基于 TinyViT 与 ResNet18 的图像分类模型实验

杨竞舟 2025011317

2025 年 11 月 18 日

摘要： 采用 TinyViT 与 ResNet18 模型，在 CIFAR-10 数据集上进行图像分类实验，比较两种模型的性能表现。分为从零训练和预训练微调两种方式，评估损失下降和训练准确度。通过实验，学习 ResNet 和 Transformer 经典模型的原理结构，掌握其在图像分类任务中的应用方法。评估 TinyVit 和 ResNet18 在 CIFAR-10 数据集上的表现差异，分析各自的优缺点。

开源 GitHub 代码仓库地址：https://github.com/heroyjz/tinyvit-vs-resnet-cifar10

关键字： Vit；TinyVit；ResNet；预训练微调；蒸馏

目录

一 实验目的

- 学习 ResNet 架构
- 学习 transformer 架构
- 理解 transformer 在视觉领域的应用：ViT
- 动手使用 TinyViT 和 ResNet18，在个人 PC(laptop RTX-4070)上做小规模图像数据集(CIFAR-10)训练
- 评估指标和知识总结

二 实验环境

硬件

- CPU：Intel(R) Core(TM) i7-14700HX（WSL2 上可见 8 线程）
- GPU：NVIDIA GeForce RTX 4070 Laptop，驱动 581.42，CUDA 13.0
- 内存：≥16 GB（本地实验在 32 GB RAM 上完成）

系统

- Windows 11 + WSL2，内核 `Linux 6.6.87.2-microsoft-standard-WSL2`
- glibc 2.39，`bash` 终端
- Conda 虚拟环境 `vit`

软件

- Python 3.11.7
- PyTorch 2.9.0+cu130（CUDA runtime 13.0，cuDNN 91300），TorchVision 0.24.0+cu130
- library: `timm`、`matplotlib`、`pandas`、`tensorboard`（可选，用于日志/可视化）

三 实验内容

3.1 CNN 基线（ResNet-18）

- 目标：以经典卷积残差网络 ResNet-18 作为参照，衡量在小型数据集上“微调 vs. 从零训练”的效果差异。
- 方法：分别加载 ImageNet-1k 预训练权重和随机初始化权重，在 CIFAR-10 上以相同的数据增强、优化器和训练计划进行 30 轮训练。预训练版本作为迁移学习基线，随机初始化版本用于衡量纯 CIFAR 训练的上限。
- 参数选择：输入分辨率统一为 224×224，以便与 TinyViT 保持一致；优化器选用 AdamW + Cosine LR（初始学习率 3e-4、weight decay 0.05），结合 RandAugment 与 RandomResizeCrop，使 CNN 与 Transformer 在相同的正则化条件下比较；batch size 128 足以填满 RTX 4070 显存并确保统计稳定。

- 指标：关注训练/验证 loss、accuracy 曲线和最终测试精度，用于评估卷积架构在 CIFAR-10 上的收敛速度与泛化能力。

3.2 Transformer 对照（TinyViT-5M）

- 目标：在与 ResNet 完全一致的训练设置下，比较轻量级 Vision Transformer TinyViT 在预训练与从零训练两种模式下的表现，检验“预训练知识迁移”对 Transformer 的影响。
- 方法：选择 TinyViT-5M（先在 ImageNet-22k 上训练，再在 1k 上微调的**权重**），保持输入尺寸、batch size、优化器、数据增强等参数不变，分别进行微调与随机初始化训练。
- 参数选择：保持 30 个 epoch 的训练预算，记录 TinyViT 微调在前 6–10 个 epoch 的快速收敛，同时允许在需要时提前终止（考虑到 transformer 在普通 GPU 的运算复杂度）；学习率调度与 ResNet 相同，确保比较指标一致。
- 指标：和 ResNet 一样关注 train/val loss、accuracy，并额外分析 TinyViT 在相同训练预算下的吞吐量与算力开销，展示 Transformer 在小数据集上的效率差异与精度优势。

3.3 可视化与案例分析

- 为每次运行输出 `metrics.csv`、`config.json`、`best.pt`、`last.pt`，确保实验可复现。
- 通过编写可视化脚本，生成 Train/Val Loss 与 Accuracy 的四象限对比图，突出四种设置（ResNet/TinyViT × 预训练/从零）的差异，并在曲线末尾标注最终数值。
- 利用展示脚本随机抽取 CIFAR-10 图像展示实际预测结果，配合图表形成完整的定量+定性分析。

通过以上设置，可以在相同的硬件预算与训练超参数下比较 Transformer 与 CNN 的迁移学习效果，清晰说明预训练对 TinyViT 的收益，并用数据/图表支持“在小型数据集上是否值得使用 ViT”这一问题。

四　Models

4.1 基础 1：ResNet

ResNet [1] 是为了解决传统卷积神经网络中的**深度失效**问题，由 XX 提出的一种神经网络架构。其核心思想是引入**残差连接**（Residual Connections），通过跳跃连接将输入直接传递到后续层，形成所谓的“残差块”（Residual Blocks）。这种设计使得网络能够学习到恒等映射，从而缓解了随着网络加深而导致的梯度消失和梯度爆炸问题。

4.1.1 梯度消失与爆炸

设一个没有残差连接的深层网络，其第 l 层到第 $l+1$ 层的映射为

$$x_{\{l+1\}} = f_l(x_l), l = 0, ..., L-1$$

损失函数为 $\mathcal{L}$。在反向传播时，第 l 层处的梯度为

$$\frac{\partial \mathcal{L}}{\partial x_l} = \frac{\partial \mathcal{L}}{\partial x_L} \prod_{k=l}^{L-1} \frac{\partial x_{k+1}}{\partial x_k} = \frac{\partial \mathcal{L}}{\partial x_L} \prod_{k=l}^{L-1} J_k$$

其中 $J_k = \frac{\partial x_{k+1}}{\partial x_k}$ 为第 k 层的雅可比矩阵。

在网络足够深时，多个雅可比矩阵相乘会导致以下问题：

- 若这些矩阵的谱半径（例如最大奇异值）普遍小于 1，则连乘结果的范数迅速衰减，出现**梯度消失**；
- 若谱半径普遍大于 1，则连乘结果迅速放大，出现**梯度爆炸**。

因此，在传统深层网络中，梯度在层与层之间的传播极易出现数值不稳定。

4.1.2 残差链接

ResNet 引入了残差块（residual block），将层的映射从直接学习 $H(x)$ 改为学习残差形式

$$H(x) = x + F(x, W)$$

其中：

- x 为块的输入；
- $F(x, W)$ 是若干卷积、批归一化（Batch Normalization）、非线性激活等运算的组合，称为“残差分支”；
- x 通过一条不作变换的**恒等短路连接（identity shortcut）**直接与$F(x, W)$的输出相加。

在这种结构下，残差块的前向传播可以写为

$$y = F(x, W) + x$$

网络整体不再是单纯的“堆叠非线性变换”，而是大量“输入 + 残差修正”的组合。为反向传播提供了一个 **ShortCut**。在反向传播时，考虑单个残差块对输入x的梯度为

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \left(\frac{\partial F(x, W)}{\partial x} + I \right),$$

在合适的初始化和归一化条件下（*He/Kaiming 正态初始化 & BatchNorm*），可以期望$\frac{\partial F(x)}{\partial x}$的“尺度”相对较小，从而$\frac{\partial F(x)}{\partial x} + I$ 的谱半径更接近 1。换言之，即便残差分支的梯度不稳定，**恒等项 $\boldsymbol{I}$**仍然为梯度提供了一条近似不衰减的直通路径，使得梯度可以更稳定地从深层传回浅层。

4.1.3 ResNet-18 结构

残差块里首先有 2 个有相同输出通道数的 3x3 卷积层。每个卷积层后接一个批量规范化层和 ReLU 激活函数。然后我们通过跨层数据通路，跳过这 2 个卷积运算，将输入直接加在最后的 ReLU 激活函数前。 这样的设计要求 2 个卷积层的输出与输入形状一样，从而使它们可以相加。如果想改变通道数，就需要引入一个额外的 1x1 卷积层来将输入变换成需要的形状后再做相加运算。

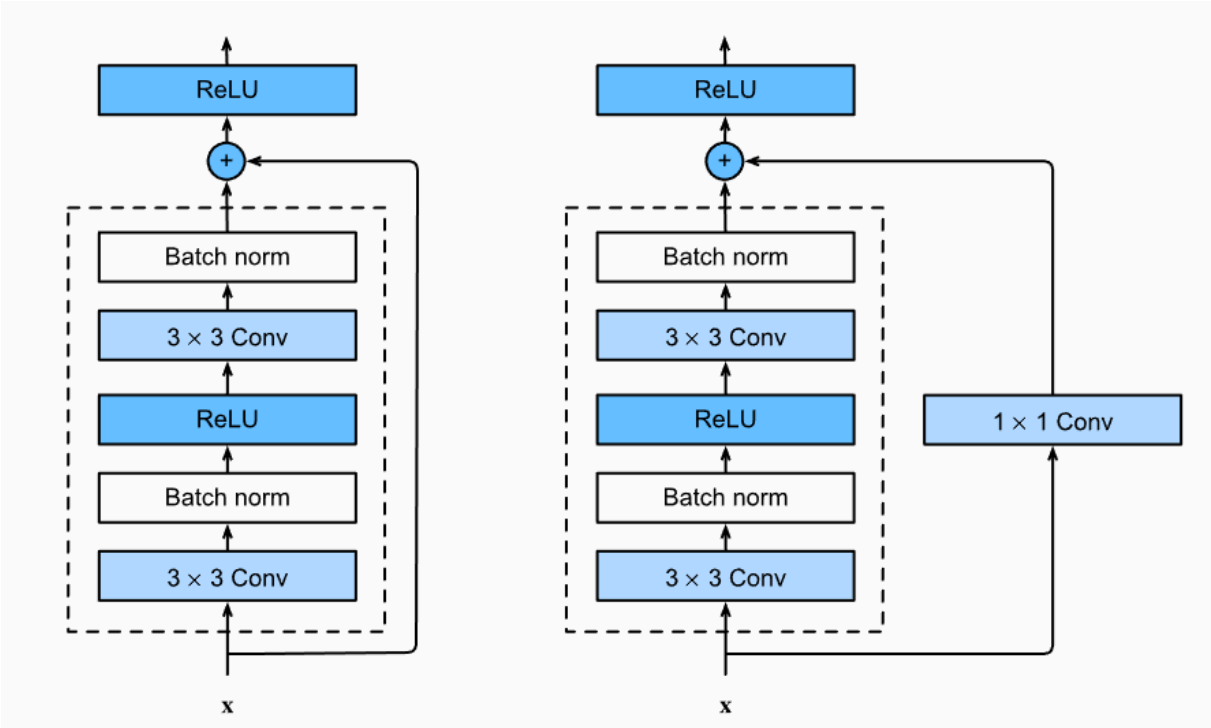

图 1　ResNet 残差块结构示意图

图 x 展示了 ResNet-18 的基本构建模块。ResNet-18 的输入由一个 7x7 卷积层和一个 3x3 最大池化层组成，用于初步提取图像特征并降低空间分辨率。残差卷积部分由 4 个阶段（stage）组成，每个阶段包含若干个残差块。每个阶段的第一个残差块可能会通过步幅为 2 的卷积来减小空间分辨率，同时增加通道数。具体结构如下：

- **Stage 1**: 2 个残差块，输出通道数 64
- **Stage 2**: 2 个残差块，输出通道数 128
- **Stage 3**: 2 个残差块，输出通道数 256
- **Stage 4**: 2 个残差块，输出通道数 512

在经过 4 个阶段后，ResNet-18 使用全局平均池化层将空间维度降至 1x1，然后通过一个全连接层输出最终的分类结果。该设计使其在保持较低计算复杂度的同时，能够有效地学习深层特征表示，广泛应用于各种计算机视觉任务中。

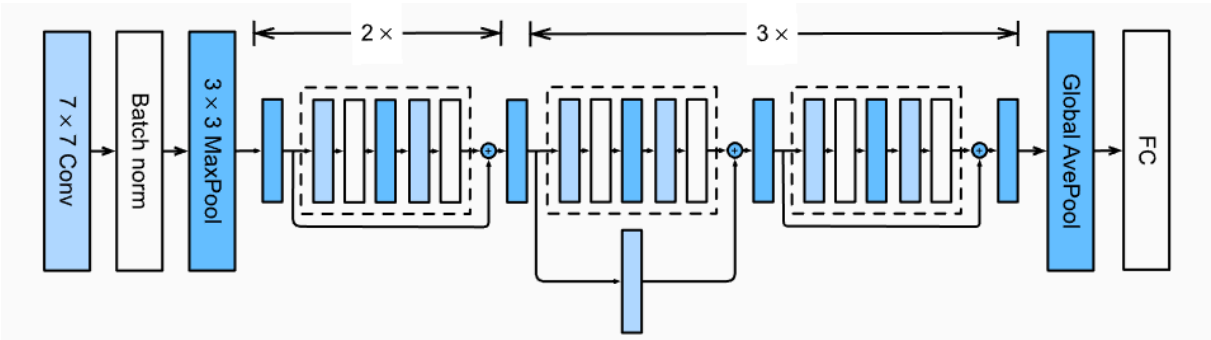

图 2　ResNet-18 网络结构示意图

4.2　基础 2：Transformer

在 transformer [2] 的论述中我只覆盖关注的核心原理。其细节部分可以参考 *Dive into Deep Learning* [3] 的 transformer 章节。或参考原文或其他海量互联网资料。

4.2.1　注意力机制

$$X \in R^{\{d\times n\}}$$

$$W_Q \in R^{\{d_k\times d\}},\quad W_K \in R^{\{d_k\times d\}},\quad W_V \in R^{\{d_v\times d\}}$$

$$Q = W_Q X \quad \left(Q \in R^{\{d_k\times n\}}\right)$$

$$K = W_K X \quad \left(K \in R^{\{d_k\times n\}}\right)$$

$$V = W_V X \quad \left(V \in R^{\{d_v\times n\}}\right)$$

$$S = \frac{Q^T K}{\sqrt{d_k}} \quad \left(S \in R^{\{n\times n\}}\right)$$

$$A = \mathrm{softmax}(S)$$

$$Y = V A^T \quad \left(Y \in R^{\{d_v\times n\}}\right)$$

$$\mathrm{Attention}(Q,K,V) = V\cdot \mathrm{softmax}\left(\frac{Q^T K}{\sqrt{d_k}}\right)^T$$

- X 是原始向量，如词嵌入或图像块嵌入矩阵
- Q（Query）、K（Key）、V（Value）是通过线性变换从 X 得到的表征空间矩阵
- 注意力机制通过计算 Q 和 K 的相似度来生成注意力权重 A，Softmax 确保权重归一化可微
- 注意力是权重对 V 的加权和，表征了输入 X（一段话或一幅图）在表征空间中的自相关性

4.2.2　多头注意力

多头注意力将 Q、K、V 分别线性映射为 h 组子空间，独立计算注意力，然后将 h 个输出拼接并线性变换回原始维度。这样做允许模型在不同子空间中捕捉不同的相关性模式，增强表达能力

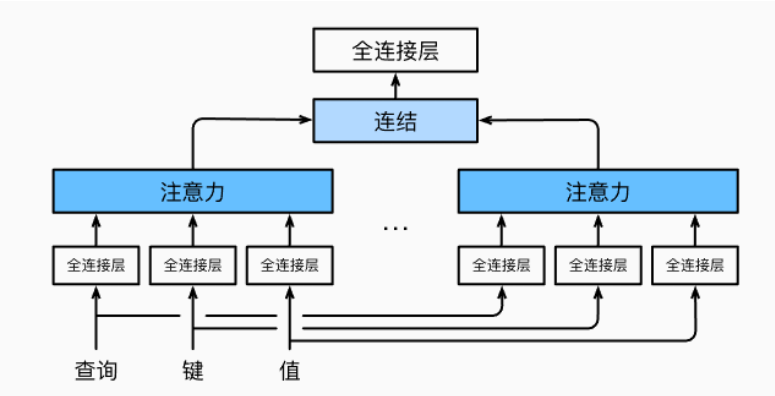

图 3　多头注意力结构图

4.2.3 Encoder Decoder

Transformer 原文

Encoder 架构由多头自注意力模块、前馈网络（MLP）模块、LayerNorm 和残差连接组成，其中：

- 多头自注意力模块用于在同一序列内部构建全局依赖关系，通过多头在不同子空间中并行建模多种“谁和谁相关”的模式，从而解决 RNN 长程依赖难建模、CNN 感受野有限的问题。
- 前馈网络（MLP）模块用于对每个位置的表征独立进行非线性变换，在不改变序列结构的前提下提高单点表示的表达能力，弥补注意力本身近似线性的不足。
- LayerNorm 用于对每个位置的特征做归一化，稳定各层输入的数值尺度，减小内部分布漂移，使深层网络在较大学习率下仍然易于优化。
- 残差连接用于在子层外部提供接近恒等映射的通路，使信息与梯度可以直接跨层传播，从而缓解网络加深带来的梯度消失和表示退化问题。

Decoder 架构由 masked 多头自注意力模块、encoder–decoder 多头注意力模块、前馈网络（MLP）模块、LayerNorm 和残差连接组成，其中：

- masked 多头自注意力模块用于在目标序列内部建模自回归依赖关系，通过对未来位置施加 mask，确保生成第 t 个 token 时仅依赖先前已生成的前缀序列。
- encoder–decoder 多头注意力模块用于以 Encoder 输出为键值（K/V），从源序列的编码表示中选择与当前解码位置最相关的信息，实现对输入条件的显式对齐与利用。
- 前馈网络（MLP）、LayerNorm 与残差连接在 Decoder 中的作用与在 Encoder 中相同，分别负责增强单点非线性表达、稳定数值分布与梯度、保证多层堆叠下的可训练性。

4.2.4 Decoder Only

尽管原始 Transformer 架构在多项任务中表现出色，但研究者发现，在某些情况下，仅使用解码器部分就能达到甚至超越完整 Transformer 的性能。这一发现导致了 Decoder-Only Transformer 架构的兴起，它在语言建模和生成任务中表现尤为出色。

Decoder-Only Transformer 相比完整的 Encoder-Decoder 结构有以下优势：

1. 参数效率：去除编码器后，模型参数量减少，但性能不减。
2. 训练简化：无需处理编码器-解码器之间的交互，简化了训练过程。
3. 自回归特性：天然适合语言生成任务。
4. 灵活性：可以轻松适应各种 NLP 任务，包括分类、生成和问答等。

下图可见，Decoder-Only 已经逐渐成为了大语言模型的主流架构选择。

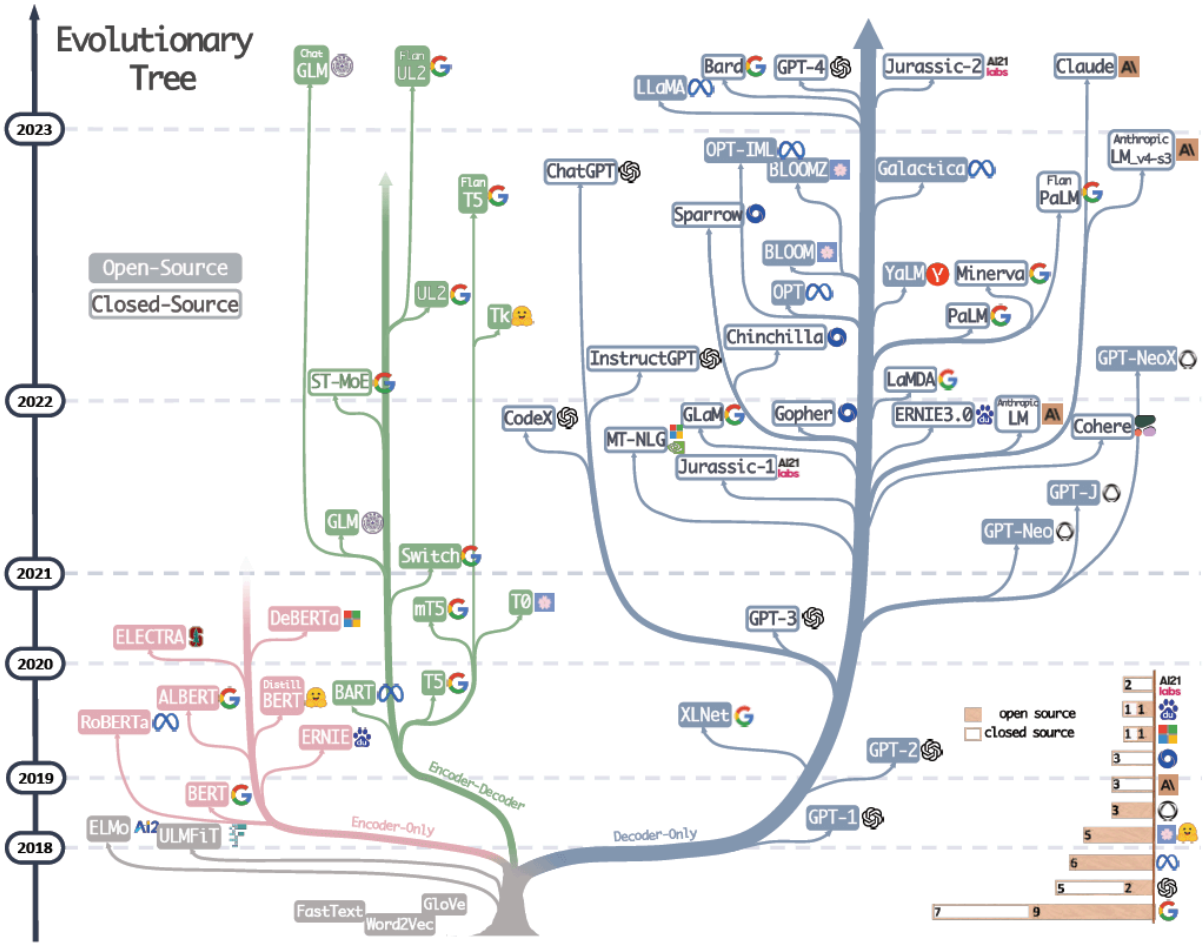

图 4　大语言模型的演变树

4.3　ViT & TinyViT

4.3.1　Vision Transformer (ViT)

ViT [4] 是 2020 年 Google 团队提出的将 Transformer 应用在图像分类的模型，虽然不是第一篇将 transformer 应用在视觉任务的论文，但是因为其模型“简单”且效果好，可扩展性强（scalable，模型越大效果越好），成为了 transformer 在 CV 领域应用的里程碑著作，也引爆了后续相关研究。

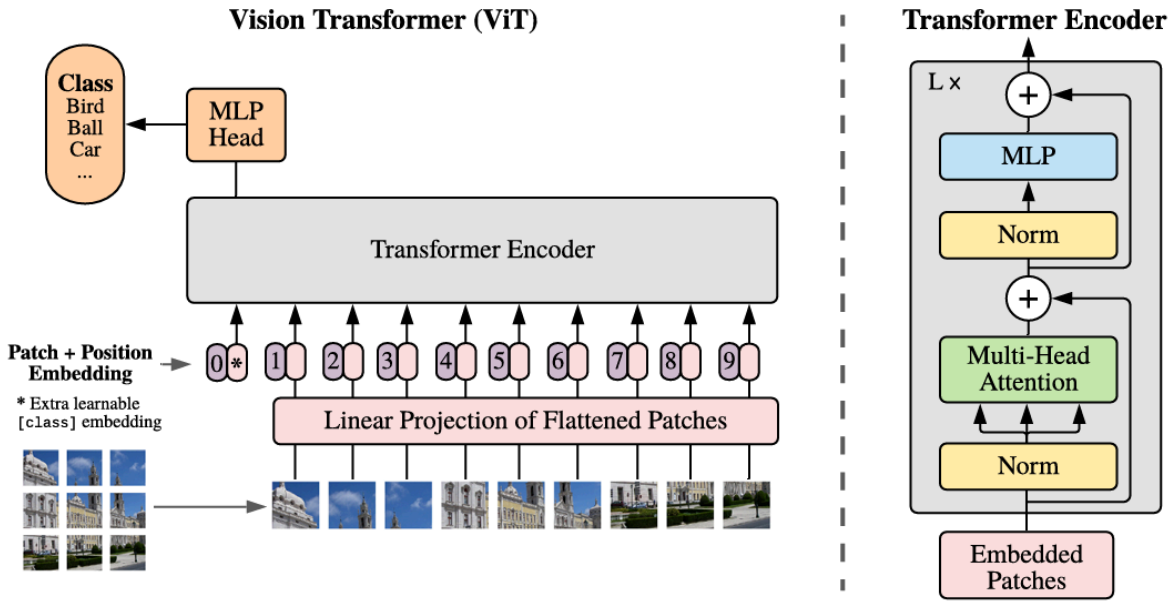

图 5　vit 结构示意图

如图，vit 将图像拆成 patch，每个 patch 展平并线性映射为一个向量，类似于 NLP 中的词嵌入。然后将这些 patch 嵌入序列输入到标准的 transformer encoder 中进行处理。为了让模型学习图像的空间位置信息，ViT 还引入了位置编码（positional encoding）。最后通过一个分类 token 的输出进行图像分类。

ViT 原论文中最核心的结论是，当拥有足够多的数据进行预训练的时候，ViT 的表现就会超过 CNN，突破 transformer 缺少归纳偏置的限制，可以在下游任务中获得较好的迁移效果。但是当训练数据集不够大的时候，ViT 的表现通常比同等大小的 ResNets 要差一些，因为 Transformer 和 CNN 相比缺少归纳偏置（inductive bias），即一种先验知识（局部相似性和平移不变形）。

4.3.2　TinyViT

TinyViT [5] 采用知识蒸馏方法提取了大模型的知识，并结合轻量级设计原则，构建了适合资源受限环境的高效视觉 Transformer 模型。本文旨在做较为初级的机器学习实验，故选择 TinyViT 作为 Transformer 的代表模型进行实验。不深究其蒸馏细节。下面是 Transformer 模型常见压缩方法概述

量化（Quantization）：将权重和/或激活由高精度浮点（如 FP32）压缩为低比特表示（如 FP16、INT8、INT4），在尽量保持模型精度的前提下显著减少存储和带宽占用，并利用低精度算力提高推理速度。常用方式包括后训练量化和量化感知训练。

剪枝（Pruning）：按照重要性指标删除部分参数或子结构，以降低计算量和参数量。非结构化剪枝产生稀疏权重，压缩率高但对硬件友好度有限；结构化剪枝以通道、注意力头、整层等为单位裁剪，更容易带来实际推理加速。

知识蒸馏（Knowledge Distillation）：通过大模型（Teacher）的输出分布或中间表征指导小模型（Student）训练，使小模型在显著较小的规模下逼近大模型性能。蒸馏常与量化、剪枝结合，用于弥补压缩带来的精度损失。

低秩分解与因子分解（Low-Rank / Factorization）：利用权重矩阵的冗余，将大矩阵近似分解为低秩矩阵乘积，如 $(W \approx AB)$，从而减少参数数目和矩阵乘法的计算复杂度。常应用于注意力投影和 MLP 权重。

参数共享与结构缩减（Parameter Sharing & Architecture Scaling）：通过在层间共享权重，或直接减少层数、注意力头数、隐藏维度等结构超参数，构造更紧凑的 Transformer 变体。通常配合知识蒸馏，以在较小结构下维持可接受的任务性能。

稀疏与结构化注意力（Sparse / Structured Attention）：对标准全连接注意力施加结构约束，仅在局部窗口、块结构或特定模式下计算注意力，减少 $(O(n^2))$ 的计算与显存开销。该类方法在长序列场景中尤为重要，可视为对注意力算子的结构级压缩。

五　Coding Design

本项目的编码设计围绕四类脚本展开：

- 训练脚本（`train_resnet_cifar.py`、`train_tinyvit_cifar.py`）
- 结果可视化脚本（`train_show_resnet.py`、`compare_runs.py`、`plot_four_panels.py`）
- 案例展示脚本（`show_resnet_example.py`）
- 环境/工具脚本（`check.py`、`env_setup.md`）

5.1　模块依赖

- **PyTorch（`torch`，`torchvision`）**：负责模型训练、自动求导、CIFAR-10 数据加载与 DataLoader 构建。`torch.cuda.amp` 提供 AMP 支持，`torch.utils.data.Subset` 用于可复现的 train/val 划分。
- **timm**：统一模型入口，通过 `timm.create_model` 创建 ResNet-18 与 TinyViT-5M，支持 `pretrained=True/False`、`num_classes=10` 等参数。
- **Matplotlib / Pandas**：解析 `metrics.csv` 并绘制 train/val loss 与 accuracy 曲线；`compare_runs.py`、`plot_four_panels.py` 与 `train_show_resnet.py` 均依赖二者。
- **其他工具**：`json` 负责保存运行配置；`argparse` 管理命令行参数；`csv` 用于训练阶段的指标落盘。

5.2　训练伪代码设计

整体训练流程伪代码如下

```
function main():
    args ← parse_args()                    # 读取外部配置（模型、预训练、超参）
    set_seed(args.seed)                    # 固定随机性
    device ← select_device(args.device)    # 优先使用 GPU

    (train_loader, val_loader, test_loader) ← create_dataloaders(args)   # 统一数据管线
    model ← create_model(args).to(device)   # timm 加载模型及权重
    criterion ← CrossEntropyLoss
    optimizer ← create_optimizer(model, args)
```

```
    scheduler ← create_scheduler(optimizer, args)
    scaler ← GradScaler(enabled = use_amp_on(device))  # AMP 混合精度

    best_val_acc ← 0
    for epoch in 1..args.epochs:            # 主训练循环
        train_stats ← train_one_epoch(model, train_loader, criterion,
                                      optimizer, device, scaler,
                                      epoch, args.log_interval,
                                      args.use_amp, args.mixup, args.cutmix)
        val_stats ← evaluate(model, val_loader, criterion, device)
        log_metrics(epoch, train_stats, val_stats, current_lr(optimizer))   # 写入 CSV
        save_checkpoint(last_ckpt, model, optimizer, scaler, args)          # 持久化 latest
        if val_stats.acc > best_val_acc:                                    # 维护 best
            best_val_acc ← val_stats.acc
            save_checkpoint(best_ckpt, model, optimizer, scaler, args)
        if scheduler ≠ None:
            scheduler.step()

    load_checkpoint(best_ckpt, model)      # 载入验证集最优权重
    test_stats ← evaluate(model, test_loader, criterion, device)
    report(test_stats)                     # 输出测试损失/精度/吞吐
```

单次 epoch 训练的代码如下

```
function train_one_epoch(...):
    model.train()
    init running_loss, running_correct, total_samples
    for each batch (images, targets) in loader:
        images, targets ← move_to_device(images, targets, device)          # 数据搬运
        (images_aug, targets_a, targets_b, lam) ← maybe_mix_targets(...)   # Mixup/CutMix
        optimizer.zero_grad()
        with autocast(enabled = use_amp):                                  # 混合精度前向
            outputs ← model(images_aug)
            loss ← lam * criterion(outputs, targets_a)
                    + (1 - lam) * criterion(outputs, targets_b)
        scaler.scale(loss).backward()                                      # 缩放反向
        scaler.step(optimizer)
        scaler.update()
        update_running_metrics(running_loss, running_correct, total_samples,
                               loss, outputs, targets)
    return aggregate_stats(running_loss, running_correct, total_samples)
```

六　Results

从预训练和从零训练的损失和准确率曲线对比来看，预训练微调的模型在训练初期就表现出较低的损失和较高的准确率，说明预训练权重提供了良好的初始化，有助于模型快速收敛。而从零训练的模型则需要更长时间才能达到类似的性能水平。

在训练集和验证集上，TinyViT 的微调都是表现最好的（只用了 21 轮 FineTune 就达到了 98.36% 的验证集精度）。其次是 ResNet-18 的 30 轮微调，说明传统 ResNet 架构在小数据集上依然具有较强的表现力。相比之下，从零训练的模型无论是 ResNet-18 还是 TinyViT，性能都明显落后于预训练版本，验证集精度分别为 83.16% 和 88.02%。

值得注意的是，ResNet-18 在 CIFAR-10 上的微调表现好于 TinyViT-5M 的从零训练，说明在数据有限的情况下，传统卷积网络仍然具有一定优势。

从零训练的实验对比中，TinyViT-5M 的表现好于 ResNet-18，验证集精度分别为 88.02% 和 83.16%，可能和 TinyViT-5M 本身是蒸馏自大模型有关。

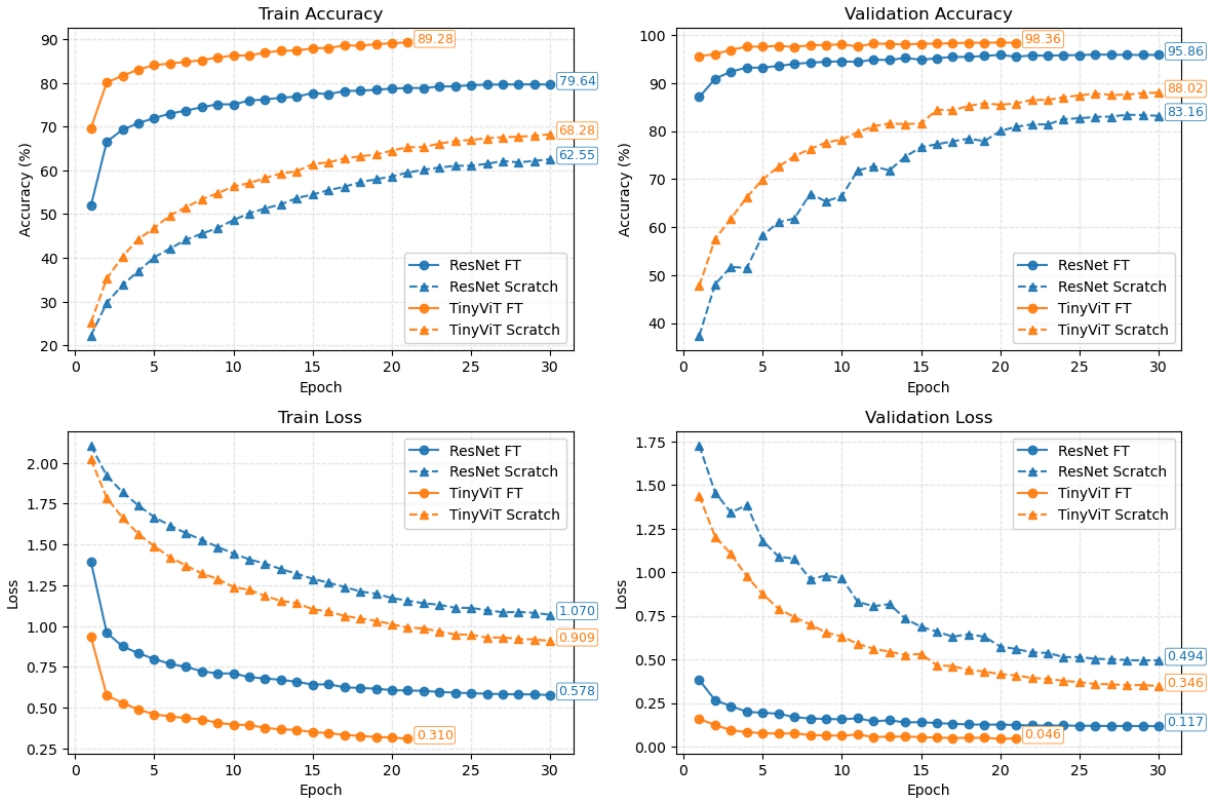

图 6　ResNet-18 在 CIFAR-10 上的训练与验证损失、准确率曲线

表 1 中，TinyViT-5M 微调在 CIFAR-10 上取得了 98.36% 的验证集准确率，显著优于 ResNet-18 的 95.86%。同时，TinyViT-5M 的训练准确率也达到了 89.28%，表明其在训练集上拟合能力较强。验证损失方面，TinyViT-5M 微调的 0.046 远低于 ResNet-18 的 0.117，显示出更好的泛化性能。

表 1　ResNet-18 与 TinyViT-5M 在 CIFAR-10 数据集上的性能比较表

Model	Pretrained	Epochs	Val Acc (%)	Train Acc (%)	Val Loss
ResNet-18 FT	Yes	30	95.86	79.64	0.117
ResNet-18 Scratch	No	30	83.16	62.55	0.494
TinyViT-5M FT	Yes	21	**98.36**	**89.28**	**0.046**
TinyViT-5M Scratch	No	30	88.02	68.28	0.346

另外，本实验也支持对训练集中的图片做分类可视化，如图 7 是一个卡车在 tinyvit-ft 下的正确分类结果展示。

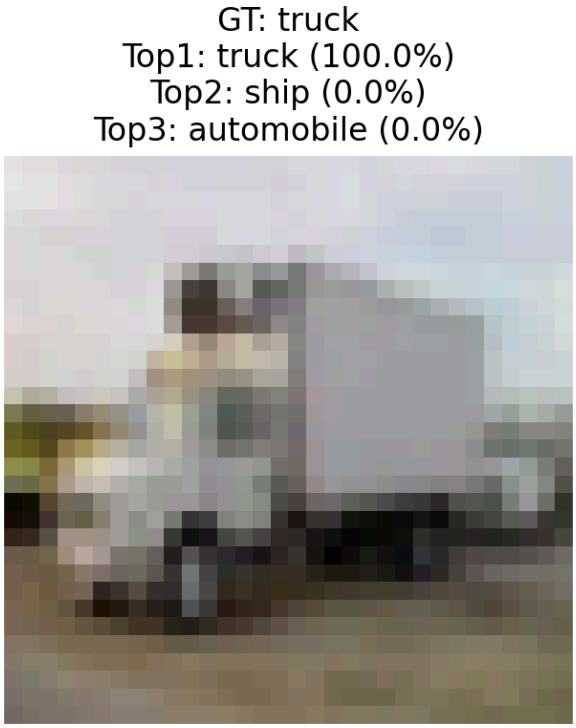

图 7　案例：卡车在 tinyvit-ft 下的正确分类结果

七　Conclusion

通过本次实验，学习了 ResNet 的残差连接设计，理解其缓解深层网络的梯度失效问题的本质。学习了 Transformer 的自注意力机制和多头注意力设计，理解了其在捕捉全局依赖关系方面的优势。通过 TinyViT 模型，了解了轻量级视觉 Transformer 的架构特点及其在资源受限环境下的应用潜力。

通过实验设计和 coding 运行，我们比较了 ResNet-18 和 TinyViT-5M 两种经典模型在 CIFAR-10 数据集上的表现，重点考察了预训练微调与从零训练两种方式的差异。

实验结果显示，预训练微调显著优于从零训练，TinyViT-5M 微调在验证集上达到了 98.36% 的准确率，远超 ResNet-18 的 95.86%。这表明预训练权重为模型提供了良好的初始化，有助于快速收敛和提升泛化能力。同时，TinyViT-5M 在从零训练时也优于 ResNet-18，显示出 Transformer 架构在小数据集上的潜力。

参考文献

[1] K. He, X. Zhang, S. Ren, 和 J. Sun, 《Deep Residual Learning for Image Recognition》, 收入 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi: 10.1109/CVPR.2016.90.

[2] A. Vaswani 等, 《Attention Is All You Need》, 收入 *Advances in Neural Information Processing Systems*, 2017. doi: 10.48550/arXiv.1706.03762.

[3] A. Zhang, Z. C. Lipton, M. Li, 和 A. J. Smola, *Dive into Deep Learning*. d2l.ai, 2023. [在线]. 载于: https://d2l.ai/

[4] A. Dosovitskiy 等, 《An Image is Worth 16x16 Words — Transformers for Image Recognition at Scale》, 2021 年. doi: 10.48550/arXiv.2010.11929.

[5] K. Wu 等, 《TinyViT — Fast Pretraining Distillation for Small Vision Transformers》, 收入 *European Conference on Computer Vision (ECCV)*, 2022.