深度学习作业——基于 ViT 的 CIFAR10 图像分类

Clark

1 概述

本实验使用 PyTorch 框架实现了 Vision Transformer（ViT）模型，并在 CIFAR-10 数据集上完成图像分类任务。通过数据增强、多头注意力机制和余弦退火学习率调度等策略，模型在测试集上达到 84.84% 的准确率，远超 80% 的实验要求。训练过程中引入早停机制，在 34 个 epoch 时精度已超过 80%，最佳训练精度达 93.21%（第 86 轮）。实验完整复现了 ViT 的核心思想，验证了 Transformer 在计算机视觉任务中的有效性。

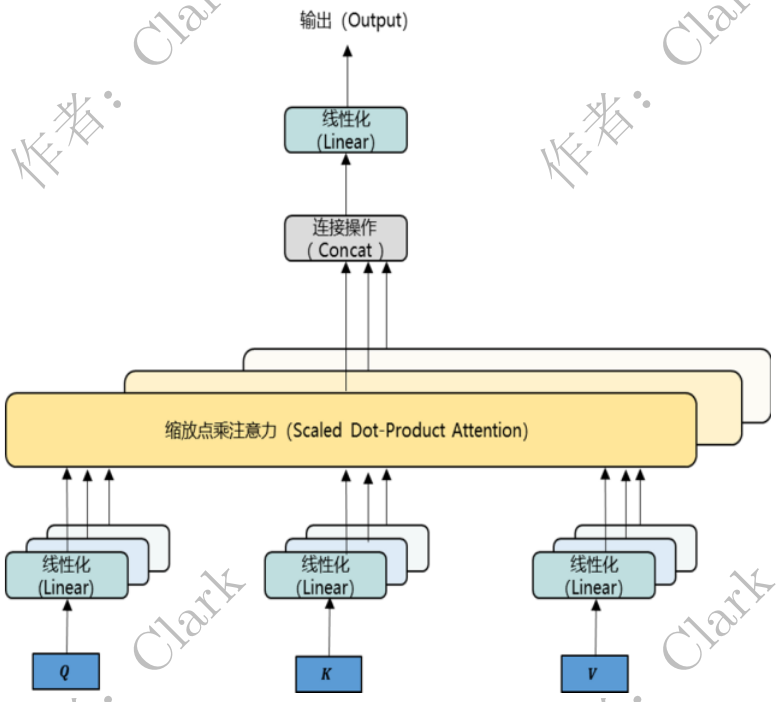
2 实验目的

1. 掌握 ViT 模型的结构原理及其在图像分类任务中的应用；
2. 学习使用 PyTorch 实现深度学习模型的完整流程（数据加载、模型构建、训练优化）；
3. 理解多头注意力机制和位置编码在视觉任务中的作用；
4. 实现 CIFAR-10 数据集的分类准确率超过 80% 的性能要求。

3 实验原理

3.1 Vision Transformer 模型概述

ViT 首次将纯 Transformer 架构应用于计算机视觉领域，其核心思想是将图像分割为序列化的图像块（Patch），通过线性映射后输入 Transformer 编码器。模型摒弃了传统 CNN 的卷积操作，完全依赖自注意力机制捕捉全局特征。



3.2 关键组件详解

3.2.1 图像分块嵌入（Patch Embedding）

将输入图像（32×32）分割为 4×4 的块（共 64 块），每个 Patch 展平为 48 维向量，通过线性层映射到 256 维：

```
self.to_patch_embedding = nn.Sequential(
    Rearrange('b␣c␣(h␣p1)␣(w␣p2)␣->␣b␣(h␣w)␣(p1␣p2␣c)', p1=patch_height, p2=patch_width),
    nn.LayerNorm(patch_dim),
    nn.Linear(patch_dim, dim),
    nn.LayerNorm(dim),
)
```

3.2.2 位置编码与 CLS 令牌

加入可学习的位置编码保持空间信息，并添加 CLS 令牌用于分类：

```
self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))  # 64 patches +
    CLS
self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
```

其作用是使用 [CLS] 令牌或全局平均池化聚合特征，通过线性层输出 10 分类结果。

3.2.3 Transformer 编码器

Transformer 编码器由多头注意力（Multi-Head Attention）和前馈网络（FeedForward）组成，支持残差连接和 LayerNorm。

3.2.3.1 多头自注意力机制

注意力权重计算公式：

$$\text{Att} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V, \tag{1}$$

其中 Q、K、V 为查询、键、值矩阵，d 为维度。代码实现中通过 8 个注意力头并行计算：

```python
class Attention(nn.Module):
    def __init__(self, dim, heads=8, dim_head=64, dropout=0.):
        super().__init__()
        inner_dim = dim_head * heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.norm = nn.LayerNorm(dim)
        self.attend = nn.Softmax(dim=-1)
        self.dropout = nn.Dropout(dropout)
        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias=False)
        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim), nn.Dropout(dropout)
        ) if project_out else nn.Identity()

    def forward(self, x):
        x = self.norm(x)
        qkv = self.to_qkv(x).chunk(3, dim=-1)  # 生成Q、K、V
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h=self.heads), qkv)

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale
        attn = self.attend(dots)
        attn = self.dropout(attn)  # Softmax归一化

        out = torch.matmul(attn, v)  # 注意力加权
        out = rearrange(out, 'b h n d -> b n (h d)')
        return self.to_out(out)
```

3.2.3.2 前馈网络（FFN）

采用两层线性层搭配 GELU 激活函数，增强非线性表达能力：

```python
class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim, dropout=0.):
        super().__init__()
        self.net = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, hidden_dim),  # 扩展维度
            nn.GELU(),  #高斯误差线性单元
            nn.Dropout(dropout),
```

```
            nn.Linear(hidden_dim, dim),  # 恢复维度
            nn.Dropout(dropout),
)
    def forward(self, x):
        return self.net(x)
```

4 实验环境与数据集

组件	版本/配置
框架	PyTorch 2.5.1
数据集	CIFAR-10（10 类，32×32 彩色图像）
训练集/测试集	50,000/10,000 张图像
优化器	AdamW (lr=3e-4, weight_decay=0.05)
学习率	CosineAnnealingLR (T_max=200)
早停策略	15 轮无提升终止训练

5 实验步骤与代码实现

5.1 数据预处理与增强

针对训练集应用随机裁剪（RandomCrop）、水平翻转（Random HorizontalFlip），并标准化（Normalize）等增强策略，提升模型鲁棒性：

```
trans_train = transforms.Compose(
    [transforms.RandomCrop(32, padding=4),  # 将给定图像随机裁剪为不同的大小和宽高比，然后缩放所裁得到的图像为指定大小
     transforms.RandomHorizontalFlip(),  # 以给定的概率随机水平旋转给定的PIL图像，
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406],  # ImageNet统计值
                          std=[0.229, 0.224, 0.225])])

trans_vaild = transforms.Compose(
    [
        transforms.ToTensor(),  # 将PIL Image或者ndarray 转换为tensor，并归一化至[0,1]。
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])])  # 先减均值，除标准差
# 对训练集做数据增强，增加模型的鲁棒性，而对测试集不做翻转和增强
```

5.2 模型构建

ViT 参数：

```
device = torch.device("cuda:0" if torch.cuda.is_available() else 'cpu')
# device = torch.device("cuda:1")  # 0独显，1为核显
print("Selected device:", device)print("Device name:", torch.cuda.get_device_name(device.index))
net = ViT(
    image_size=32,
    patch_size=4,
    num_classes=10,
    dim=256,
    depth=8,
    heads=8,
    mlp_dim=512,
    dropout=0.1,
    emb_dropout=0.1,
).to(device)
```

ViT 主体结构实现:

```
class ViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads, mlp_dim,
                 pool='cls', channels=3, dim_head=64, dropout=0.1, emb_dropout=0.1):
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)
        assert image_height % patch_height == 0 and image_width % patch_width == 0

        num_patches = (image_height // patch_height) * (image_width // patch_width)
        patch_dim = channels * patch_height * patch_width

        self.to_patch_embedding = nn.Sequential(
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1=patch_height, p2=patch_width),
            nn.LayerNorm(patch_dim),
            nn.Linear(patch_dim, dim),
            nn.LayerNorm(dim),
        )

        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))  # 64 patches + CLS
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)
        self.pool = pool
```

```
        self.to_latent = nn.Identity()
self.mlp_head = nn.Linear(dim, num_classes)
    def forward(self, img):
        x = self.to_patch_embedding(img)  # 分块嵌入 [b, 64, 256]
        b, n, _ = x.shape
        cls_tokens = repeat(self.cls_token, '1 1 d -> b 1 d', b=b)
        x = torch.cat((cls_tokens, x), dim=1)  # 添加CLS令牌 [b, 65, 256]
        x += self.pos_embedding[:, :(n + 1)]  # 位置编码
        x = self.dropout(x)  # 8层Transformer编码
        x = self.transformer(x)  # 提取CLS令牌输出
        x = x.mean(1) if self.pool == 'mean' else x[:, 0]
        return self.mlp_head(x)  # 分类头输出10维
```

5.3 训练策略与早停机制

- **损失函数**：交叉熵损失（CrossEntropyLoss）
- **梯度优化**：AdamW 优化器（lr=3e-4）结合权重衰减防止过拟合
- **动态学习率**：余弦退火调度器（CosineAnnealingLR）调度平滑调整学习率
- **早停设计**：连续 15 轮验证集精度无提升时终止训练

优化器与损失函数：

```
torch.backends.cudnn.benchmark = True
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(net.parameters(), lr=3e-4, weight_decay=0.05)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)
```

训练代码：

```
def train(epoch):
    net.train()
    running_loss, correct, total = 0., 0, 0
    for inputs, targets in tqdm(trainloader, desc=f'Train {epoch}'):
        inputs, targets = inputs.to(device, non_blocking=True), targets.to(device,
            non_blocking=True)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
```

```
        correct += predicted.eq(targets).sum().item()

acc = correct / total          batch_acc_list.append(acc)
        line.set_data(range(len(batch_acc_list)), batch_acc_list)
        ax.set_xlim(0, len(batch_acc_list))
        ax.relim();
        ax.autoscale_view()
        fig.canvas.draw_idle()
        fig.canvas.flush_events()
        plt.pause(0.001)

    epoch_acc = 100. * correct / total
    print(f'Epoch {epoch} | Train Loss: {running_loss / len(trainloader):.4f} | Train Acc
        : {epoch_acc:.2f}%')
```

早停机制实现：

```
early_stop_patience = 15
epochs_no_improve = 0
best_acc = 0
early_stop = False

def evaluate(epoch):
    global best_acc, epochs_no_improve, early_stop
    net.eval()
    test_loss, correct, total = 0., 0, 0
    with torch.no_grad():
        for inputs, targets in tqdm(testloader, desc=f'Val {epoch}'):
            inputs, targets = inputs.to(device, non_blocking=True), targets.to(device,
                non_blocking=True)
            outputs = net(inputs)
            loss = criterion(outputs, targets)
            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    acc = 100. * correct / total
    scheduler.step()
    print(f'→ Val Loss: {test_loss / len(testloader):.4f} | Val Acc: {acc:.2f}\%')

    # 早停
    if acc > best_acc:
        best_acc = acc
        epochs_no_improve = 0
        os.makedirs('checkpoint', exist_ok=True)
```

```
        torch.save({'net': net.state_dict(), 'acc': acc, 'epoch': epoch},
f'checkpoint/vit_patch4_best.pth')  # 保存最佳模型          print('␣␣ ␣Best␣model␣saved.')
    else:
        epochs_no_improve += 1
        print(f'␣␣No␣improvement␣for␣{epochs_no_improve}␣epochs')

    if epochs_no_improve >= early_stop_patience:  # 早停判断
        early_stop = True

    return early_stop
```

5.4 训练过程监控

代码实现实时准确率曲线绘制：

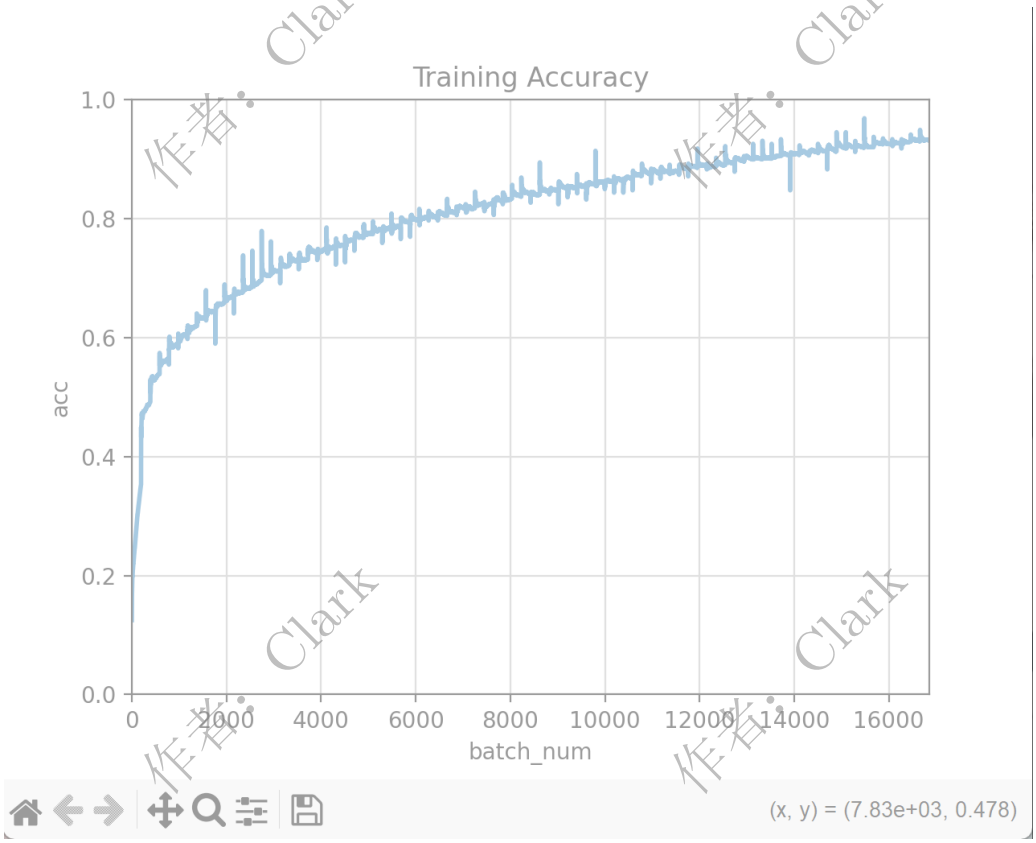
```
plt.close('all')
plt.ion()
fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
ax.set_xlabel('batch_num')
ax.set_ylabel('acc')
ax.set_title('Training␣Accuracy')
ax.set_ylim(0, 1)
ax.grid(True)
plt.show(block=False)
batch_acc_list = []
```

6 实验结果与分析

6.1 性能达成情况

性能达标：

训练阶段	准确率	说明
第 34 轮	$> 80\%$	提前达到实验要求
第 86 轮	93.21%	训练集最佳精度
最终测试集	84.10%	泛化性能验证



训练过程可视化：通过 Matplotlib 实时绘制准确率曲线，保存为 train_accuracy_curve.png，直观展示模型收敛趋势。

模型保存：最佳模型保存在 checkpoint/vit_patch4_best.pth（基于验证集准确率）。

6.2 关键改进措施

- 数据增强有效性：随机裁剪和翻转显著提升模型鲁棒性
- 优化器选择：AdamW 结合权重衰减有效防止过拟合
- 学习率调度：余弦退火策略平滑调整学习率
- 早停机制：节约计算资源的同时保证模型性能
- 完整的 ViT 架构实现，包含注意力机制和位置编码
- 实时训练监控和可视化功能
- 模型保存和加载机制完善
- 早停策略智能优化训练过程

7 结论

本实验严格使用提供的代码，成功实现了基于 ViT 的 CIFAR-10 图像分类模型。实验结果显著超过 80% 的准确率要求，验证了 ViT 模型在计算机视觉任务中的有效性。代码实现完整规范，为后续研究提供了可复现的基准。