期末作业：自监督训练、Transformer与三维重建

何益涵　20307110032
吕文韬　23210180109

摘要： 本项目的仓库地址为该超链接[1]. 详见其中的文件夹 `final`. 其中包含任务 1：自监督框架任务 `task1_self_supervised_learning`、任务 2：Transformer 任务 `task2_Transformer_vs_CNN` 以及借助 NeRF 和 COLMAP 完成的三维重建任务 `task3_object_reconstruction_view_synthesis`.

本项目提供了训练好的模型, 详见百度网盘[2], 若链接失效, 也可以通过 Github Issue 联系作者.

[1] https://github.com/HeDesertFox/Neural-Networks-and-Deep-Learning-Homework-Group-Tasks.git

[2] 链接：`https://pan.baidu.com/s/1P5qObkLj0eNHasfmEn_PVw?pwd=wwt1` 提取码：wwt1

第一章　对比监督学习和自监督学习在图像分类任务上的性能表现

第 1 节　任务描述

1. 实现任一自监督学习算法并使用该算法在自选的数据集上训练 ResNet-18, 随后在 CIFAR-100 数据集中使用 Linear Classification Protocol 对其性能进行评测；
2. 将上述结果与在 ImageNet 数据集上采用监督学习训练得到的表征在相同的协议下进行对比, 并比较二者相对于在 CIFAR-100 数据集上从零开始以监督学习方式进行训练所带来的提升；
3. 尝试不同的超参数组合, 探索自监督预训练数据集规模对性能的影响.

第 2 节　项目架构

1. `finetuning_notebook.ipynb` 文件：其中包含下游微调任务；
2. `pretraining_notebook.ipynb` 文件：包含预训练任务；
3. `simclr.py` 文件：该文件实现了模型训练的主要流程，以及损失函数;
4. `simclr_model.py` 文件：该文件实现了 SimCLR 的模型

第 3 节　实验设置

3.1.　模型选择

本项目的自监督训练框架是 SimCLR, 详见以下论文.

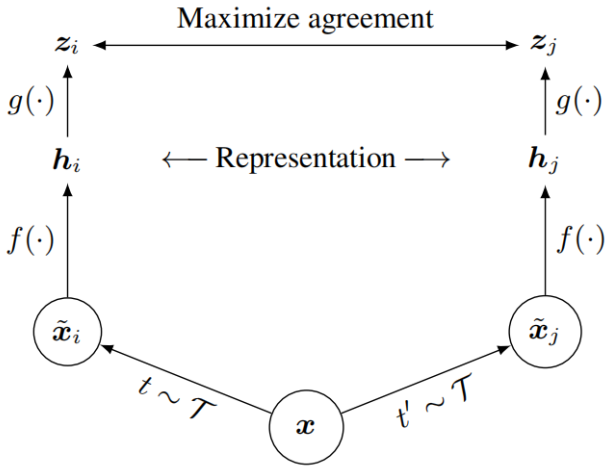

图 1.1: SimCLR 架构

SimCLR 是一种用于自监督学习的框架, 旨在学习高质量的视觉表示. 其核心思想是通过对比学习来最大化正样本对之间的相似度, 并最小化负样本对之间的相似度. SimCLR 架构主要包括以下几个关键步骤：

1. 数据增强：SimCLR 对每张输入图像应用一系列随机的数据增强操作（如裁剪、旋转、颜色抖动等）, 生成两种不同的视图. 这些视图构成正样本对, 而来自不同图像的视图构成负样本对.

2. 编码器：使用深度神经网络（通常是 ResNet）作为编码器, 将每个增强后的图像视图映射到一个低维特征空间中. 编码器的输出是一个固定长度的特征向量.

3. 投影头（Projection Head）：为了计算对比损失, SimCLR 在编码器之后引入了一个小型的 MLP 投影头, 将编码器的输出特征向量进一步映射到另一个特征空间中. 这个步骤有助于学习更好的表示.

4. 对比损失（Contrastive Loss）：SimCLR 使用一种称为 NT-Xent（Normalized Temperature-scaled Cross Entropy）对比损失函数. 对于每对正样本对, 损失函数鼓励它们在特征空间中彼此接近, 同时使负样本对之间的距离尽可能远.

整个 SimCLR 框架通过在一个大规模未标注数据集上进行训练, 逐步优化编码器和投影头的参数. 经过训练后, 编码器能够提取有用的图像特征, 这些特征可以用于下游任务（如分类、检测等）.

SimCLR 的一个显著优点是它仅依赖于数据增强和对比学习, 无需任何人工标注的数据, 从而有效利用大量未标注的数据进行训练. 通过学习更加通用的特征表示, SimCLR 在许多视觉任务中表现出色, 并成为自监督学习领域的重要方法之一.

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^{\top} \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

图 1.2: SimCLR 伪代码

3.2. 数据集

在本次期末作业的实验中, 我们选择了 CIFAR-10 数据集和 STL-10 数据集来预训练 ResNet18, 并在预训练好的 ResNet18 后加入二层 MLP 分类头, 并在 CIFAR-100 数据集上进行了微调.

3.3. 优化器和超参数

本次实验使用 AdamW 做为优化器, 学习率为 0.0003, 设置批次大小为 512. 在 CIFAR10 数据集上, 我们预训练了 90 个 epochs, 在 STL10 数据集上, 我们预训练了 70 个 epochs, 得到了两个版本的 ResNet18.

在训练中, 我们的温度参数设置为 0.07, 特征投射头的维度为 128.

3.4. 数据增强

在本次实验中, 我们选择了和论文中相同的数据增强方法来生成正样本对, 我们对图像按照以下顺序进行数据增强：对图像进行随机裁剪：对图像进行随机裁剪、对图像进行随机色彩失真, 对图像进行高斯模糊.

第 4 节　实验结果

4.1. 数据分析

我们比较了几组 ResNet18 在下游 CIFAR100 分类任务上的性能. 以下是准确率曲线, 四条曲线由上到下分别是在 ImageNet 上预训练的 ResNet18, 在 CIFAR10 上使用 SimCLR 预训练的 ResNet18, 在 STL10 上使用 SimCLR 预训练的 ResNet18 和随机初始化的 ResNet18. 可以看到, 预训练的模型在分类任务上微调时很快收敛, 详见下图：

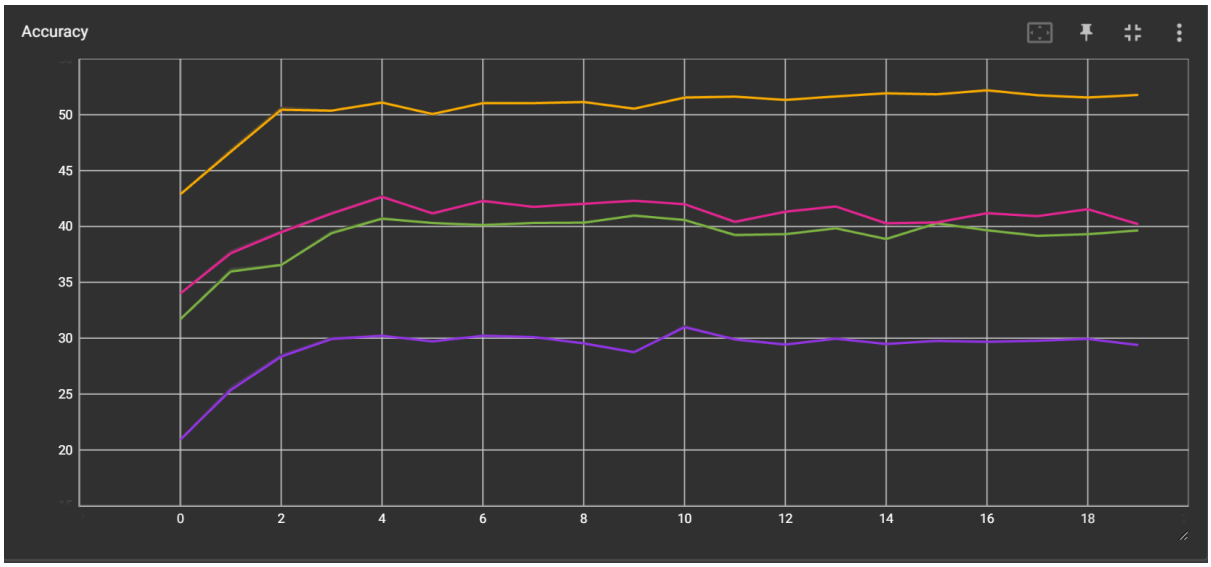

图 1.3: 不同预训练策略下的 ResNet18 在分类任务上的性能比较

下图是四个模型的微调损失函数曲线, 由下到上分别是在 ImageNet 上预训练的 ResNet18, 在 CIFAR10 上使用 SimCLR 预训练的 ResNet18, 在 STL10 上使用 SimCLR 预训练的 ResNet18 和随机初始化的 ResNet18.：

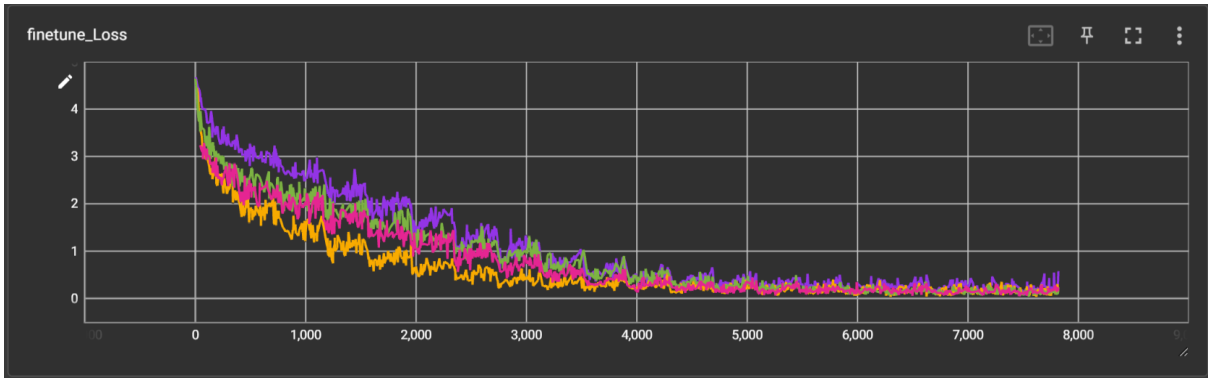

图 1.4: 微调任务的损失函数曲线

4.2. 结论

通过以上实验, 我们发现使用自监督训练方法预训练得到的 ResNet18 相比于随机初始化的 ResNet18 在具体任务微调上具有更为显著的优势.

但同时, 并非是预训练数据集越大越好, STL10 的数据集体量远大于 CIFAR10, 但两个预训练得到的模型在 CIFAR100 分类任务上表现接近, 甚至 CIFAR10 预训练得到的模型在某些 epochs 的评测中效果要好于 STL10, 推测有以下两个可能的原因：

1. SimCLR 的特征投射头设置较小, CIFAR10 的数据量已经达到模型承载极限.
2. SimCLR 损失函数存在优化空间.

第二章　在 CIFAR-100 数据集上比较基于 Transformer 和 CNN 的图像分类模型

第 1 节　任务描述

1. 分别基于 CNN 和 Transformer 架构实现具有相近参数量的图像分类网络；
2. 在 CIFAR-100 数据集上采用相同的训练策略对二者进行训练, 其中数据增强策略中应包含 CutMix；
3. 尝试不同的超参数组合, 尽可能提升各架构在 CIFAR-100 上的性能以进行合理的比较.

第 2 节　项目架构

此任务的所有文件在 `task2_Transformer_vs_CNN` 文件夹中, 其中包含以下文件：

1. `data_preparation.py` 文件：包含数据下载函数与预处理函数, 其中包含 Cutmix 的实现.
2. `model.py` 文件：包含 Resnet 和 ViT 的模型构造函数.
3. `training.py` 文件：包含训练和超参数调优函数.
4. `test_notebook.ipynb` 文件：包含 Transformer 架构和 CNN 架构的对比任务的全部流程, 如数据加载、调参和训练可视化.
5. `para_count.py` 文件：精细统计模型每一层的参数量.

第 3 节　实验设置

3.1.　数据增广

此项目的数据增广在常规图像变化的基础上增加了 Cutmix 增强方法. 详见以下论文.

CutMix 是一种数据增强技术, 旨在提升网络的泛化性能. CutMix 的主要思想是在训练图像之间剪切并粘贴图像块, 同时按比例混合相应的标签. 此方法有助于模型从更多样化的训练样本中学习, 从而提高其鲁棒性和性能.

CutMix 的原理包括两个主要步骤：从一张图像中剪切出一个块并将其粘贴到另一张图像上, 然后按比例调整两张图像的标签. 具体来说, 对于两张图像 x_A 和 x_B 及其对应的标签 y_A 和 y_B, 新的图像 $\tilde{x}$ 和标签 $\tilde{y}$ 的生成方式如下：

$$\tilde{x} = M \odot x_A + (1 - M) \odot x_B \tag{2.1}$$

$$\tilde{y} = \lambda y_A + (1 - \lambda) y_B \tag{2.2}$$

其中, M 是一个二进制掩码, 表示图像块的应用位置, $\odot$ 表示元素级乘法, $\lambda \in [0, 1]$ 是表示原始图像保留比例的随机变量, 通常从 Beta 分布中采样.

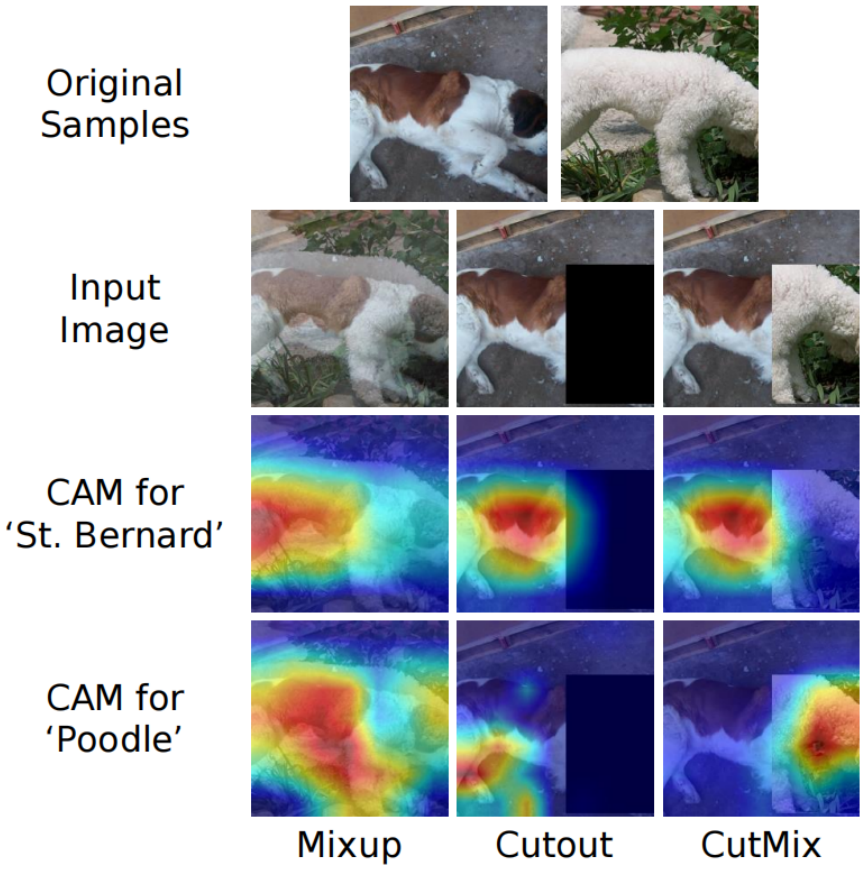

图 2.1: Cutmix 与其他混合增强方法的对比

CutMix 已被证明能通过以下几方面增强深度学习模型的性能：

- 提高泛化性能：通过使模型暴露于更广泛的图像变体, CutMix 有助于防止过拟合, 从而提高在未见数据上的泛化性能.

- 正则化：该技术类似于 dropout, 通过引入随机性防止模型过度依赖特定特征, 起到正则化的作用.
- 标签平滑：标签混合有助于模型学习更柔和的标签, 这对于分类任务中类间界限不明确的情况尤为有益.

实验证明, 与传统的数据增强技术相比, 使用 CutMix 训练的模型在准确率和对抗攻击的鲁棒性方面表现更佳.

CutMix 的实现包括以下步骤：

1. 随机选择两张图像：从训练集中选择两张图像 x_A 和 x_B 及其对应的标签 y_A 和 y_B.
2. 生成二进制掩码：通过在第一张图像中选择一个随机矩形区域, 并将该区域对应的第二张图像的部分设为零, 创建一个二进制掩码 M.
3. 应用 CutMix：使用二进制掩码组合两张图像, 创建新的训练样本. 根据掩码区域的面积按比例调整标签.
4. 更新训练数据：使用新的图像和标签作为训练过程的一部分.

在本项目的实现中, 函数 `rand_bbox` 用于生成随机边界框, 而 `cutmix` 函数则将 CutMix 增强应用于一批图像. 在训练函数每次载入小批量数据时, 我们调用 `cutmix` 函数.

3.2. 模型选择

本项目采用的 CNN 架构是 resnet 架构, 具体采用了 resnet18 架构, 也可以改为更大的 resnet 架构.

本项目采用的 Transformer 架构是经典的 ViT(Vision Transformer) 架构. 详见以下论文：

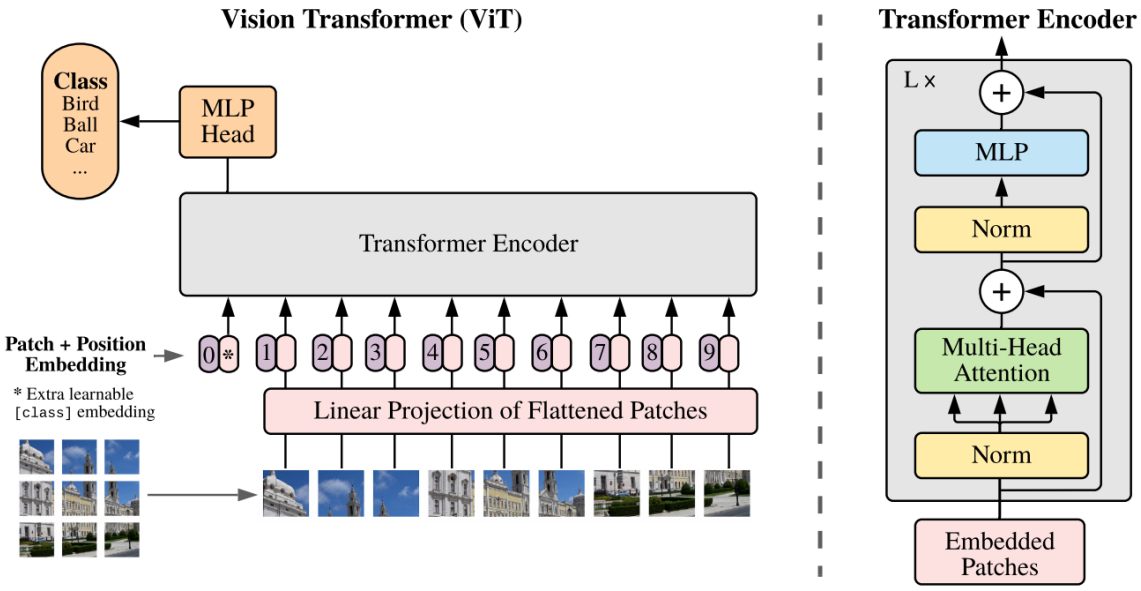

图 2.2: ViT 架构

ViT 是一种将 Transformer 应用于图像分类任务的新型架构. Transformer 最初在自然语言处理（NLP）任务中取得了巨大成功, ViT 则将其应用于计算机视觉领域, 通过将图像划分为一系列图像块（patch）, 并将这些图像块视为序列数据进行处理. ViT 在许多图像分类任务上表现出色, 特别是在大型数据集上.

ViT 的核心思想是将图像分割成固定大小的图像块, 然后将这些图像块展平并嵌入到更高维度的向量空间中. 每个图像块通过线性变换被映射为一个特征向量, 这些特征向量连同位置编码一起作为 Transformer 编码器的输入.

具体步骤如下：

1. 图像分割：将输入图像 $x \in \mathbb{R}^{H \times W \times C}$ 划分为 N 个不重叠的图像块, 每个图像块的大小为 $P \times P$, 因此 $N = \frac{HW}{P^2}$.

2. 图像块展平和嵌入：将每个图像块展平并通过线性变换映射到 d 维特征向量, 得到 $z_0 \in \mathbb{R}^{N \times d}$.

3. 位置编码：为每个图像块添加位置编码, 以保留空间位置信息, 位置编码与特征向量相加形成最终输入 $z_0 + E_{pos}$.

4. Transformer 编码器：将上述输入序列传递给标准的 Transformer 编码器, 编码器由多层自注意力机制和前馈神经网络组成.

5. 分类：使用分类标记（class token）作为输入序列的一部分, 通过 Transformer 编码器的输出进行图像分类.

ViT 在大型数据集上的表现优异, 并且在参数数量相近的情况下, 能够比传统卷积神经网络取得更好的性能. 其主要优势包括：

- 灵活性：ViT 能够处理不同分辨率和大小的图像, 只需调整图像块的大小即可.
- 全局信息：自注意力机制能够捕捉图像中的全局信息, 而不是局限于局部感受野.
- 可扩展性：ViT 能够轻松扩展到更大的模型尺寸和更多的参数, 从而进一步提高性能.

然而, ViT 的一个主要挑战是在小型数据集上容易过拟合, 因此在使用 ViT 时, 通常需要预训练模型或更强的数据增强技术（如 Cutmix）.

3.3. 模型参数量估计

ResNet 的分类头输出维数是 100, resnet18 模型的参数量统计为：11227812.

在我们的 ViT 模型中, `image_size=32` 表示输入图像的尺寸为 32x32 像素;`patch_size=4` 表示将图像划分为 4x4 像素的 patch, 因此一个 32x32 的图像将被划分为 64 个 patch；`num_classes=100` 表示模型需要进行 100 类分类；`dim=192` 表示每个 patch 嵌入到一个 192 维的向量空间；`depth=25` 表示模型包含 25 层 Transformer 编码器块；`heads=6` 表示每个自注意力层包含 6 个独立的注意力头；`mlp_dim=384` 表示前馈神经网络的隐藏层维度为 384. ViT 模型的参数统计为：11139844.

更细致的统计见附录 A.

3.4. 优化器设置

两种架构的优化器都选择 Adam 优化器. 除学习率和权重衰退以外, 其余参数都为默认参数.

第 4 节　调参结果

本实验调节两个参数以尽可能提高模型性能:学习率 `lr` 和权重衰退 `weight_decay`.

经过多轮调参, 最终确定 resnet18 的 `lr` 为 1e-3, `weight_decay` 为 0. ViT 的 `lr` 为 4e-4, `weight_decay` 为 1e-5.

第 5 节　实验结果

5.1. 数据分析

以下图片中, 橙线都是 resnet18 网络的数据曲线, 蓝线都是 ViT 网络的数据曲线.

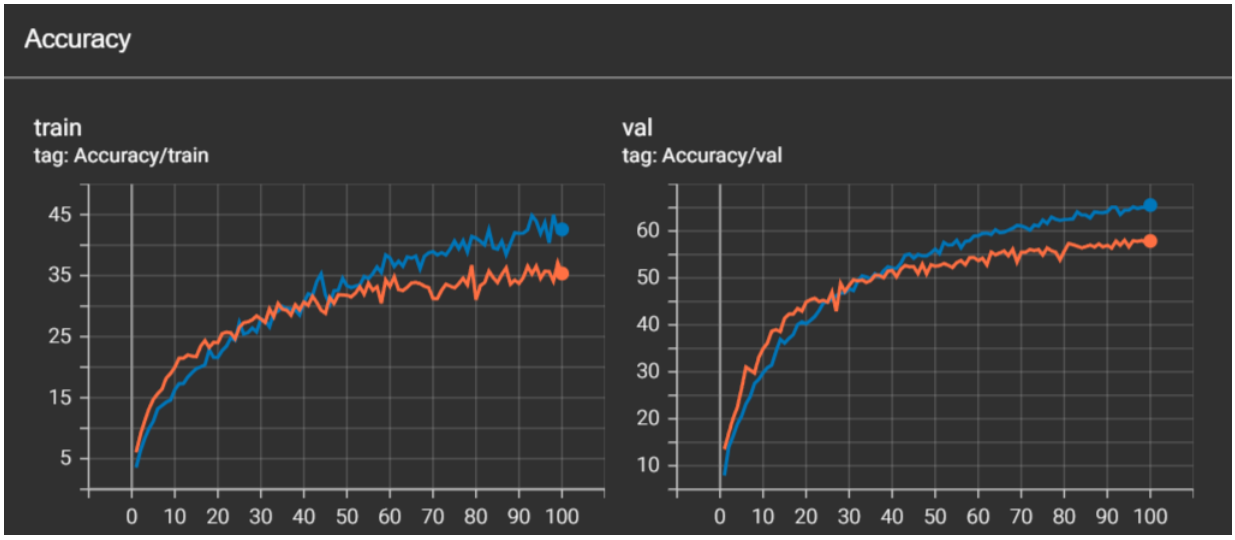

图 2.3: 准确率对比

可以看出两个模型的准确率在训练的过程中不断上升, 并且都没有饱和, 理论上如果训练更长时间, 两者都能获得更好的效果. 训练初始阶段 resnet18 性能更好, 约 30 个 epoch 左右被 ViT 超越. resnet18 的最终验证集准确率为 57.88%, ViT 的最终验证集准确率为 65.53%.

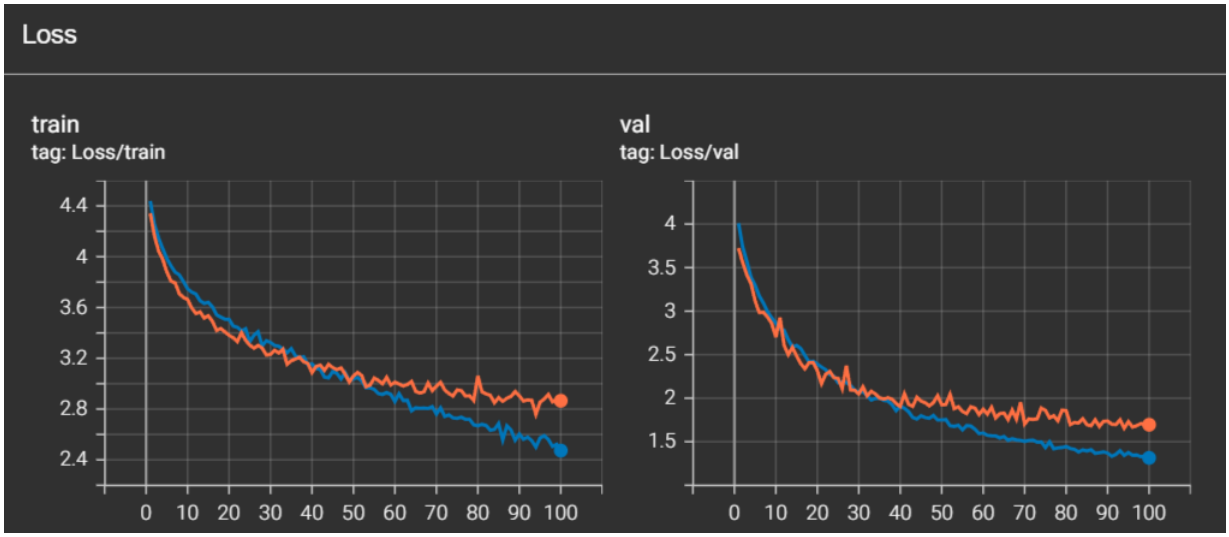

图 2.4: 损失函数对比

两个模型的损失函数值也逐渐下降, 初始时 resnet18 下降更快, 但随后被 ViT 超越.

5.2. 结论

以上实验结果一定程度上验证了 ViT 原论文中的结论, 即 ViT 虽然在较小的数据集上性能不如 CNN 网络, 但在更大的数据集上, ViT 的性能可以超越 CNN 架构. 在我们的实验中, 可以认为训练轮数较少等价于数据集较小, 因此随着训练的进行, 模型学习的数据量增多, ViT 的性能逐渐超越 resnet18.

以下是 ViT 原论文中两种模型的性能对比, 其中 BiT 是一种基于 CNN 的模型架构, 灰色区域是其架构的性能范围, 随着数据量的增加, ViT 的性能增加比 BiT 更多.

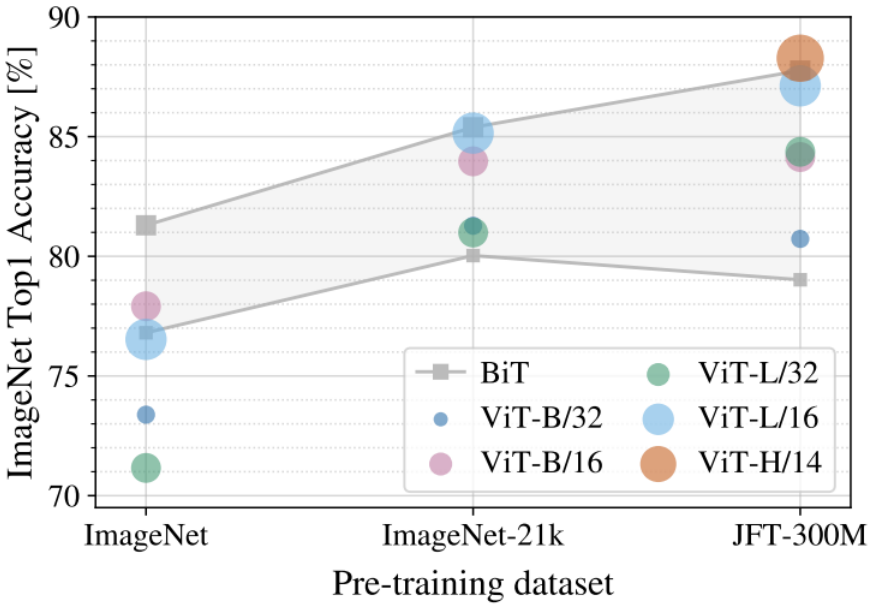

图 2.5: 数据集大小与模型性能的关系

ViT 由于其较复杂的结构, 在小数据集和短期训练中容易过拟合, 导致其性能不如 resnet18. 但随着训练数据量的增加, ViT 能够更好地利用其全局注意力机制, 捕捉数据中的长距离依赖关系, 从而逐渐提升其性能. 在相同训练策略下, ViT 需要更多的训练时间和数据量来展现其优势. 因此, 在实际应用中, 为了充分发挥 ViT 的潜力, 通常需要结合大规模数据集或进行预训练. 对于数据量有限的任务, resnet18 等传统 CNN 架构依然是有效的选择. 而对于数据量充足的任务, ViT 则展现出更强的性能和泛化能力.

第三章　基于 NeRF 的物体重建和新视图合成

第 1 节　任务描述

1. 选取身边的物体拍摄多角度图片/视频, 并使用 COLMAP 估计相机参数, 随后使用现成的框架进行训练；

2. 基于训练好的 NeRF 渲染环绕物体的视频, 并在预留的测试图片上评价定量结果.

第 2 节　项目架构

本次实验使用 nerfstudio 完成了工程部分, 选择的模型是 nerfacto, 是 nerfstudio 的默认选项, 关于模型和 nerfstudio 的架构细节, 详见此论文.

本次实验使用了一段 67s 的视频做为训练数据集, COLMAP 将视频划分为了 335 帧, 其中 90% 的数据被用作模型的训练, 10% 的数据被用作模型的测试.

第 3 节　NeRF 介绍

NeRF（Neural Radiance Fields）是一种使用神经网络来表示三维场景的技术. 其核心思想是通过输入空间位置和视角方向, 利用神经网络预测该位置的颜色和体积密度. NeRF 的主要原理包括以下几个方面：

3.1. 体积渲染

体积渲染是 NeRF 的基础, 通过对三维空间中的点进行采样来生成图像. 具体步骤如下：

1. 对每条摄像机射线（ray）进行均匀采样, 得到多个三维采样点.

2. 对每个采样点, 输入其空间坐标和视角方向到神经网络中, 预测该点的颜色和密度.

3. 通过体积渲染方程, 结合所有采样点的颜色和密度, 计算出该射线对应的像素颜色.

3.2. 网络架构

NeRF 的神经网络通常是一个多层感知器（MLP）, 其输入是空间坐标和视角方向, 输出是颜色和体积密度. 网络架构的细节如下：

- 输入: 三维坐标 $\mathbf{x} = (x, y, z)$ 和视角方向 $\mathbf{d} = (d_x, d_y, d_z)$.
- 输出: 体积密度 σ 和颜色 $\mathbf{c} = (r, g, b)$.
- 网络层数: 通常包含 8 层, 每层包含 256 个神经元, 使用 ReLU 激活函数.
- 跳跃连接: 在中间层添加跳跃连接, 以保留高频信息.

3.3. 优化

NeRF 通过优化网络参数, 使得合成图像与真实图像之间的误差最小.

使用 NeRF 进行三维重建的流程包括数据准备、网络初始化、训练和渲染. 具体步骤如下：

3.4. 数据准备

为了训练 NeRF 模型, 需要多视角的图像数据和相机参数. 数据准备过程如下：

1. 收集多视角图像：从不同视角拍摄目标场景的图像, 确保覆盖足够的视角范围.
2. 获取相机参数：记录每张图像对应的相机位置和方向, 通常使用相机标定技术获取这些参数.

3.5. 网络初始化

初始化 NeRF 模型, 包括网络参数和优化器. 具体步骤如下：

1. 定义网络结构：根据 NeRF 的架构定义 MLP 模型.
2. 初始化参数：随机初始化网络参数.
3. 配置优化器：通常使用 Adam 优化器, 并设置学习率等超参数.

3.6. 训练

训练 NeRF 模型, 使其能够准确预测三维场景中的颜色和密度. 训练过程如下：

1. 射线采样：对于每张图像中的每个像素, 从相机中心沿视线方向采样多条射线.

2. 空间采样：对于每条射线, 在其沿途均匀采样多个三维点.
3. 颜色和密度预测：将每个采样点的坐标和视角方向输入神经网络, 预测该点的颜色和密度.
4. 体积渲染：根据所有采样点的颜色和密度, 通过体积渲染方程计算每条射线的像素颜色.
5. 损失计算：计算合成图像与真实图像之间的 L2 损失.
6. 参数更新：通过反向传播和梯度下降算法, 更新网络参数.

3.7. 渲染

训练完成后, 使用优化后的 NeRF 模型生成新视角的图像. 渲染过程如下：

1. 定义视角：选择新视角的相机位置和方向.
2. 射线采样：从新视角的相机位置沿视线方向采样射线.
3. 空间采样：在每条射线上均匀采样多个三维点.
4. 颜色和密度预测：将每个采样点的坐标和视角方向输入神经网络, 预测该点的颜色和密度.
5. 体积渲染：根据所有采样点的颜色和密度, 通过体积渲染方程计算每条射线的像素颜色, 生成新视角的图像.

NeRF 通过神经网络有效地表示三维场景中的颜色和密度, 结合体积渲染技术, 实现了高质量的三维重建. 这种方法不仅在保留细节方面表现出色, 而且在计算机视觉和图形学领域得到了广泛应用.

第 4 节　实验设置

本次实验借助了架构 Nerf Studio, 此框架提供了一键式训练一些现有 nerf 模型的环境. 实验的主要设置为：

1. 优化器：采用 Adam 优化器, NeRF 的训练有三部分：相机参数优化、场和建议网络, 这几部分的学习率中, 后两个的学习率我们使得其线性下降, 第一个学习率我们令其以 t^{-1} 的速度下降, 最低为 1×10^{-4}.
2. 我们训练了 100 个 epoch, 每个 epoch 都使用了 4096 的 batch size, 每 500 次迭代之后评估一张测试集的图片, 25000 个迭代之后评估所有图片.

第 5 节　实验结果

本次实验的训练损失函数和测试集上的损失函数见下图：

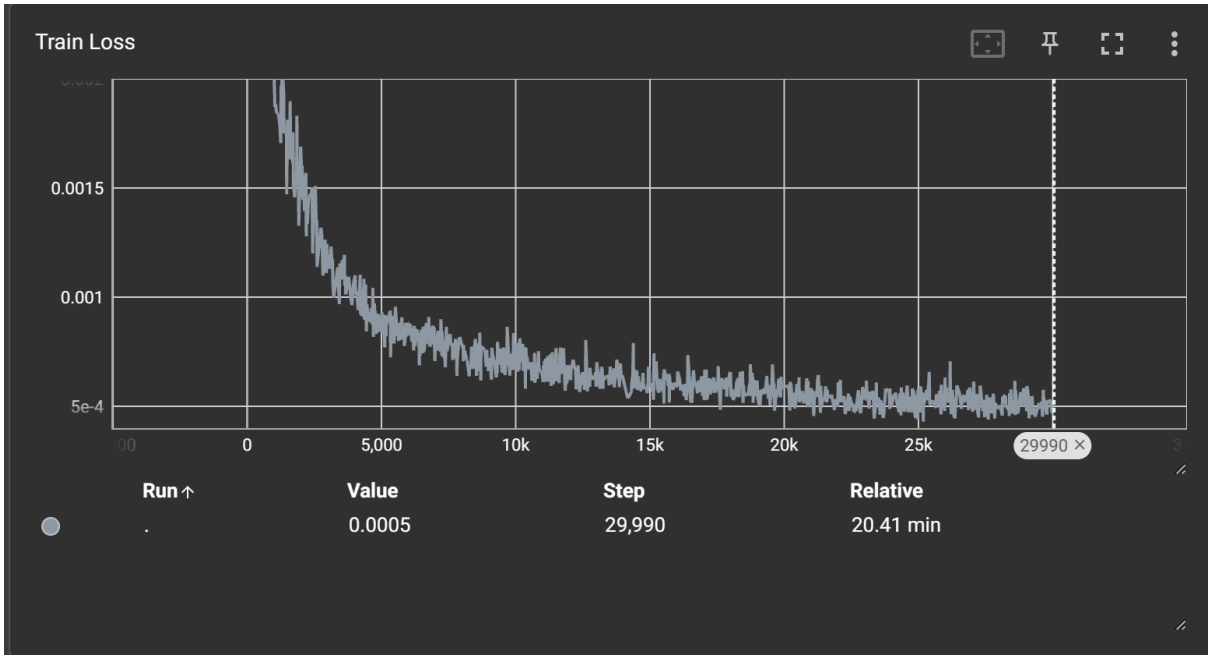

图 3.1: NeRFacto 训练损失函数

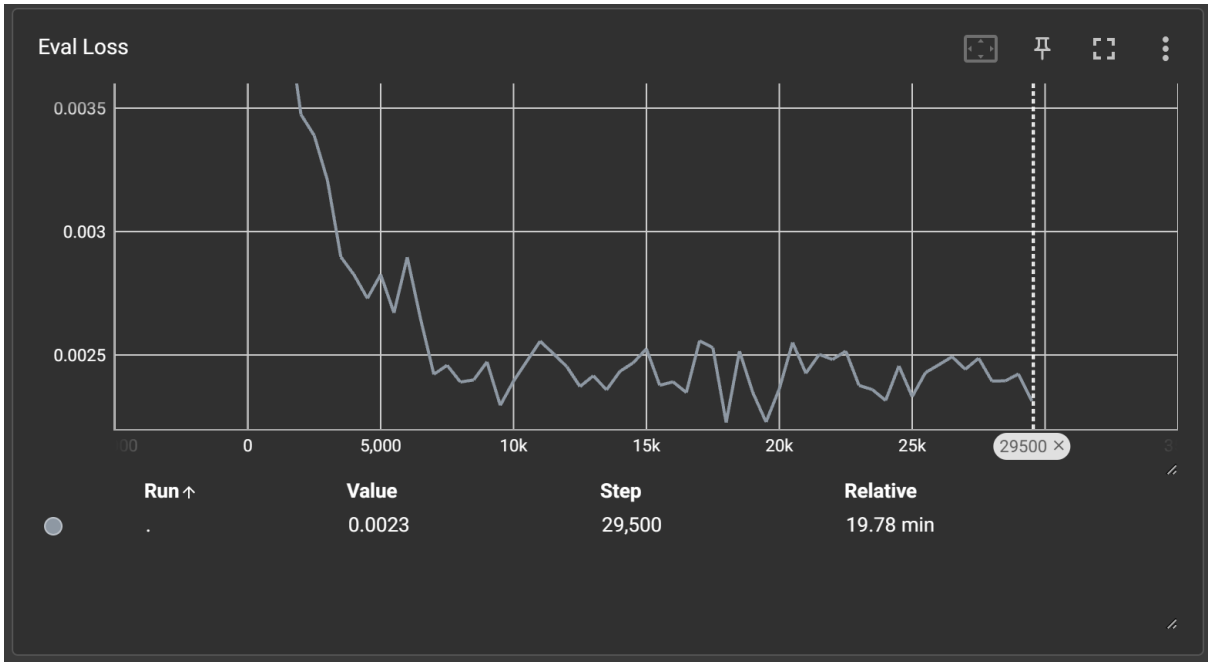

图 3.2: NeRFacto 测试损失函数

同时, 我们还输出了在测试图像上的 PSNR, 见下图：

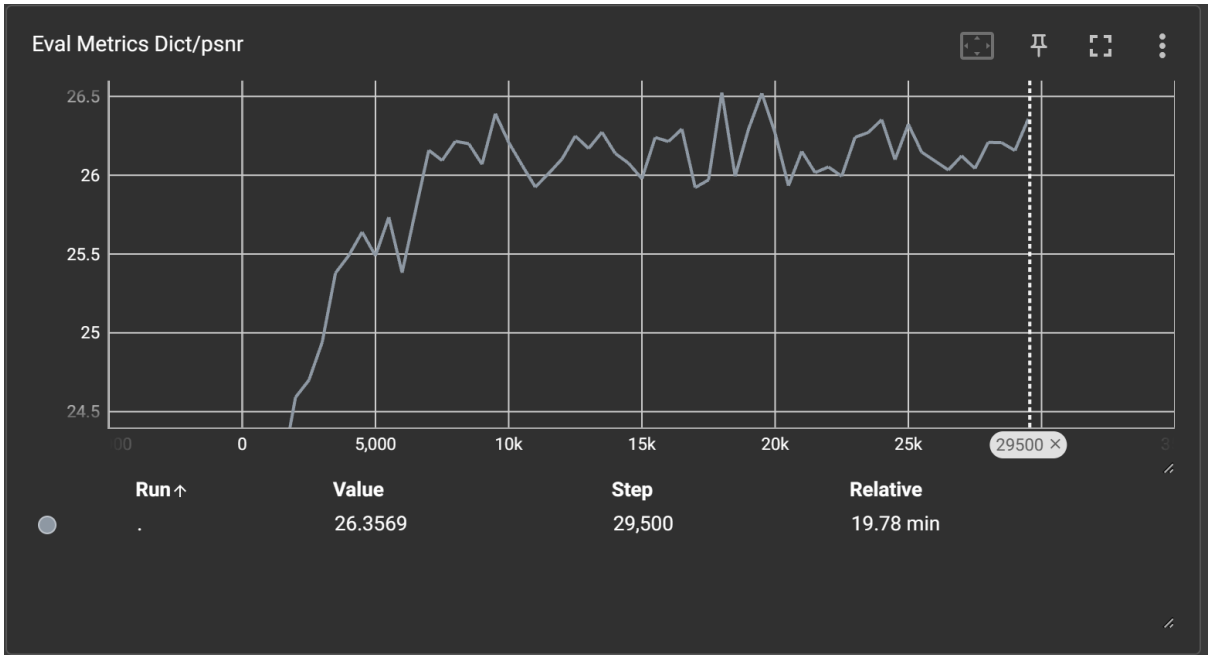

图 3.3: 测试图片上的 PSNR

5.1. 结果的可视化

我们使用 NeRF 训练的结果重建了 3D 点云, 并渲染了环绕物体的视频.

我们选取的物体是一个可回收垃圾桶，其放置的环境在光华楼东主楼 14 楼的电梯间，3D 点云重建的效果如下，可以看到图像中是我们的 3D 模型从测上方观测的视角，所有的训练集图片可以在环绕图片的照片标志上观察到. 3D 点云重建的效果如下：

图 3.4: 3D 重建的可视化

我们渲染得到的视频在电子文件中可用.

附录 A ResNet18 和 ViT 的模型参数量精细统计

可以运行 task2_Transformer_vs_CNN 文件夹中的 para_count.py 文件来统计模型参数. 以下是统计结果：

```
ResNet-18 Architecture and Parameters:
Layer                                           Shape                Params
=================================================================
conv1.weight                                    [64, 3, 7, 7]        9408
bn1.weight                                      [64]                 64
bn1.bias                                        [64]                 64
layer1.0.conv1.weight                           [64, 64, 3, 3]       36864
layer1.0.bn1.weight                             [64]                 64
layer1.0.bn1.bias                               [64]                 64
layer1.0.conv2.weight                           [64, 64, 3, 3]       36864
layer1.0.bn2.weight                             [64]                 64
layer1.0.bn2.bias                               [64]                 64
layer1.1.conv1.weight                           [64, 64, 3, 3]       36864
layer1.1.bn1.weight                             [64]                 64
layer1.1.bn1.bias                               [64]                 64
layer1.1.conv2.weight                           [64, 64, 3, 3]       36864
layer1.1.bn2.weight                             [64]                 64
layer1.1.bn2.bias                               [64]                 64
layer2.0.conv1.weight                           [128, 64, 3, 3]      73728
layer2.0.bn1.weight                             [128]                128
layer2.0.bn1.bias                               [128]                128
layer2.0.conv2.weight                           [128, 128, 3, 3]     147456
layer2.0.bn2.weight                             [128]                128
layer2.0.bn2.bias                               [128]                128
layer2.0.downsample.0.weight                    [128, 64, 1, 1]      8192
layer2.0.downsample.1.weight                    [128]                128
layer2.0.downsample.1.bias                      [128]                128
layer2.1.conv1.weight                           [128, 128, 3, 3]     147456
layer2.1.bn1.weight                             [128]                128
layer2.1.bn1.bias                               [128]                128
layer2.1.conv2.weight                           [128, 128, 3, 3]     147456
layer2.1.bn2.weight                             [128]                128
```

```
layer2.1.bn2.bias                          [128]              128
layer3.0.conv1.weight                      [256, 128, 3, 3]   294912
layer3.0.bn1.weight                        [256]              256
layer3.0.bn1.bias                          [256]              256
layer3.0.conv2.weight                      [256, 256, 3, 3]   589824
layer3.0.bn2.weight                        [256]              256
layer3.0.bn2.bias                          [256]              256
layer3.0.downsample.0.weight               [256, 128, 1, 1]   32768
layer3.0.downsample.1.weight               [256]              256
layer3.0.downsample.1.bias                 [256]              256
layer3.1.conv1.weight                      [256, 256, 3, 3]   589824
layer3.1.bn1.weight                        [256]              256
layer3.1.bn1.bias                          [256]              256
layer3.1.conv2.weight                      [256, 256, 3, 3]   589824
layer3.1.bn2.weight                        [256]              256
layer3.1.bn2.bias                          [256]              256
layer4.0.conv1.weight                      [512, 256, 3, 3]   1179648
layer4.0.bn1.weight                        [512]              512
layer4.0.bn1.bias                          [512]              512
layer4.0.conv2.weight                      [512, 512, 3, 3]   2359296
layer4.0.bn2.weight                        [512]              512
layer4.0.bn2.bias                          [512]              512
layer4.0.downsample.0.weight               [512, 256, 1, 1]   131072
layer4.0.downsample.1.weight               [512]              512
layer4.0.downsample.1.bias                 [512]              512
layer4.1.conv1.weight                      [512, 512, 3, 3]   2359296
layer4.1.bn1.weight                        [512]              512
layer4.1.bn1.bias                          [512]              512
layer4.1.conv2.weight                      [512, 512, 3, 3]   2359296
layer4.1.bn2.weight                        [512]              512
layer4.1.bn2.bias                          [512]              512
fc.weight                                  [100, 512]         51200
fc.bias                                    [100]              100
================================================================
Total Parameters: 11227812

ViT Architecture and Parameters:
Layer                                      Shape              Params
================================================================
pos_embedding                              [1, 65, 192]       12480
```

73	cls_token	[1, 1, 192]	192
74	to_patch_embedding.1.weight	[48]	48
75	to_patch_embedding.1.bias	[48]	48
76	to_patch_embedding.2.weight	[192, 48]	9216
77	to_patch_embedding.2.bias	[192]	192
78	to_patch_embedding.3.weight	[192]	192
79	to_patch_embedding.3.bias	[192]	192
80	transformer.norm.weight	[192]	192
81	transformer.norm.bias	[192]	192
82	transformer.layers.0.0.norm.weight	[192]	192
83	transformer.layers.0.0.norm.bias	[192]	192
84	transformer.layers.0.0.to_qkv.weight	[1152, 192]	221184
85	transformer.layers.0.0.to_out.0.weight	[192, 384]	73728
86	transformer.layers.0.0.to_out.0.bias	[192]	192
87	transformer.layers.0.1.net.0.weight	[192]	192
88	transformer.layers.0.1.net.0.bias	[192]	192
89	transformer.layers.0.1.net.1.weight	[384, 192]	73728
90	transformer.layers.0.1.net.1.bias	[384]	384
91	transformer.layers.0.1.net.4.weight	[192, 384]	73728
92	transformer.layers.0.1.net.4.bias	[192]	192
93	transformer.layers.1.0.norm.weight	[192]	192
94	transformer.layers.1.0.norm.bias	[192]	192
95	transformer.layers.1.0.to_qkv.weight	[1152, 192]	221184
96	transformer.layers.1.0.to_out.0.weight	[192, 384]	73728
97	transformer.layers.1.0.to_out.0.bias	[192]	192
98	transformer.layers.1.1.net.0.weight	[192]	192
99	transformer.layers.1.1.net.0.bias	[192]	192
100	transformer.layers.1.1.net.1.weight	[384, 192]	73728
101	transformer.layers.1.1.net.1.bias	[384]	384
102	transformer.layers.1.1.net.4.weight	[192, 384]	73728
103	transformer.layers.1.1.net.4.bias	[192]	192
104	transformer.layers.2.0.norm.weight	[192]	192
105	transformer.layers.2.0.norm.bias	[192]	192
106	transformer.layers.2.0.to_qkv.weight	[1152, 192]	221184
107	transformer.layers.2.0.to_out.0.weight	[192, 384]	73728
108	transformer.layers.2.0.to_out.0.bias	[192]	192
109	transformer.layers.2.1.net.0.weight	[192]	192
110	transformer.layers.2.1.net.0.bias	[192]	192
111	transformer.layers.2.1.net.1.weight	[384, 192]	73728
112	transformer.layers.2.1.net.1.bias	[384]	384

```
transformer.layers.2.1.net.4.weight      [192, 384]          73728
transformer.layers.2.1.net.4.bias        [192]               192
transformer.layers.3.0.norm.weight       [192]               192
transformer.layers.3.0.norm.bias         [192]               192
transformer.layers.3.0.to_qkv.weight     [1152, 192]         221184
transformer.layers.3.0.to_out.0.weight   [192, 384]          73728
transformer.layers.3.0.to_out.0.bias     [192]               192
transformer.layers.3.1.net.0.weight      [192]               192
transformer.layers.3.1.net.0.bias        [192]               192
transformer.layers.3.1.net.1.weight      [384, 192]          73728
transformer.layers.3.1.net.1.bias        [384]               384
transformer.layers.3.1.net.4.weight      [192, 384]          73728
transformer.layers.3.1.net.4.bias        [192]               192
transformer.layers.4.0.norm.weight       [192]               192
transformer.layers.4.0.norm.bias         [192]               192
transformer.layers.4.0.to_qkv.weight     [1152, 192]         221184
transformer.layers.4.0.to_out.0.weight   [192, 384]          73728
transformer.layers.4.0.to_out.0.bias     [192]               192
transformer.layers.4.1.net.0.weight      [192]               192
transformer.layers.4.1.net.0.bias        [192]               192
transformer.layers.4.1.net.1.weight      [384, 192]          73728
transformer.layers.4.1.net.1.bias        [384]               384
transformer.layers.4.1.net.4.weight      [192, 384]          73728
transformer.layers.4.1.net.4.bias        [192]               192
transformer.layers.5.0.norm.weight       [192]               192
transformer.layers.5.0.norm.bias         [192]               192
transformer.layers.5.0.to_qkv.weight     [1152, 192]         221184
transformer.layers.5.0.to_out.0.weight   [192, 384]          73728
transformer.layers.5.0.to_out.0.bias     [192]               192
transformer.layers.5.1.net.0.weight      [192]               192
transformer.layers.5.1.net.0.bias        [192]               192
transformer.layers.5.1.net.1.weight      [384, 192]          73728
transformer.layers.5.1.net.1.bias        [384]               384
transformer.layers.5.1.net.4.weight      [192, 384]          73728
transformer.layers.5.1.net.4.bias        [192]               192
transformer.layers.6.0.norm.weight       [192]               192
transformer.layers.6.0.norm.bias         [192]               192
transformer.layers.6.0.to_qkv.weight     [1152, 192]         221184
transformer.layers.6.0.to_out.0.weight   [192, 384]          73728
transformer.layers.6.0.to_out.0.bias     [192]               192
```

```
transformer.layers.6.1.net.0.weight      [192]          192
transformer.layers.6.1.net.0.bias        [192]          192
transformer.layers.6.1.net.1.weight      [384, 192]     73728
transformer.layers.6.1.net.1.bias        [384]          384
transformer.layers.6.1.net.4.weight      [192, 384]     73728
transformer.layers.6.1.net.4.bias        [192]          192
transformer.layers.7.0.norm.weight       [192]          192
transformer.layers.7.0.norm.bias         [192]          192
transformer.layers.7.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.7.0.to_out.0.weight   [192, 384]     73728
transformer.layers.7.0.to_out.0.bias     [192]          192
transformer.layers.7.1.net.0.weight      [192]          192
transformer.layers.7.1.net.0.bias        [192]          192
transformer.layers.7.1.net.1.weight      [384, 192]     73728
transformer.layers.7.1.net.1.bias        [384]          384
transformer.layers.7.1.net.4.weight      [192, 384]     73728
transformer.layers.7.1.net.4.bias        [192]          192
transformer.layers.8.0.norm.weight       [192]          192
transformer.layers.8.0.norm.bias         [192]          192
transformer.layers.8.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.8.0.to_out.0.weight   [192, 384]     73728
transformer.layers.8.0.to_out.0.bias     [192]          192
transformer.layers.8.1.net.0.weight      [192]          192
transformer.layers.8.1.net.0.bias        [192]          192
transformer.layers.8.1.net.1.weight      [384, 192]     73728
transformer.layers.8.1.net.1.bias        [384]          384
transformer.layers.8.1.net.4.weight      [192, 384]     73728
transformer.layers.8.1.net.4.bias        [192]          192
transformer.layers.9.0.norm.weight       [192]          192
transformer.layers.9.0.norm.bias         [192]          192
transformer.layers.9.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.9.0.to_out.0.weight   [192, 384]     73728
transformer.layers.9.0.to_out.0.bias     [192]          192
transformer.layers.9.1.net.0.weight      [192]          192
transformer.layers.9.1.net.0.bias        [192]          192
transformer.layers.9.1.net.1.weight      [384, 192]     73728
transformer.layers.9.1.net.1.bias        [384]          384
transformer.layers.9.1.net.4.weight      [192, 384]     73728
transformer.layers.9.1.net.4.bias        [192]          192
transformer.layers.10.0.norm.weight      [192]          192
```

```
transformer.layers.10.0.norm.bias         [192]          192
transformer.layers.10.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.10.0.to_out.0.weight   [192, 384]     73728
transformer.layers.10.0.to_out.0.bias     [192]          192
transformer.layers.10.1.net.0.weight      [192]          192
transformer.layers.10.1.net.0.bias        [192]          192
transformer.layers.10.1.net.1.weight      [384, 192]     73728
transformer.layers.10.1.net.1.bias        [384]          384
transformer.layers.10.1.net.4.weight      [192, 384]     73728
transformer.layers.10.1.net.4.bias        [192]          192
transformer.layers.11.0.norm.weight       [192]          192
transformer.layers.11.0.norm.bias         [192]          192
transformer.layers.11.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.11.0.to_out.0.weight   [192, 384]     73728
transformer.layers.11.0.to_out.0.bias     [192]          192
transformer.layers.11.1.net.0.weight      [192]          192
transformer.layers.11.1.net.0.bias        [192]          192
transformer.layers.11.1.net.1.weight      [384, 192]     73728
transformer.layers.11.1.net.1.bias        [384]          384
transformer.layers.11.1.net.4.weight      [192, 384]     73728
transformer.layers.11.1.net.4.bias        [192]          192
transformer.layers.12.0.norm.weight       [192]          192
transformer.layers.12.0.norm.bias         [192]          192
transformer.layers.12.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.12.0.to_out.0.weight   [192, 384]     73728
transformer.layers.12.0.to_out.0.bias     [192]          192
transformer.layers.12.1.net.0.weight      [192]          192
transformer.layers.12.1.net.0.bias        [192]          192
transformer.layers.12.1.net.1.weight      [384, 192]     73728
transformer.layers.12.1.net.1.bias        [384]          384
transformer.layers.12.1.net.4.weight      [192, 384]     73728
transformer.layers.12.1.net.4.bias        [192]          192
transformer.layers.13.0.norm.weight       [192]          192
transformer.layers.13.0.norm.bias         [192]          192
transformer.layers.13.0.to_qkv.weight     [1152, 192]    221184
transformer.layers.13.0.to_out.0.weight   [192, 384]     73728
transformer.layers.13.0.to_out.0.bias     [192]          192
transformer.layers.13.1.net.0.weight      [192]          192
transformer.layers.13.1.net.0.bias        [192]          192
transformer.layers.13.1.net.1.weight      [384, 192]     73728
```

```
transformer.layers.13.1.net.1.bias       [384]          384
transformer.layers.13.1.net.4.weight     [192, 384]     73728
transformer.layers.13.1.net.4.bias       [192]          192
transformer.layers.14.0.norm.weight      [192]          192
transformer.layers.14.0.norm.bias        [192]          192
transformer.layers.14.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.14.0.to_out.0.weight  [192, 384]     73728
transformer.layers.14.0.to_out.0.bias    [192]          192
transformer.layers.14.1.net.0.weight     [192]          192
transformer.layers.14.1.net.0.bias       [192]          192
transformer.layers.14.1.net.1.weight     [384, 192]     73728
transformer.layers.14.1.net.1.bias       [384]          384
transformer.layers.14.1.net.4.weight     [192, 384]     73728
transformer.layers.14.1.net.4.bias       [192]          192
transformer.layers.15.0.norm.weight      [192]          192
transformer.layers.15.0.norm.bias        [192]          192
transformer.layers.15.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.15.0.to_out.0.weight  [192, 384]     73728
transformer.layers.15.0.to_out.0.bias    [192]          192
transformer.layers.15.1.net.0.weight     [192]          192
transformer.layers.15.1.net.0.bias       [192]          192
transformer.layers.15.1.net.1.weight     [384, 192]     73728
transformer.layers.15.1.net.1.bias       [384]          384
transformer.layers.15.1.net.4.weight     [192, 384]     73728
transformer.layers.15.1.net.4.bias       [192]          192
transformer.layers.16.0.norm.weight      [192]          192
transformer.layers.16.0.norm.bias        [192]          192
transformer.layers.16.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.16.0.to_out.0.weight  [192, 384]     73728
transformer.layers.16.0.to_out.0.bias    [192]          192
transformer.layers.16.1.net.0.weight     [192]          192
transformer.layers.16.1.net.0.bias       [192]          192
transformer.layers.16.1.net.1.weight     [384, 192]     73728
transformer.layers.16.1.net.1.bias       [384]          384
transformer.layers.16.1.net.4.weight     [192, 384]     73728
transformer.layers.16.1.net.4.bias       [192]          192
transformer.layers.17.0.norm.weight      [192]          192
transformer.layers.17.0.norm.bias        [192]          192
transformer.layers.17.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.17.0.to_out.0.weight  [192, 384]     73728
```

```
transformer.layers.17.0.to_out.0.bias    [192]             192
transformer.layers.17.1.net.0.weight     [192]             192
transformer.layers.17.1.net.0.bias       [192]             192
transformer.layers.17.1.net.1.weight     [384, 192]        73728
transformer.layers.17.1.net.1.bias       [384]             384
transformer.layers.17.1.net.4.weight     [192, 384]        73728
transformer.layers.17.1.net.4.bias       [192]             192
transformer.layers.18.0.norm.weight      [192]             192
transformer.layers.18.0.norm.bias        [192]             192
transformer.layers.18.0.to_qkv.weight    [1152, 192]       221184
transformer.layers.18.0.to_out.0.weight  [192, 384]        73728
transformer.layers.18.0.to_out.0.bias    [192]             192
transformer.layers.18.1.net.0.weight     [192]             192
transformer.layers.18.1.net.0.bias       [192]             192
transformer.layers.18.1.net.1.weight     [384, 192]        73728
transformer.layers.18.1.net.1.bias       [384]             384
transformer.layers.18.1.net.4.weight     [192, 384]        73728
transformer.layers.18.1.net.4.bias       [192]             192
transformer.layers.19.0.norm.weight      [192]             192
transformer.layers.19.0.norm.bias        [192]             192
transformer.layers.19.0.to_qkv.weight    [1152, 192]       221184
transformer.layers.19.0.to_out.0.weight  [192, 384]        73728
transformer.layers.19.0.to_out.0.bias    [192]             192
transformer.layers.19.1.net.0.weight     [192]             192
transformer.layers.19.1.net.0.bias       [192]             192
transformer.layers.19.1.net.1.weight     [384, 192]        73728
transformer.layers.19.1.net.1.bias       [384]             384
transformer.layers.19.1.net.4.weight     [192, 384]        73728
transformer.layers.19.1.net.4.bias       [192]             192
transformer.layers.20.0.norm.weight      [192]             192
transformer.layers.20.0.norm.bias        [192]             192
transformer.layers.20.0.to_qkv.weight    [1152, 192]       221184
transformer.layers.20.0.to_out.0.weight  [192, 384]        73728
transformer.layers.20.0.to_out.0.bias    [192]             192
transformer.layers.20.1.net.0.weight     [192]             192
transformer.layers.20.1.net.0.bias       [192]             192
transformer.layers.20.1.net.1.weight     [384, 192]        73728
transformer.layers.20.1.net.1.bias       [384]             384
transformer.layers.20.1.net.4.weight     [192, 384]        73728
transformer.layers.20.1.net.4.bias       [192]             192
```

```
transformer.layers.21.0.norm.weight      [192]          192
transformer.layers.21.0.norm.bias        [192]          192
transformer.layers.21.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.21.0.to_out.0.weight  [192, 384]     73728
transformer.layers.21.0.to_out.0.bias    [192]          192
transformer.layers.21.1.net.0.weight     [192]          192
transformer.layers.21.1.net.0.bias       [192]          192
transformer.layers.21.1.net.1.weight     [384, 192]     73728
transformer.layers.21.1.net.1.bias       [384]          384
transformer.layers.21.1.net.4.weight     [192, 384]     73728
transformer.layers.21.1.net.4.bias       [192]          192
transformer.layers.22.0.norm.weight      [192]          192
transformer.layers.22.0.norm.bias        [192]          192
transformer.layers.22.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.22.0.to_out.0.weight  [192, 384]     73728
transformer.layers.22.0.to_out.0.bias    [192]          192
transformer.layers.22.1.net.0.weight     [192]          192
transformer.layers.22.1.net.0.bias       [192]          192
transformer.layers.22.1.net.1.weight     [384, 192]     73728
transformer.layers.22.1.net.1.bias       [384]          384
transformer.layers.22.1.net.4.weight     [192, 384]     73728
transformer.layers.22.1.net.4.bias       [192]          192
transformer.layers.23.0.norm.weight      [192]          192
transformer.layers.23.0.norm.bias        [192]          192
transformer.layers.23.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.23.0.to_out.0.weight  [192, 384]     73728
transformer.layers.23.0.to_out.0.bias    [192]          192
transformer.layers.23.1.net.0.weight     [192]          192
transformer.layers.23.1.net.0.bias       [192]          192
transformer.layers.23.1.net.1.weight     [384, 192]     73728
transformer.layers.23.1.net.1.bias       [384]          384
transformer.layers.23.1.net.4.weight     [192, 384]     73728
transformer.layers.23.1.net.4.bias       [192]          192
transformer.layers.24.0.norm.weight      [192]          192
transformer.layers.24.0.norm.bias        [192]          192
transformer.layers.24.0.to_qkv.weight    [1152, 192]    221184
transformer.layers.24.0.to_out.0.weight  [192, 384]     73728
transformer.layers.24.0.to_out.0.bias    [192]          192
transformer.layers.24.1.net.0.weight     [192]          192
transformer.layers.24.1.net.0.bias       [192]          192
```

```
transformer.layers.24.1.net.1.weight     [384, 192]          73728
transformer.layers.24.1.net.1.bias       [384]               384
transformer.layers.24.1.net.4.weight     [192, 384]          73728
transformer.layers.24.1.net.4.bias       [192]               192
mlp_head.weight                          [100, 192]          19200
mlp_head.bias                            [100]               100
==========================================================================
Total Parameters: 11139844
```