利用 Swin-v2 模型实现目标分类

任柳博 U202115156 人工智能 2103 班

1 引言

在现代计算机视觉领域，注意力机制和 Transformer 架构的发展为图像处理和目标分类带来了革命性的进步。注意力机制最初应用于自然语言处理[1]，其核心思想在于根据任务的需求，灵活地关注输入序列中的不同部分，从而实现更精准的表征和更有效的信息提取，通过动态调整模型对不同输入部分的关注度，大幅提高了特征提取的精度和效率。

2017 年，Transformer 模型的提出标志着一种全新的模型架构[2]，它完全摒弃了传统的卷积神经网络（CNN）和循环神经网络（RNN），依靠自注意力机制来处理序列数据。这种创新不仅在自然语言处理任务中取得了巨大的成功，也启发了研究人员将其应用于计算机视觉领域。Vision Transformer (ViT)是这一趋势的代表性模型，但其计算复杂度和对大规模数据集的依赖限制了其应用范围[3]。为了克服这些挑战，研究人员提出了一系列改进模型，其中 Swin Transformer（Shifted Window Transformer）因其独特的分层架构和局部注意力机制而受到广泛关注[4]。Swin Transformer 通过将图像分解为多个层次的窗口，并在窗口内部应用自注意力机制，有效地降低了计算复杂度，同时保持了高效的特征提取能力。

Swin-v2 模型作为 Swin Transformer 的改进版本，引入了更多的层次结构和多尺度特征提取方法，能够更好地处理高分辨率图像和复杂的视觉任务[5]。其主要改进包括更高效的注意力计算方式、更深的网络结构以及更强的特征表示能力，这些改进使得 Swin-v2 在多个基准测试中表现优异。Swin-v2 不仅能够有效地捕捉图像中的细节信息，还能够在较低计算成本的情况下，实现与传统卷积神经网络相媲美甚至超越的分类精度。因此，本报告选择 Swin-v2 模型进行目标分类研究，旨在探讨其在分类视觉任务中的应用潜力和优势。

2 方法

Swin-v2 模型（Shifted Window Transformer v2）的主要目的是在保持高性能的同时，通过引入分层结构和局部注意力机制，进一步优化计算效率和处理高分辨率图像的能力。相比传统的卷积神经网络（CNN），Swin-v2 模型利用 Transformer 的自注意力机制，更有效地捕捉图像中的长距离依赖关系和复杂特征，特别适用于大规模视觉任务，如目标分类、目标检测和语义分割等[5]。

2.1 Swin Transformer

2.1.1 模型原理

Swin Transformer 是一种 Vision Transformer[4]。主要原理基于 Transformer 的自注意力机制，但通过引入局部窗口和移动窗口策略，解决了全局自注意力在高分辨率图像上的计算复杂度问题。

具体来说，模型首先将输入图像划分为多个局部窗口，每个窗口内应用多头自注意力机制。这种局部处理方式显著降低了计算量，同时保持了局部特征的捕捉能力，并且，由于仅在每个局部窗口内计算自注意力，使得它对输入图像大小的计算复杂度为线性。正是这样，它可以作为图像分类和密集识别任务的通用骨干。

此外，为了确保模型能够捕捉全局信息，Swin Transformer 在每层之间引入了窗口移动（Shifted Window）机制，通过交替移动窗口位置，使得相邻窗口之间的特征可以交互。这种设计既保留了自注意力机制的优势，又有效地降低了计算成本。

分层结构使得模型在不同尺度上提取特征，从而提高了对高分辨率图像和复杂场景的处理能力。每一层的输出作为下一层的输入，通过逐层抽象，模型能够捕捉到从局部细节到全局语义的多层次信息。

2.1.2 模型结构

Swin Transformer 模型的结构可以概括为以下几个关键部分：

1.Patch Partitioning（块划分）：输入图像被划分为多个固定大小的非重叠块。

2.Linear Embedding（线性嵌入）：每个图像块被映射到特定维度的特征空间，形成初始的嵌入表示。

3.Multi-Head Self-Attention（多头自注意力）：在每个窗口内应用多头自注意力机制，提取局部特征。

4.Shifted Window（窗口移动）：窗口在每层之间交替移动，以捕捉跨窗口的全局信息。

5.Hierarchical Structure（分层结构）：模型逐层进行特征提取，每层输出作为下一层的输入。

6.Patch Merging（补丁融合）：对特征图进行下采样处理，减少空间维度，增加通道数。

7.MLP Head（多层感知器头部）：最终将综合特征输入到多层感知器进行分类，输出分类结果。

下图展示了 Swin Transformer 模型的详细结构[4]：

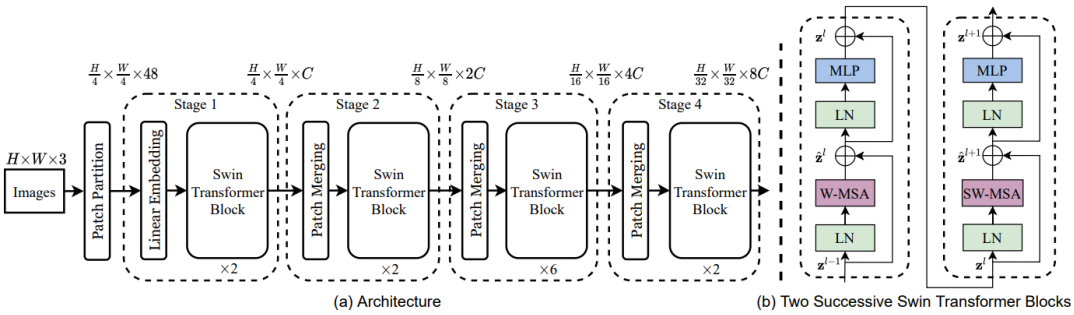

图 1.(a)Swin Transformer 的结构示意图；(b)两个连续的 Swin Transformer 块

从图中可以看到，Swin Transformer 模型通过多个阶段逐步处理输入图像，每个阶段都包含一个或多个 Swin Transformer 模块，并通过补丁融合操作（Patch Merging）减少特征图的空间维度，同时增加通道数。

2.2 Swin-v2 的改进

Swin-v2 模型在 Swin-T 的基础上进行了改进和优化。主要的改进包括：

1. 注意力机制方面：在 Swin-v2 中，自注意力机制由余弦相似度计算替代了传统的点积计算。这一改进能够更好地捕捉特征之间的关系，并且在计算上更加稳定和高效。

2. 位置编码方面：Swin-v2 引入了对数相对位置偏差（Log-CPB）机制，通过多层感知器（MLP）对位置偏差进行学习和调整，增强模型的空间感知能力。

3. 层归一化方面：Swin-v2 在注意力和前馈网络层之后进行层归一化而不是在之前，优化了模型的稳定性和训练效率。

Swin Transformer V2 的结构块如下图所示[5]，其与第一代的区别被用红色突出。

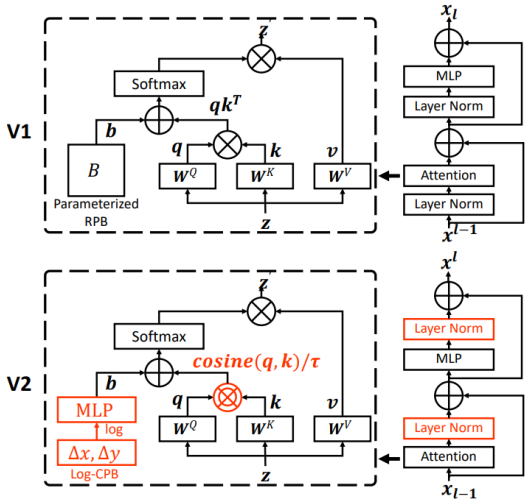

图 2.Swin Transformer V2 的块结构

3 实验和结果

3.1 数据集

本次实验采用 CIFAR-100 数据集进行模型训练与评估。CIFAR-100 数据集是计算机视觉领域中广泛使用的标准数据集之一，由加拿大多伦多大学的 Alex Krizhevsky 和 Geoffrey Hinton 在 2009 年收集和发布，旨在推动计算机视觉算法的研究与发展[6]。

CIFAR-100 数据集主要用于图像分类任务，特别适用于评估和比较不同图像分类算法的性能。由于其多样性和复杂性，该数据集广泛应用于学术研究和工业界，包括但不限于深度学习、图像识别、物体检测和图像生成等领域。

CIFAR-100 数据集包含 60000 张彩色图像，其中 50000 张用于训练，10000 张用于测试。数据集分为 100 个类别，每个类别包含 600 张图像。类别之间存在一定的层次结构，划分为 20 个超级类，每个超级类包含若干个子类。其中，图像以 RGB 格式存储，每张图像具有 32x32x3 的尺寸。数据集以二进制格式提供，每张图像的像素值范围为 0 到 255。标签信息包含在数据集中，每张图像对应一个类别标签（0-99）。

数据集中的标签文件以字典格式存储，包含两个键值对：“fine_labels”和“coarse_labels”，分别对应细粒度标签和粗粒度标签的列表，细粒度标签表示图像的具体类别（共 100 种），粗粒度标签表示图像的超级类（共 20 种），例如，超级类“食物”包含“苹果”、“橙子”等细粒度类别。下面是标签的示例：

```
{
"fine_labels": [19, 29, 33, ...],"coarse_labels": [3, 6, 11, ...]
}
```

3.2 实验指标

在本次实验中，为了训练和评估 Swin-v2 模型在 CIFAR-100 数据集上的分类性能，我们使用了交叉熵损失函数和 Top-1 准确率 ACC@1 两个关键指标：

3.2.1 交叉熵损失函数（Cross-Entropy Loss）

交叉熵损失函数是分类任务中常用的损失函数，能够度量模型预测结果与实际标签之间的差异。其基本思想是通过比较模型输出的概率分布与实际分布之间的差异，来衡量模型的预测性能。具体来说，交叉熵损失函数通过计算预测概率与实际类别的对数概率之积的负和来反映预测的准确性。具体实现见方程（1）：

$$\text{Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{1}$$

其中，N 是样本数，y_i 是实际标签的独热编码，$\hat{y}_i$ 是模型的预测概率。通过最小化交叉熵损失，模型可以在训练过程中逐步调整参数，提高对训练数据的拟合程度，从而提升预测准确性。

3.2.2 Top-1 准确率（ACC@1）

Top-1 准确率是分类任务中常用的评估指标之一，表示模型预测的类别与实际类别完全一致的比例。具体来说，Top-1 类别预测指的是模型在所有类别中选取概率最大的一个类别作为预测结果。如果该预测类别与实际标签相符，则视为一次正确的预测。ACC@1 是 Top-1 准确率的缩写，其计算公式见方程（2）：

$$\text{ACC@1} = \frac{1}{N}\sum_{i=1}^{N} 1(\hat{y}_i = y_i) \tag{2}$$

其中，$1(\hat{y}_i = y_i)$是指示函数，当预测类别$\hat{y}_i$ 与实际类别y_i 相同时，取值为 1，否则为 0。Top-1 准确率反映了模型在测试集上的整体预测精度，是评估模型性能的关键指标之一。

3.3 训练

鉴于预训练模型通常能够显著提升训练效果和效率，并减少训练成本，而从头开始训练模型不仅耗时长，并且要获得性能优异的预训练模型也并非易事，往往需要许多技术细节的支持[7]。因此，在本次实验中，我分别对未经预训练的 Swin-v2 模型和 Transformers 库中官方提供的经过预训练的 Swin-v2 模型[8]，在 CIFAR-100 数据集上进行训练。需要指出的是，该 Swin Transformer v2 模型是在 ImageNet-1k 数据集上以 256x256 的分辨率进行预训练的。我们所有实验统一使用两张 RTX 3080 显卡，在 Python 3.8.10 环境下进行。我们使用了 PyTorch 2.0.0+cu118 版本，并借助 Transformers 库[8]来更加方便地构建 Swin-v2 模型，该库版本为 4.41.2。模型学习率设置为 1E-4，并采用 Adam 优化器进行优化。二者训练过程中具体的区别如表格 1 所示。

Model	No-Pretrained	Pretrained
Batch Size	64	20
Epoch	160	50
Train Time	14h	11.5h

表格 1.无预训练和预训练模型的训练细节差别

在训练过程中，我们规定每 1 个 epoch 使用测试集进行一次测试，获得该 epoch 的 Top-1 准确率，并且，我们以 Top-1 准确率 ACC@1 为标准保存最优模型。此外，我们记录了训练损失随训练轮数的变化情况，二者的训练损失曲线

图见图 3 和图 4。

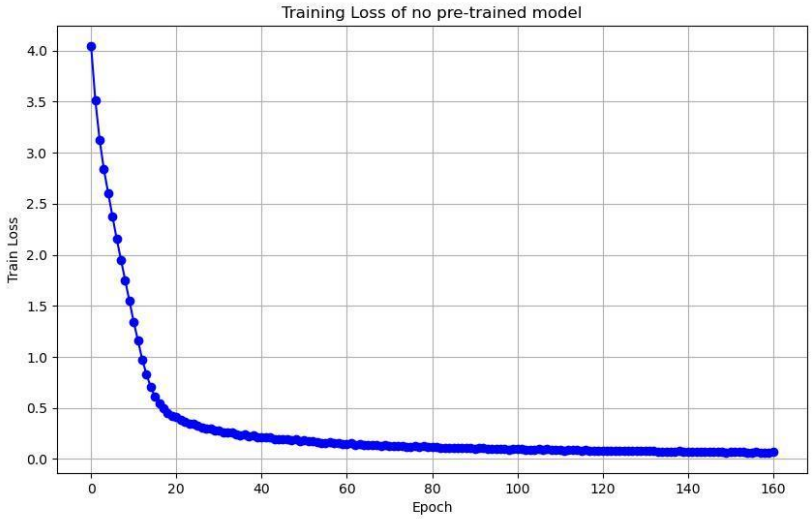

图 3.无预训练模型的训练损失曲线

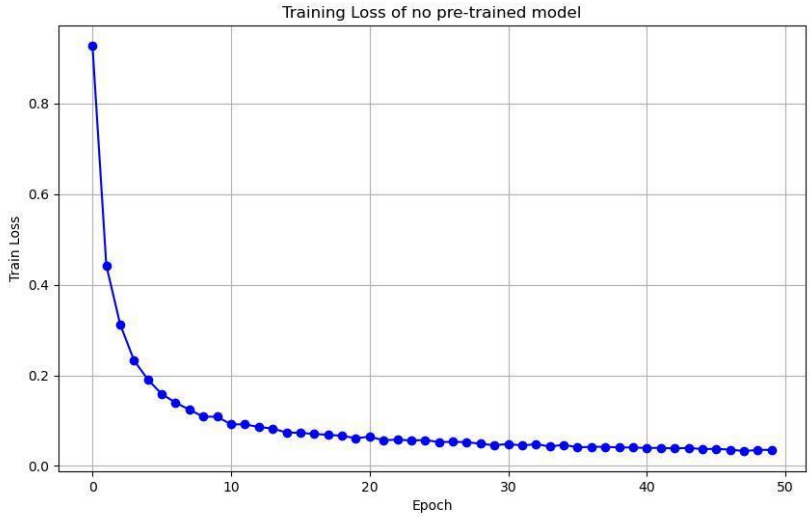

图 4.预训练模型的训练损失曲线

综合考虑训练效率和成本，我们观察到未经预训练的模型在训练 160 个 epoch 后基本收敛，因此决定停止训练。对于预训练模型，同样的情况在 50 个 epoch 后出现，因此也相应地停止了训练。

3.4 测试结果与分析

3.4.1 未预训练模型

从图 3 的训练损失曲线可以看出，模型在前 20 个 epoch 的训练中损失迅速下降，之后逐渐趋于平稳，表明模型在不断调整参数以适应训练数据，最终达到收敛状态。

训练后，我们绘制了 Top-1 准确率的变化情况，图 5 是未预训练模型的 Top-1 准确率 ACC@1 曲线：

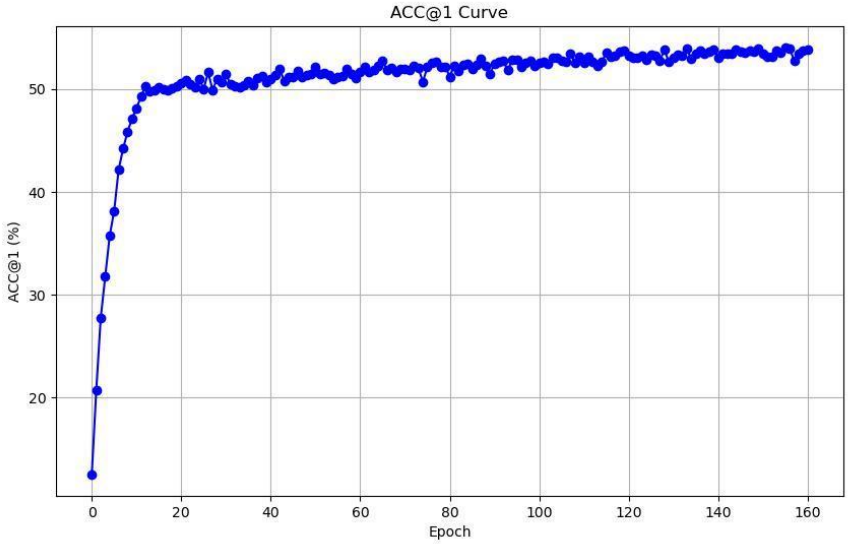

图 5.无预训练模型的 ACC@1 曲线

在 Top-1 准确率曲线中，前 20 个 epoch 随着训练的进行，模型的 Top-1 准确率从 12%迅速提升至 50%，这一点与模型前 20 个 epoch 的训练损失迅速下降相对应。在训练的中后期，模型的准确率逐渐上升，并经过 160 个 epoch 的训练，最终达到 54%的水平。实际上，停止训练时模型的准确率仍在缓慢上升。但是，可以预见的是，对于未经预训练的模型，要想达到较高的准确率，需要耗费大量的算力、时间，以及精细的参数调试，这并非易事。因此，直接使用预训练模型是合理的选择。

对于上面的现象，可能是由于未经预训练的模型在从头开始学习时，需要通过大量的训练数据和时间来逐步调整其参数，以找到最佳的特征表示。对于 CIFAR-100 这样的较小数据集，模型在初期能够快速学习到一些基本特征，因此准确率会迅速上升。然而，随着训练的深入，我推测准确率提升并逐渐停滞可能源自模型面临的以下几个挑战：

1.数据集规模限制：CIFAR-100 数据集的样本量相对较小，包含的信息和多样性有限。未经预训练的模型在初期学习到基础特征后，需要更多的数据来进一步提升性能。然而，受限于数据集的规模，模型无法获得足够的新的信息来持续提高准确率。

2.模型复杂度与数据匹配：Swin-v2 模型具有较高的复杂度，包含大量的参数。对于一个较小的数据集来说，这样的模型在训练的中后期，模型可能更多地拟合训练数据的噪声和细节，而不是学习到有用的泛化特征，这导致了准确率的提升变得缓慢甚至停滞。

3.缺乏预训练特征：预训练模型通常在大规模数据集上训练，学到了丰富且通用的特征表示，这些特征在迁移到新的任务或数据集时具有很大的优势。而未经预训练的模型需要从零开始学习所有特征，这一过程不仅缓慢，还容易陷入局部最优解，导致最终的准确率不高。

我们调用保存的最优模型在测试集上进行了测试，并随机选择了 10 张图片

进行可视化，如图 6。它展示了该模型在测试集上随机抽取的部分预测结果，其中包含了模型的正确预测和错误预测。图中标注了每张图片的真实标签和模型的预测标签，其中“True”表示样本的真实类别，“Pred”表示模型对该样本的预测类别。从中可以看出模型在大部分情况下能够正确分类，但也存在一定的误差。

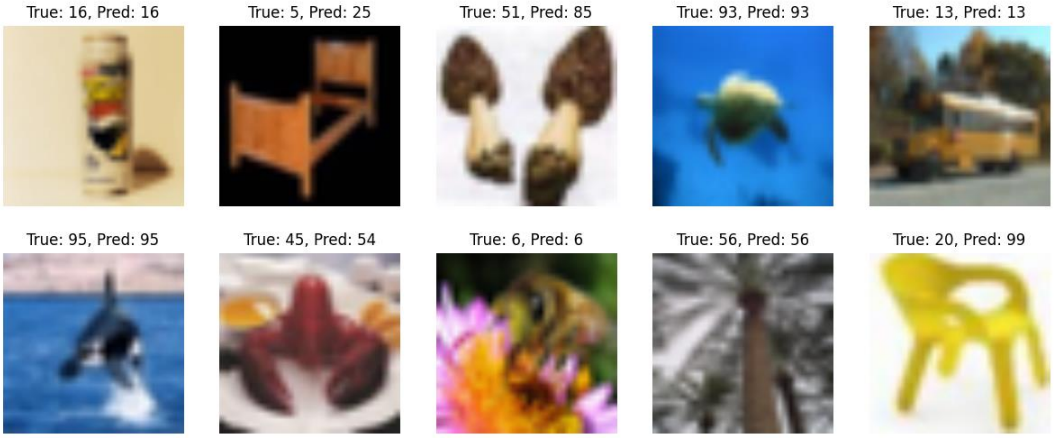

图 6.无预训练分类测试结果可视化

3.4.2 预训练模型

从图 4 的训练损失曲线可以看出，模型在前 10 个 epoch 的训练中损失迅速下降，之后逐渐达到相对稳定水平。

和上一节相同，训练后，我们绘制了预训练模型的 Top-1 准确率的变化情况，图 7 是它的 Top-1 准确率 ACC@1 曲线：

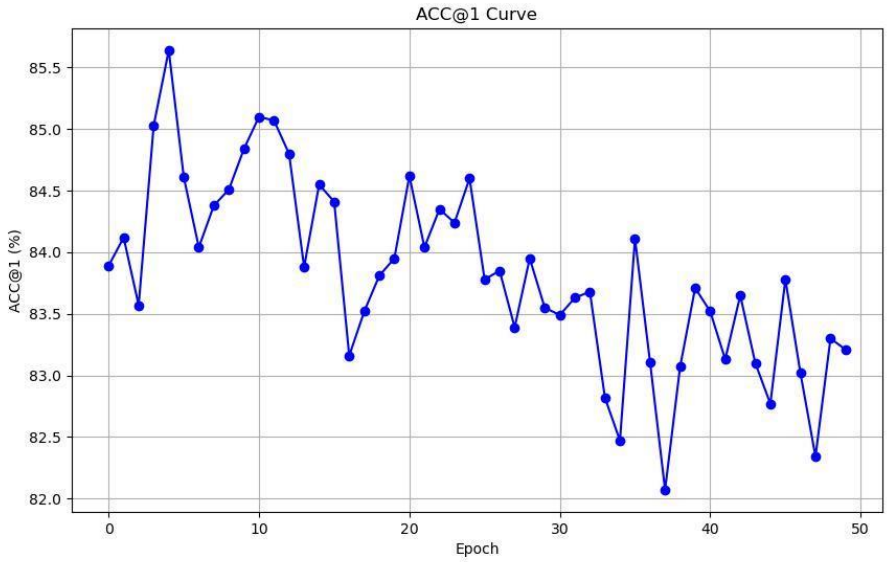

图 7.无预训练模型的 ACC@1 曲线

与未经预训练的模型完全不同，使用预训练模型后，Top-1 准确率曲线显示模型从一开始就具备相当高的准确率（83.5%），并且在前 5 个 epoch 迅速上

升至 85.64%。然而，随着训练的进行，虽然训练损失在下降，但是模型的准确率出现了波动下降的现象，甚至在第 38 个 epoch 时达到了最低值（82.07%），这是典型的过拟合现象。因此，及时停止训练是一个明智的选择。

这一现象的出现可以归因于预训练模型在小规模数据集上的表现。预训练模型通常在大规模数据集（如 ImageNet-1k）上进行训练，这些数据集包含了丰富的特征和复杂的模式，使得模型能够学习到通用的特征表示。然而，当这些预训练模型在较小规模的数据集（如 CIFAR-100）上进行微调时，模型容易过度拟合训练数据的特征，而未能很好地泛化到测试数据。此外，预训练模型在大规模数据集上学到的特征在小规模数据集上可能并不完全适用，进一步加剧了过拟合的风险。

与上一节相同，我们调用了保存的最优模型，也就是第 5 个 epoch 保存的模型在测试集上进行了测试，也随机选择了 10 张图片进行可视化，如图 8。从中可以看出，随机抽取的样本模型全部进行了正确的分类，这也与 85.6%的正确率基本一致。

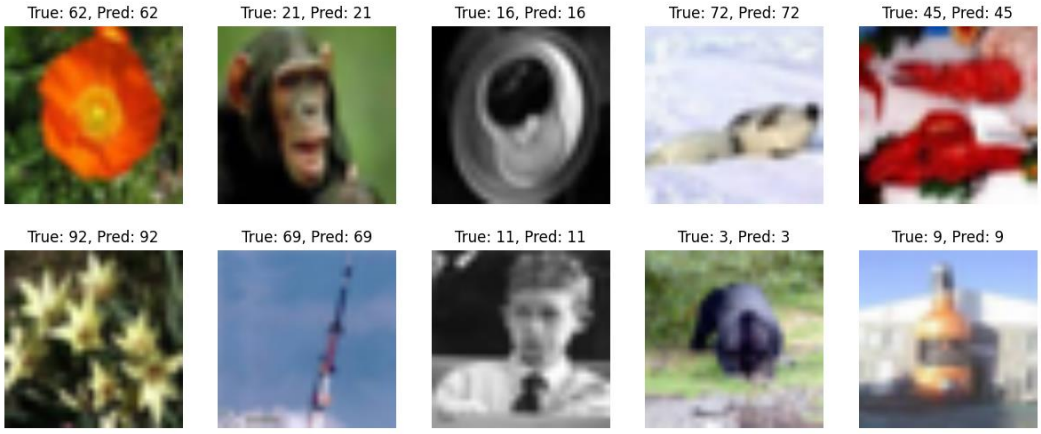

图 8.无预训练分类测试结果可视化

3.4.3 与基准模型的对比

在本次实验中，我们将 Swin-v2 模型的训练结果与多个基准模型在 CIFAR-100 数据集上的表现进行了对比。需要注意的是，其他 baseline 和本次实验一样为该模型最基本实现，并没有使用额外的辅助模块。不过，和本次实验的预训练方式一样，它们都使用了 imagenet1k 预训练数据。表 2 是各模型的 Top-1 准确率对比结果。

Method	Transformer?	CIFAR-100
VGG16	N	67.0
Resnet18		81.9
Resnet50		81.3
ViT-S/16	Y	83.2
ViT-B/16		86.5
Swin-T		86.7
Swin-S		88.0
Swin-B		88.3
CLIP-ViT-B/16		80.2
Swin-v2(no-pretrained)		**54.05**
Swin-v2(pretrained)		**85.64**

表 2.Swin-v2 与 baseline 在 CIFAR-100 数据集的表现对比

表格显示，未经预训练的 Swin-v2 模型在 CIFAR-100 数据集上的 Top-1 准确率为 54.05%，显著低于其他经过预训练的基准模型，从中可以看出分类任务中模型预训练的重要性。而经过预训练的 Swin-v2 模型在 CIFAR-100 数据集上的 Top-1ACC@1 准确率达到了 85.64%，显著优于大多数传统卷积神经网络模型（如 VGG16 和 Resnet 系列），接近或超过其他基于 Transformer 的模型（如 ViT 系列和 CLIP-ViT-B/16），这应该是由于 Swin-v2 本身出色的性能。

不过，需要注意的是，Swin-v2 在第一代版本上进行了改进，理论上会取得更好的效果，但是从表格的结果来看　其表现仍差于 Swin Transformer v1 的版本。出现这一现象的原因可能包括以下几个方面：

1.优化算法和训练参数的调试：Swin-v2 的训练参数（如学习率、优化器、epoch 数量等）可能需要进行更多的调整和优化，以适应 CIFAR-100 数据集的特性，而具体的调参需要经验、技巧和时间。因此，在当前的实验设置下，模型的训练参数未必是最优的，这也可能导致性能不如预期。

2.模型复杂度和适应性：Swin-v2 模型在设计上引入了更多的复杂性，如余弦相似度注意力和对数相对位置偏置等。这些改进虽然在理论上提升了模型的表达能力，但在实际应用中，比如特定的数据集，可能无法充分发挥其优势，甚至可能导致过拟合或训练不稳定。

4 结论

在这篇报告中，我们探讨了利用 Swin-v2 模型在 CIFAR-100 数据集上实现目标分类的过程与效果。通过实验，我们对未经预训练和预训练的 Swin-v2 模型进行了详细的训练和评估，并分析了其在图像分类任务中的表现。

首先，在未经预训练的模型训练中，尽管模型在前 20 个 epoch 内准确率从 12%迅速提升至 50%，但其后的提升变得缓慢，并最终在 160 个 epoch 后达到

54.05%的准确率。这一结果反映了从头开始训练模型所面临的挑战，包括数据集规模限制、模型复杂度高、以及参数调试难度大等。未经预训练的模型在性能上不及预期，说明其要达到较高的准确率，需要耗费大量的算力、时间和精细的参数调试。

相比之下，预训练的 Swin-v2 模型在 50 个 epoch 的训练后，准确率迅速达到了 85.64%。预训练模型在初期就具备较高的准确率，并在前期迅速上升。然而，随着训练的进行，准确率出现波动下降的现象，这是典型的过拟合问题。最终，尽管 Swin-v2 的表现略低于 Swin-v1 版本，但其预训练的优势依然明显。

通过与其他基准模型的对比分析，预训练的 Swin-v2 模型在较短时间内表现出色，准确率显著优于传统卷积神经网络模型，并且接近甚至超过其他基于 Transformer 架构的模型。预训练不仅提高了模型的训练效率，减少了计算资源的消耗，还有效提升了模型的最终性能。

5 参考文献

[1] Galassi, A., Lippi, M., & Torroni, P. (2020). Attention in natural language processing. *IEEE transactions on neural networks and learning systems*, *32*(10), 4291-4308.

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

[3] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[4] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10012-10022).

[5] Liu, Z., Hu, H., Lin, Y., Yao, Z., Xie, Z., Wei, Y., ... & Guo, B. (2022). Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12009-12019).

[6] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[7] Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., ... & Zhu, J. (2021). Pre-trained models: Past, present and future. *AI Open*, *2*, 225-250.

[8] Huggingface,(2023,12,21)swinv2-base-patch4-window8-256. https://huggingface.co/microsoft/swinv2-base-patch4-window8-256

附件

附件 1 无预训练模型的训练测试代码

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from transformers import Swinv2Config, Swinv2ForImageClassification
from tqdm import tqdm
import time
import logging
import os
import random
import matplotlib.pyplot as plt
import numpy as np


def configure_logging():
    os.makedirs('/root/swin/results', exist_ok=True)
    os.makedirs('/root/swin/results/vis', exist_ok=True)
    logging.basicConfig(filename='/root/swin/results/training_log.log',
level=logging.INFO,
                        format='%(asctime)s %(message)s')


def get_device():
    return torch.device("cuda" if torch.cuda.is_available() else "cpu")


def get_transforms():
    cifar100_mean = [0.5071, 0.4867, 0.4408]
    cifar100_std = [0.2675, 0.2565, 0.2761]
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=cifar100_mean, std=cifar100_std),
    ])


def get_data_loaders(transform):
    train_dataset = datasets.CIFAR100(root='./data', train=True,
transform=transform, download=True)
    test_dataset = datasets.CIFAR100(root='./data', train=False,
transform=transform, download=True)
    train_loader = DataLoader(dataset=train_dataset, batch_size=64,
shuffle=True)
    test_loader = DataLoader(dataset=test_dataset, batch_size=64,
shuffle=False)
    return train_loader, test_loader


def initialize_model(device):
    config = Swinv2Config(num_labels=100)  # CIFAR-100 有 100 个类别
    model = Swinv2ForImageClassification(config)
    model = model.to(device)
    return model
```

```python
def load_checkpoint(model, optimizer, checkpoint_path):
    start_epoch = 0
    best_acc1 = 0
    if os.path.exists(checkpoint_path):
        checkpoint = torch.load(checkpoint_path)
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        start_epoch = checkpoint['epoch'] + 1
        best_acc1 = checkpoint['best_acc1']
        logging.info(f'Loaded checkpoint from epoch {start_epoch}')
    return start_epoch, best_acc1


def visualize_results(images, labels, predictions, epoch, cifar100_mean,
cifar100_std):
    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    for i in range(10):
        image = images[i].cpu().numpy().transpose((1, 2, 0))
        mean = np.array(cifar100_mean)
        std = np.array(cifar100_std)
        image = std * image + mean
        image = np.clip(image, 0, 1)
        ax = axes[i // 5, i % 5]
        ax.imshow(image)
        ax.set_title(f'True: {labels[i]}, Pred: {predictions[i]}')
        ax.axis('off')
    plt.savefig(f'/root/swin/results/vis/epoch_{epoch}.png')
    plt.close()


def evaluate_model(model, test_loader, criterion, epoch, device,
cifar100_mean, cifar100_std):
    model.eval()
    correct = 0
    total = 0
    running_test_loss = 0.0
    all_images, all_labels, all_predictions = [], [], []
    with torch.no_grad():
        for images, labels in tqdm(test_loader, desc='Evaluating'):
            images, labels = images.to(device), labels.to(device)
            outputs = model(images).logits
            loss = criterion(outputs, labels)
            running_test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # 收集所有图片、标签和预测值
           all_images.extend(images.cpu())
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    acc1 = 100 * correct / total
    avg_test_loss = running_test_loss / len(test_loader)
    logging.info(f'Test Loss: {avg_test_loss:.4f}, ACC@1: {acc1:.2f}%')
    print(f'ACC@1 of the model on the CIFAR-100 test images: {acc1:.2f}%')

    # 随机选择 10 张图片进行可视化
```

```
    if len(all_images) > 0:
        indices = random.sample(range(len(all_images)), 10)
        selected_images = [all_images[i] for i in indices]
        selected_labels = [all_labels[i] for i in indices]
        selected_predictions = [all_predictions[i] for i in indices]
        visualize_results(selected_images, selected_labels,
selected_predictions, epoch, cifar100_mean, cifar100_std)

    return acc1


def train_model(model, train_loader, test_loader, criterion, optimizer,
num_epochs, eval_interval, start_epoch, device,
                cifar100_mean, cifar100_std):
    global best_acc1
    logging.info(f'Starting training at: {time.time()}')

    for epoch in range(start_epoch, num_epochs):
        model.train()
        running_loss = 0.0
        start_time = time.time()

        # 创建一个进度条
       with tqdm(total=len(train_loader), desc=f'Epoch
{epoch}/{num_epochs}', unit='batch') as pbar:
            for images, labels in train_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images).logits
                loss = criterion(outputs, labels)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                running_loss += loss.item()
                pbar.set_postfix({'loss': running_loss / (pbar.n + 1)})
                pbar.update(1)

        epoch_time = time.time() - start_time
        remaining_time = epoch_time * (num_epochs - epoch - 1)
        avg_loss = running_loss / len(train_loader)
        logging.info(
            f'Epoch [{epoch}/{num_epochs}], Loss: {avg_loss:.4f}, Time:
{epoch_time:.2f}s, Remaining time: {remaining_time / 60:.2f}min')
        print(
            f"Epoch [{epoch}/{num_epochs}], Loss: {avg_loss:.4f}, Time:
{epoch_time:.2f}s, Remaining time: {remaining_time / 60:.2f}min")

        # 每 eval_interval 个 epoch 进行一次测试
       if (epoch) % eval_interval == 0 or (epoch) == num_epochs:
            print("Testing......")
            acc1 = evaluate_model(model, test_loader, criterion, epoch,
device, cifar100_mean, cifar100_std)
            # 保存最优模型
           if acc1 > best_acc1:
                best_acc1 = acc1
                model.save_pretrained('/root/swin/results/best_model')
                logging.info(f'Saved best model with ACC@1: {acc1:.2f}%')
```

```
        # 保存 checkpoint
        checkpoint_path = '/root/swin/results/checkpoint.pth'
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'best_acc1': best_acc1
        }, checkpoint_path)
        logging.info(f'Saved checkpoint at epoch {epoch}')


def main():
    configure_logging()
    device = get_device()
    transform = get_transforms()
    train_loader, test_loader = get_data_loaders(transform)
    model = initialize_model(device)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
    checkpoint_path = '/root/swin/results/checkpoint.pth'
    start_epoch, best_acc1 = load_checkpoint(model, optimizer,
checkpoint_path)
    train_model(model, train_loader, test_loader, criterion, optimizer,
num_epochs=1000, eval_interval=1,
                start_epoch=start_epoch, device=device,
cifar100_mean=[0.5071, 0.4867, 0.4408],
                cifar100_std=[0.2675, 0.2565, 0.2761])


if __name__ == "__main__":
    main()
```

附件 2 使用 huggingface 官网的 Swin-v2 预训练模型的训练测试代码

在运行代码前，需要在 huggingface 网站（https://huggingface.co/microsoft/swinv2-base-patch4-window8-256）手动下载预训练模型至本地，并修改 load_model 函数中的地址。

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from transformers import Swinv2ForImageClassification, AutoFeatureExtractor
from tqdm import tqdm
import time
import logging
import os
import random
import matplotlib.pyplot as plt
import numpy as np


def configure_logging():
    os.makedirs('/root/swin/results', exist_ok=True)
    os.makedirs('/root/swin/results/vis', exist_ok=True)
```

```
    logging.basicConfig(filename='/root/swin/results/training_log.log',
level=logging.INFO,
                        format='%(asctime)s %(message)s')


def get_device():
    return torch.device("cuda" if torch.cuda.is_available() else "cpu")


def get_transforms():
    cifar100_mean = [0.5071, 0.4867, 0.4408]
    cifar100_std = [0.2675, 0.2565, 0.2761]
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=cifar100_mean, std=cifar100_std),
    ])


def get_data_loaders(transform):
    train_dataset = datasets.CIFAR100(root='./data', train=True,
transform=transform, download=True)
    test_dataset = datasets.CIFAR100(root='./data', train=False,
transform=transform, download=True)
    train_loader = DataLoader(dataset=train_dataset, batch_size=20,
shuffle=True)
    test_loader = DataLoader(dataset=test_dataset, batch_size=20,
shuffle=False)
    return train_loader, test_loader


def load_model(device):
    model = Swinv2ForImageClassification.from_pretrained(
        "/root/swin/model/",  # 指向包含 config.json 和 pytorch_model.bin 的
目录
       ignore_mismatched_sizes=True
    )
    model.classifier = torch.nn.Linear(model.classifier.in_features, 100)
# CIFAR-100 有 100 个类别
    model = model.to(device)
    return model


def load_checkpoint(model, optimizer, checkpoint_path):
    start_epoch = 0
    best_acc1 = 0
    if os.path.exists(checkpoint_path):
        checkpoint = torch.load(checkpoint_path)
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        start_epoch = checkpoint['epoch'] + 1
        best_acc1 = checkpoint['best_acc1']
        logging.info(f'Loaded checkpoint from epoch {start_epoch}')
    return start_epoch, best_acc1


def visualize_results(images, labels, predictions, epoch, cifar100_mean,
cifar100_std):
    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
```

```
    for i in range(10):
        image = images[i].cpu().numpy().transpose((1, 2, 0))
        mean = np.array(cifar100_mean)
        std = np.array(cifar100_std)
        image = std * image + mean
        image = np.clip(image, 0, 1)
        ax = axes[i // 5, i % 5]
        ax.imshow(image)
        ax.set_title(f'True: {labels[i]}, Pred: {predictions[i]}')
        ax.axis('off')
    plt.savefig(f'/root/swin/results/vis/epoch_{epoch}.png')
    plt.close()


def evaluate_model(model, test_loader, criterion, epoch, device,
cifar100_mean, cifar100_std):
    model.eval()
    correct = 0
    total = 0
    running_test_loss = 0.0
    all_images, all_labels, all_predictions = [], [], []
    with torch.no_grad():
        for images, labels in tqdm(test_loader, desc='Evaluating'):
            images, labels = images.to(device), labels.to(device)
            outputs = model(images).logits
            loss = criterion(outputs, labels)
            running_test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # 收集所有图片、标签和预测值
           all_images.extend(images.cpu())
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    acc1 = 100 * correct / total
    avg_test_loss = running_test_loss / len(test_loader)
    logging.info(f'Test Loss: {avg_test_loss:.4f}, ACC@1: {acc1:.2f}%')
    print(f'ACC@1 of the model on the CIFAR-100 test images: {acc1:.2f}%')

    # 随机选择 10 张图片进行可视化
    if len(all_images) > 0:
        indices = random.sample(range(len(all_images)), 10)
        selected_images = [all_images[i] for i in indices]
        selected_labels = [all_labels[i] for i in indices]
        selected_predictions = [all_predictions[i] for i in indices]
        visualize_results(selected_images, selected_labels,
selected_predictions, epoch, cifar100_mean, cifar100_std)

    return acc1


def train_model(model, train_loader, test_loader, criterion, optimizer,
num_epochs, eval_interval, start_epoch, device,
                cifar100_mean, cifar100_std):
    global best_acc1
    logging.info(f'Starting training at: {time.time()}')
```

```
    for epoch in range(start_epoch, num_epochs):
        model.train()
        running_loss = 0.0
        start_time = time.time()

        # 创建一个进度条
       with tqdm(total=len(train_loader), desc=f'Epoch
{epoch}/{num_epochs}', unit='batch') as pbar:
            for images, labels in train_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images).logits
                loss = criterion(outputs, labels)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                running_loss += loss.item()
                pbar.set_postfix({'loss': running_loss / (pbar.n + 1)})
                pbar.update(1)

        epoch_time = time.time() - start_time
        remaining_time = epoch_time * (num_epochs - epoch - 1)
        avg_loss = running_loss / len(train_loader)
        logging.info(
            f'Epoch [{epoch}/{num_epochs}], Loss: {avg_loss:.4f}, Time:
{epoch_time:.2f}s, Remaining time: {remaining_time / 60:.2f}min')
        print(
            f"Epoch [{epoch}/{num_epochs}], Loss: {avg_loss:.4f}, Time:
{epoch_time:.2f}s, Remaining time: {remaining_time / 60:.2f}min")

        # 每 eval_interval 个 epoch 进行一次测试
       if (epoch) % eval_interval == 0 or (epoch) == num_epochs:
            print("Testing......")
            acc1 = evaluate_model(model, test_loader, criterion, epoch,
device, cifar100_mean, cifar100_std)
            # 保存最优模型
           if acc1 > best_acc1:
                best_acc1 = acc1
                model.save_pretrained('/root/swin/results/best_model')
                logging.info(f'Saved best model with ACC@1: {acc1:.2f}%')

        # 保存 checkpoint
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'best_acc1': best_acc1
        }, checkpoint_path)
        logging.info(f'Saved checkpoint at epoch {epoch}')


def main():
    configure_logging()
    device = get_device()
    transform = get_transforms()
    train_loader, test_loader = get_data_loaders(transform)
    model = load_model(device)
```

```
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
    checkpoint_path = '/root/swin/results/checkpoint.pth'
    start_epoch, best_acc1 = load_checkpoint(model, optimizer,
checkpoint_path)
    train_model(model, train_loader, test_loader, criterion, optimizer,
num_epochs=20, eval_interval=1,
                start_epoch=start_epoch, device=device,
cifar100_mean=[0.5071, 0.4867, 0.4408],
                cifar100_std=[0.2675, 0.2565, 0.2761])


if __name__ == "__main__":
    main()
```