数和神经元数量，使其能够处理更复杂的分类任务。

数据预处理与增强、损失函数和优化器与训练EfficientNetB2模型时使用的一致，此处不再赘述。

