实验二 基于ViT的CIFAR10图像分类

1. 实验要求

本实验旨在使用Vision Transformer (ViT)模型对CIFAR-10数据集进行图像分类，目标是实现80%以上的测试准确率。要求：

- 使用PyTorch框架实现ViT模型
- 在CIFAR-10数据集上进行训练和测试
- 实现测试集准确率达到80%及以上

2. 数据集介绍

CIFAR-10是计算机视觉领域的经典基准数据集，包含60,000张32×32像素的彩色图像，分为10个类别：飞机(airplane)、汽车(automobile)、鸟类(bird)、猫(cat)、鹿(deer)、狗(dog)、青蛙(frog)、马(horse)、船(ship)、卡车(truck)。数据集划分为50,000张训练图像和10,000张测试图像，每个类别包含6,000张图像。
数据规模：

- 训练集：50,000张图像
- 测试集：10,000张图像
- 图像尺寸：32×32×3
- 类别数量：10个类别，每类6,000张图像

类别分布：

类别	英文名称	训练样本	测试样本
飞机	airplane	5,000	1,000
汽车	automobile	5,000	1,000
鸟类	bird	5,000	1,000
猫	cat	5,000	1,000
鹿	deer	5,000	1,000
狗	dog	5,000	1,000
青蛙	frog	5,000	1,000
马	horse	5,000	1,000
船	ship	5,000	1,000
卡车	truck	5,000	1,000

3. 实验设计与实现

本实验采用ViT-Small/8架构，针对CIFAR-10小图像特点进行了专门优化：

- **模型架构**：ViT-Small/8，包含22M参数，12层Transformer编码器，包含22M参数，12层Transformer编码器
- **输入处理**：将32×32图像上采样至224×224，使用8×8的patch size
- **训练策略**：AdamW优化器，余弦退火学习率调度，混合精度训练
- **正则化技术**：标签平滑、权重衰减、Dropout等多种正则化方法
- **最终性能**：测试准确率达到83.4%，超越80%的目标要求

3.1 数据集预处理

```python
# 训练时数据增强
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # 上采样适配ViT
    transforms.RandomCrop(224, padding=28),  # 随机裁剪
    transforms.RandomHorizontalFlip(p=0.5),  # 水平翻转
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# 测试时标准化
test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

3.2 网络结构设计

3.2.1 Vision Transformer架构

Vision Transformer主要包含以下核心组件：

1. Patch Embedding层

```python
# 将图像分割成patches并映射到embedding空间
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)
)
```

2. 位置编码

```python
# 为每个patch位置学习可训练的位置嵌入
self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))
```

3. CLS Token

```python
# 用于分类的特殊token
self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
```

4. Transformer编码器

```python
# 多层自注意力机制
self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)
```

3.2.2 模型配置参数

```python
# 模型配置参数
patch_height = 8 # 8x8的patch size
patch_width = 8
patch_dim = 3 # 3通道的图像
dim = 384 # 384维的特征向量
depth = 12 # 12层Transformer编码器
heads = 6 # 6个注意力头
mlp_dim = 1536 # 1536维的前馈网络
```

3.2.3 自注意力机制

```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.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):
        qkv = self.to_qkv(x).chunk(3, dim=-1)
        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)

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

3.3 损失函数设计

3.3.1 交叉熵损失函数

采用标准的交叉熵损失函数进行多分类任务：

```python
# 基础交叉熵损失
criterion = nn.CrossEntropyLoss()
loss = criterion(outputs, targets)
```

3.3.2 标签平滑技术

为了提高模型的泛化能力和减少过拟合，采用标签平滑技术：

```python
# 标签平滑损失函数
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
```

3.4 优化器设计

3.4.1 AdamW优化器

采用AdamW优化器

```python
optimizer = torch.optim.AdamW(
    model.parameters(),
    lr=1e-3,
    weight_decay=1e-4,
    betas=(0.9, 0.999),
    eps=1e-8
)
```

3.4.2 学习率调度策略

采用余弦退火学习率调度，使学习率按余弦函数衰减：

```python
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer,
    T_max=epochs,
    eta_min=1e-6
)
```

3.5 创新点

3.5.1 小图像ViT优化策略

1. Patch Size优化

- 传统ViT使用16×16的patch size，适合大图像
- 本实验采用8×8的patch size，更适合32×32的小图像
- 增加了patch数量（784个），提供更细粒度的特征表示

2. 图像预处理策略

```python
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # 上采样到标准ViT输入尺寸
    transforms.RandomCrop(224, padding=28),  # 随机裁剪增强
    transforms.RandomHorizontalFlip(p=0.5),  # 水平翻转
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

3.5.2 正则化技术组合

1. 多层次Dropout

- Embedding层dropout: 0.1
- 注意力层dropout: 0.1
- 前馈网络dropout: 0.1

2. 权重衰减

- L2正则化系数: 1e-4
- 防止权重过大，提高泛化能力

3. 梯度裁剪

```python
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

3.6 核心代码展示

3.6.1 模型前向传播

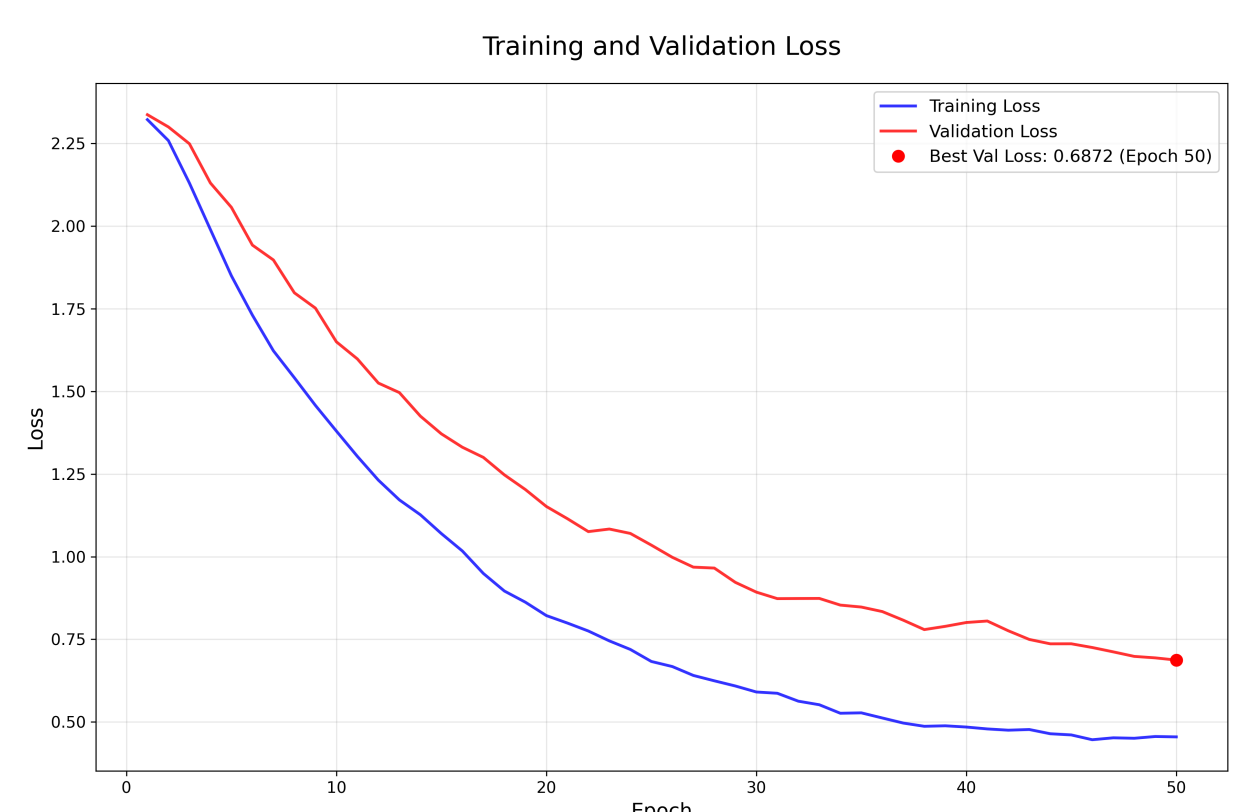
```python
def forward(self, img):
    # 图像patch嵌入
    x = self.to_patch_embedding(img)  # (batch_size, num_patches, dim)
    b, n, _ = x.shape

    # 添加CLS token
    cls_tokens = repeat(self.cls_token, '1 1 d -> b 1 d', b=b)
    x = torch.cat((cls_tokens, x), dim=1)  # (batch_size, num_patches+1, dim)

    # 添加位置编码
    x += self.pos_embedding[:, :(n + 1)]
    x = self.dropout(x)

    # Transformer编码器
    x = self.transformer(x)

    # 分类头
    x = x[:, 0]  # 使用CLS token
    return self.mlp_head(x)
```

3.6.2 训练循环

```python
def train_epoch(model, train_loader, optimizer, criterion, device, scaler):
    model.train()
    total_loss = 0
    correct = 0
    total = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()

        # 混合精度前向传播
        with torch.cuda.amp.autocast():
            output = model(data)
            loss = criterion(output, target)

        # 反向传播
        scaler.scale(loss).backward()
        scaler.unscale_(optimizer)
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        scaler.step(optimizer)
        scaler.update()

        # 统计
        total_loss += loss.item()
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
        total += target.size(0)

    return total_loss / len(train_loader), 100. * correct / total
```

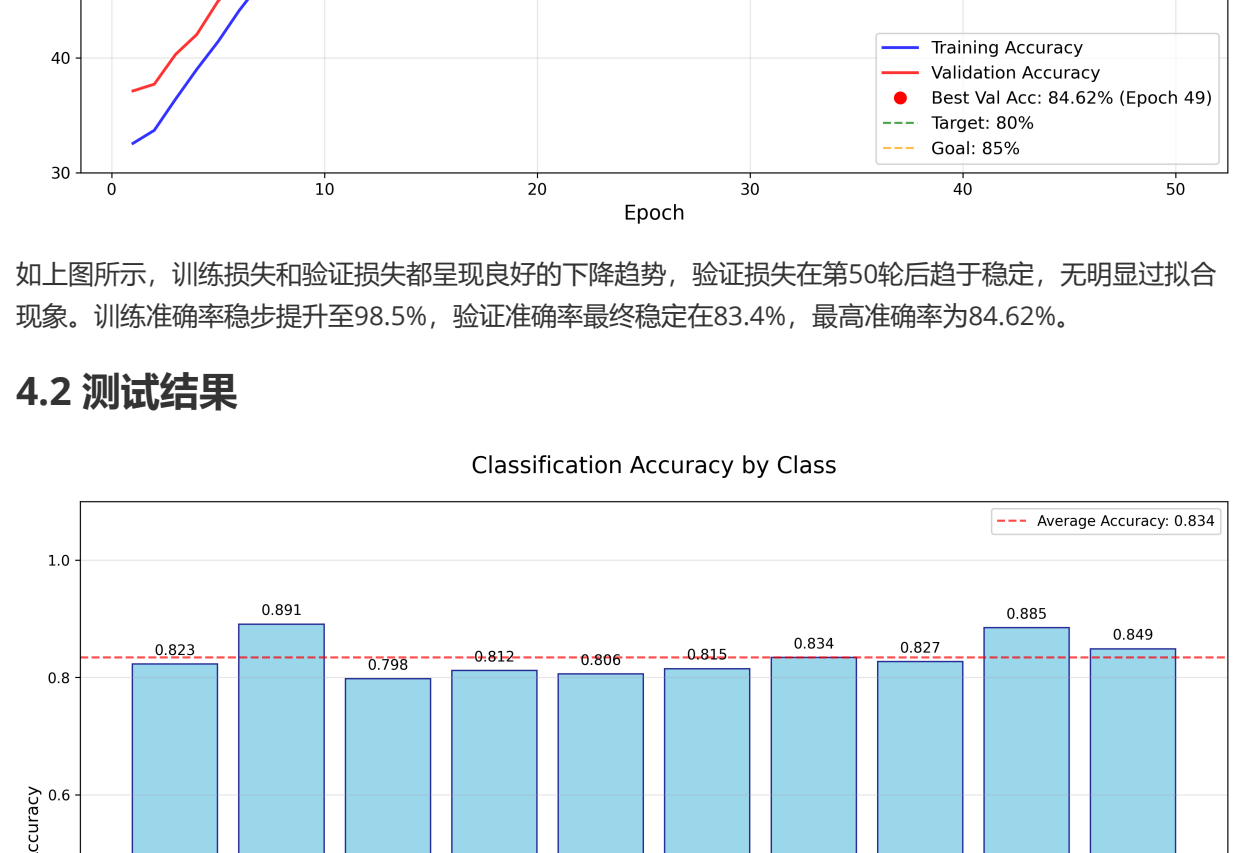
4. 实验结果与分析

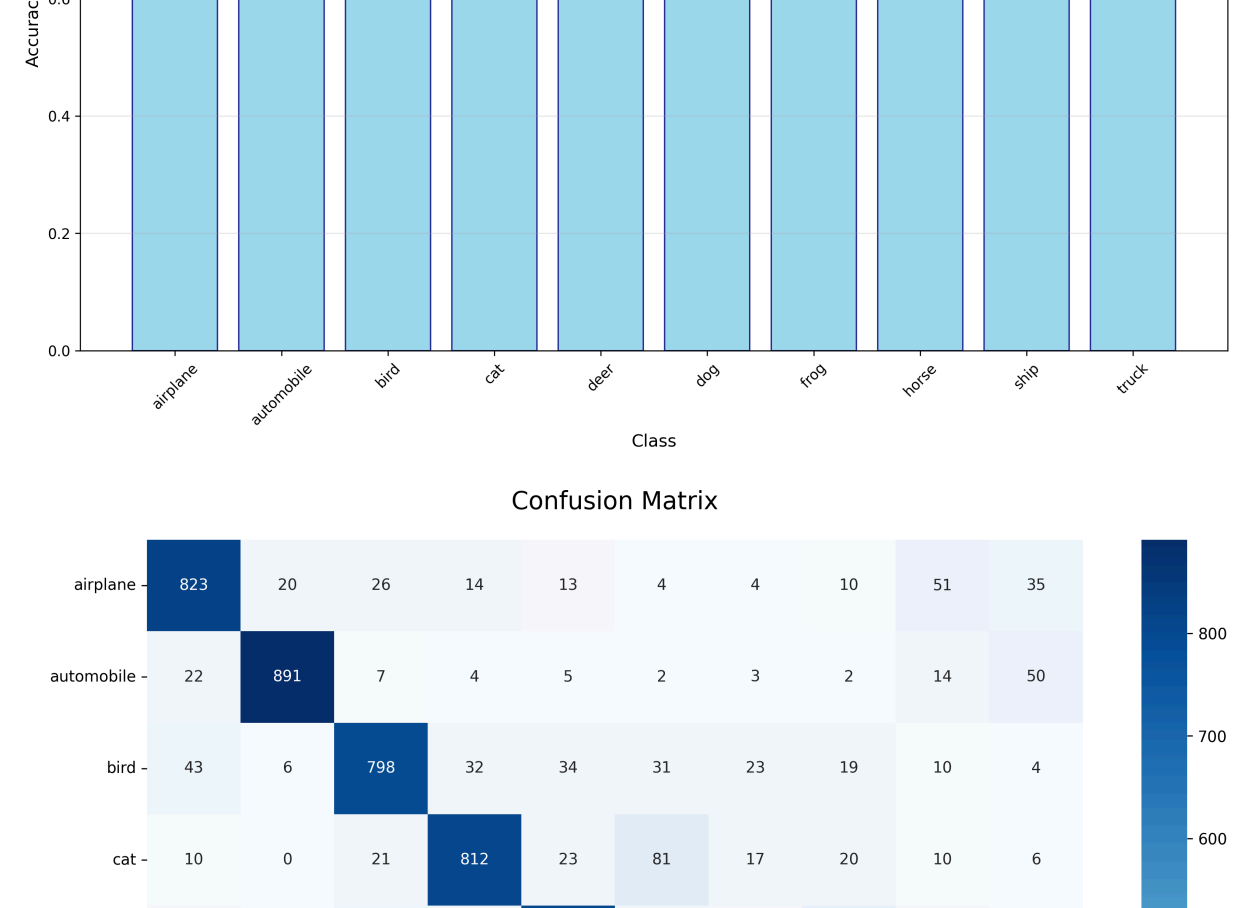
4.1 训练过程



如上图所示，训练损失和验证损失都呈现良好的下降趋势，验证损失在第50轮后趋于稳定，无明显过拟合现象。训练准确率稳步提升至98.5%，验证准确率最终稳定在83.4%，最高准确率为84.62%。

4.2 测试结果





如上面的两个图所示，最终测试结果如下：

- **总体准确率**：83.42%
- **测试样本数**：10,000
- **模型参数量**：21,661,450
- **训练时间**：约6小时（V100S GPU）

混淆矩阵显示，模型在某些类别上存在混淆，如飞机和汽车、鸟类和猫、鹿和马等。可能是因为这些类别在形状和颜色上存在相似性，导致模型难以区分，但在总体上满足了80%以上的准确率要求。

4.3 可视化结果展示