|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ---------------------------------------------装---------------------------------------------------订-----------------------------------------------线----------------------------------------------  班级 班 姓名 学号 教学班名称 序号 任课教师 胡苏 - ---------------------------------------------------密----------------------------------------封-------------------------------------------线----------------------------------------------------- | | **广东财经大学线上考试答题纸（格式二）**  课程 AI处理器架构与编程 考试时间  成绩 评阅人 胡苏   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 评语：   |  |  |  | | --- | --- | --- | | **检查项目** | **权重** | **得 分** | | **(1)** **选题合理性：**选题与本课程的关联性。 | 10 |  | | **(2)** **材料完整性**：包括数据获取及清洗，特征工程，模型选择及对比，模型调优，模型评估及结果可视化展示等；材料真实，内容翔实。 | 35 |  | | **(3)** **模型设计的科学性和合理性**：主要考虑特征工程阶段及模型选择阶段的分析与设计过程的科学性及合理性。 | 35 |  | | **(4)** **文档规范性**：项目报告规范，内容完整，术语表达准确，描述清晰，结果可视化直观。 | 20 |  | | **合计** | **100** |  | |   ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝  题目：综合应用本门课程所学编程方法技术解决实际问题，问题从以下项目选题中任选一道完成。  提交文档要求：设计报告格式参照设计模板（在正文处给出）。报告模板格式仅供参考，注意图文并茂，以最清晰详细和可理解方式展示报告内容。  项目选题：  **【选题1】 基于VIT模型，完成模型训练和预测，随机选择测试集一张图片，给出预测结果。**  **论文：****<https://arxiv.org/abs/2010.11929>**  **教程：<https://aistudio.baidu.com/projectdetail/3499029?ad-from=17070>**  **【选题2】完成SAM模型的部署，选取多张照片进行抠像和语义分割。进阶：也可以通过文字来指导抠像。**  **论文：<https://arxiv.org/abs/2304.02643>教程：<https://blog.csdn.net/Father_of_Python/article/details/130004935>**  **【选题3】** **部署Mixtral 8x7B，并选取C-EVAL数据集当中前100条，来评价模型性能。给出预测结果和准确率并将结果可视化。**  **论文：<https://huggingface.co/docs/transformers/model_doc/mixtral>**  **教程：<https://www.bilibili.com/video/BV1Mi4y1e7px/?vd_source=5bd27da54603d28a7ef02b16a2cd8ad8>**  **【选题4】基于Langchain框架和向量库，实现大语言模型联网搜索。**  **【选题5】 华为mindspore官网学术论文研读、分析与复现，共五篇，任选一篇即可（官网有论文和代码）**  **网址：<https://www.mindspore.cn/resources/papers>**   1. MOSO: Decomposing MOtion, Scene and Object for Video Prediction 2. Short‐term and long‐term memory self‐attention network for segmentation of tumours in 3D medical images 3. Free form deformation and symmetry constraint‐based multi‐modal brain image registration using generative adversarial nets   4.Fully Transformer Network for Change Detection of Remote Sensing Images  5.Learning Progressive Modality-shared Transformers for Effective Visible-Infrared Person Re-identification  **【选题6】通义千问-72B-Chat阅读、理解、分析与开发。阅读官网模型介绍，paper，代码，并进行开发实验，以上全部需要提供图文并茂过程。注意可以通过阿里云快速进行本地实现。**  **<https://modelscope.cn/models/qwen/Qwen-72B-Chat/summary>**  **【选题7】基于百度飞浆的模型部署实践分析。**  **先注册百度飞浆，100小时免费使用：**  **<https://aistudio.baidu.com/projectoverview/public>**    在任选模型进行部署，开发实验：  <https://aistudio.baidu.com/modelsoverview>    网站每个模型均有详细教程供参考：    **【选题8】华为实验手册《HCIA-AIV3.0LabGuides220427》第8-12章，五个AI服务实验，复现其中一个实验，图文并茂，给出完整实现过程，最终结果，说明，并对遇到问题如何解决详细说明。（8-12章：人脸识别，图像识别，文字识别，内容审核，语音合成和语音识别）** | |
| （正文）  **《基于VIT模型的训练与预测**  **——ViT图像分类任务》项目报告**  **一、项目概况**  （五号宋体）  实验环境：  python版本：python3.11  paddle版本：paddlepaddle-gpu 2.6.0  IDE：JupyterLab  GPU版本：RTX3060    项目基本介绍：  随着深度学习技术的快速发展，图像分类作为计算机视觉领域的重要任务之一，受到了广泛关注。ViT（Vision Transformer）作为一种新型的基于Transformer的图像分类模型，以其全局建模能力和自注意力机制在图像分类任务中取得了显著效果。本项目旨在使用PaddlePaddle深度学习框架复现ViT模型，并完成图像分类任务的训练，以进一步探索ViT模型在图像分类领域的应用潜力。  项目实现功能：  该项目实现了一个完整的基于Vision Transformer（ViT）的图像分类模型的训练、验证和测试流程。通过训练过程，模型能够学习到从图像中提取有用特征并进行分类的能力。训练过程中使用了日志记录器来监控训练状态，并通过验证集来评估模型的性能。最终，使用测试集对训练好的模型进行评估，并输出测试集上的准确率。  **二、数据获取及分析**  （五号宋体）  2.1 数据集基本介绍  源数据集如下所示（包含test、train、validation文件夹）    每个文件夹中包含大量蔬果类的图片，用于给程序分析。    2.2 数据处理部分代码介绍与分析  在线运行Jupyter的cell，结果如下所示：    **代码分析：**  2.2.1.读取数据集    1.导入必要的库：  ·paddle 和 paddle.nn：这是 PaddlePaddle 的基础库和神经网络模块。  ·os：虽然在这段代码中并未直接使用，但通常用于操作系统相关的操作。  ·PIL.Image：用于读取和处理图像。  ·numpy：用于数值计算。  ·paddle.vision.transforms：这是 PaddlePaddle 的视觉变换模块，用于图像预处理。  定义训练数据预处理流程 train\_trans：  ·T.Resize(300)：将图像大小调整为 300x300。  ·T.RandomResizedCrop(size=224)：从输入图像中随机选择一个区域，然后将其调整为 224x224 大小。这是一种数据增强方法，可以帮助模型学习更鲁棒的特征。  ·T.RandomHorizontalFlip()：以一定的概率对图像进行水平翻转。这同样是一种数据增强方法。  ·T.ToTensor()：将 PIL 图像或 NumPy ndarray 转换为 PaddlePaddle Tensor，并将数据范围从 [0, 255] 缩放到 [0.0, 1.0]。  定义测试数据预处理流程 test\_trans：  ·T.Resize(300) 和 T.RandomResizedCrop(size=224)：为了与训练时保持一致  ·T.ToTensor()：与训练数据预处理中的这个步骤相同。  ·ColorJitter 在训练数据预处理中被注释掉了，但在某些情况下，它可能有助于模型学习更鲁棒的特征。  2.2.2.定义数据集接口  **preprocess 函数：**  这个函数接受一个 PIL 图像对象 img 和一个布尔值 is\_train 作为输入。  函数首先将图像转换为 RGB 格式（尽管这通常是 PIL 图像加载时的默认格式，但这里进行了明确的转换）。  如果 is\_train 为 True，则对图像应用训练时的数据增强变换 train\_trans；否则，应用测试时的数据变换 test\_trans。  最后，返回处理后的图像。  注意：这里假设 train\_trans 和 test\_trans 已经在其他地方定义好了，但在这段代码中并没有给出它们的定义。  **MyDataset 类：**  \_\_init\_\_ 方法：  初始化时接受三个参数：数据集路径 path\_dir、根目录 root\_dir（默认为 'work'）和是否为训练数据 train（默认为 True）。  通过遍历 path\_dir 下的所有子目录，构建图像路径列表 self.img\_path。  创建一个标签映射 self.label\_map，将每个子目录名映射到一个唯一的整数标签。  \_\_getitem\_\_ 方法：  根据给定的索引 idx，从 self.img\_path 中获取对应的图像路径。  使用 PIL 打开图像，并使用 preprocess 函数进行预处理。  从图像路径中提取标签（这里假设图像路径中包含子目录名作为标签）。  将标签转换为 PaddlePaddle Tensor，并设置数据类型为 'int64'。  返回预处理后的图像和标签。  \_\_len\_\_ 方法：  返回图像路径列表的长度，即数据集中的图像数量。  **创建数据集实例：**  最后，代码创建了三个数据集实例：train\_set、val\_set 和 test\_set，分别对应于训练集、验证集和测试集。  2.2.3.可视化  运行可视化cell结果如下所示：      **三、特征工程**  （五号宋体）   1. 读取数据集部分特征处理的实现   1.尺寸调整 (T.Resize(300)):  这一步将图像的尺寸调整为300x300（假设图像是正方形的，否则它将被等比例缩放，但可能不是严格的300x300）。这一步的目的是为了后续处理，例如裁剪，能在一个统一的尺寸上进行。  2.随机裁剪 (T.RandomResizedCrop(size=224)):  这一步在300x300的图像上随机选择一个区域，然后将其尺寸调整为224x224。这有助于模型学习图像中的局部特征，因为模型在每次迭代时都会看到不同的裁剪区域。  3.随机水平翻转 (T.RandomHorizontalFlip()):  这一步以一定的概率（默认为0.5）将图像水平翻转。这有助于模型学习图像的方向不变性，因为许多图像在水平方向上是对称的（例如，猫或狗的脸）。  4.颜色抖动 (T.ColorJitter(0.4, 0.4, 0.4, 0.4)):  这一步在代码中是被注释掉的，但如果取消注释，它会在图像的亮度、对比度、饱和度和色调上添加随机的抖动。这有助于模型学习对颜色变化的鲁棒性，因为现实世界中的图像可能会因光照、相机设置等因素而有所不同。  5.转换为张量 (T.ToTensor()):  这一步将PIL图像或NumPy数组转换为Paddle框架可以理解的格式，即张量（Tensor）。在转换过程中，像素值还会被归一化到[0, 1]的范围内（如果原始像素值是[0, 255]的话）。   1. 定义数据集接口部分特征处理的实现   1.数据预处理函数 (preprocess): 该函数接受一个图像对象 img 和一个布尔值 is\_train。根据 is\_train 的值，它决定使用 train\_trans 还是 test\_trans 来对图像进行预处理。这包括颜色空间的转换（确保图像是RGB格式）、随机裁剪和大小调整（在训练模式下）、以及将图像转换为张量。  2.数据集类 (MyDataset):  \_\_init\_\_ 方法初始化了数据集，并遍历了指定目录下的所有图像文件，将它们的路径存储在 self.img\_path 中。同时，它还创建了一个标签映射 self.label\_map，将每个子目录（假设每个子目录代表一个类别）映射到一个整数标签。  \_\_getitem\_\_ 方法根据给定的索引 idx 从 self.img\_path 中加载图像，并调用 preprocess 函数进行预处理。然后，它使用 img\_path 中的子目录名作为标签，并通过 self.label\_map 将其转换为整数。最后，它返回预处理后的图像和对应的标签。  \_\_len\_\_ 方法返回数据集中图像的总数。  3.实例化数据集: 通过调用 MyDataset 类的构造函数，创建了三个数据集对象：train\_set、val\_set 和 test\_set，分别对应于训练集、验证集和测试集。注意，对于 test\_set，你明确地将 train 参数设置为 False，以确保在测试时使用不同的预处理流程（没有随机裁剪）。  **四、模型设计、选择**  （五号宋体）  I、VIT的具体分析  VIT结构图如下所示：  VIT  以下是对上述结构图的具体分析：  VIT（Vision Transformer）结构示意图分析  VIT，即Vision Transformer，是一种将自然语言处理中的Transformer模型应用于计算机视觉任务的深度学习架构。  **1. Patch+Position Embedding**  图像在输入到Transformer模型之前，需要先被处理成一系列的“patches”（图像块）。每个patch可以看作是一个小的图像区域，并且会被转换成一个向量（即embedding）。此外，为了使模型能够理解patch之间的空间关系，通常还需要将位置信息也嵌入到每个patch中。这就是所谓的“Patch+Position Embedding”。  **2. Linear Projection of Flattened Patches**  在处理完patches之后，为了适应Transformer的输入格式，通常需要将二维的patches数据展平（flatten）并经过一个线性层进行投影，将其转换为模型可以处理的特征向量。这个步骤就是“Linear Projection of Flattened Patches”。  **3. Multi-Head Attention**  Multi-Head Attention是Transformer模型的核心组件之一。它通过将输入特征分成多个“头”（head），并在每个头上独立执行注意力机制，然后将结果合并，从而能够捕获输入数据中的多种不同关系。这种多头机制有助于提高模型的表示能力。  **4. MLP (Multi-Layer Perceptron)**  在Multi-Head Attention之后，通常会跟一个或多个多层感知器（MLP）。MLP是一种全连接的神经网络，能够对输入数据进行非线性变换，从而进一步提取特征。在Transformer模型中，MLP通常用于对Multi-Head Attention的输出进行加工。  **5. Normalization (Norm)**  归一化层用于在模型的训练过程中稳定梯度，加速收敛。在Transformer模型中，归一化层通常位于每个子层（如Multi-Head Attention和MLP）之后。  **6. Transformer Encoder**  Transformer Encoder由多个相同的子层堆叠而成，每个子层都包含一个Multi-Head Attention和一个MLP。通过堆叠多个这样的子层，Transformer Encoder能够逐步提取输入数据的深层特征。  **7. Extra learnable [class] embedding**  这是一个额外的可学习嵌入，通常用于表示图像的全局信息或类别信息。在分类任务中，这个嵌入可以与patch embeddings一起输入到Transformer Encoder中，以帮助模型更好地理解图像的整体内容。  **II.多头注意力机制**  多头注意力机制（Multi-Head Attention）是Transformer模型中的一个关键组件。  **初始化函数 \_\_init\_\_:**  ·embed\_dim: 输入嵌入的维度，即每个token的维度。  ·num\_heads: 注意力机制的头数。  ·qkv\_bias: 是否在QKV线性层中使用偏置项。  ·qk\_scale: 对Q和K的点积结果进行缩放的因子，默认为embed\_dim -0.5，这是为了防止点积结果过大，导致softmax函数进入梯度饱和区。  ·dropout: dropout的比例，但在这段代码中并未使用。  **属性**  ·embed\_dim, num\_heads, head\_dim, all\_head\_dim: 分别表示嵌入维度、头数、每个头的维度和所有头的总维度。  ·qkv: 这是一个线性层，它将输入的嵌入投射到QKV三个空间的拼接上。  ·scale: 缩放因子，用于缩放Q和K的点积结果。  ·softmax: softmax函数，用于将点积结果转换为概率分布。  ·proj: 这是一个线性层，用于将多头注意力的输出投射回原始的嵌入维度。  **方法 transpose\_multi\_head**  这个方法用于将输入的张量x重新形状并转置，以便为每个头单独计算注意力。  ·x.shape[:-1]: 获取除了最后一个维度之外的所有维度的大小。  ·x.reshape(new\_shape): 将x重新形状为包含头数的四维张量。  ·x.transpose([0, 2, 1, 3]): 转置张量，将头数（第二个维度）移动到第二个位置，以便对每个头单独计算注意力。  **forward 方法**  这个方法定义了模型在前向传播时如何处理输入数据x。   1. 输入形状的分析：  |  | | --- | | batch, num\_patches, \_ = x.shape |   这行代码解析了输入张量x的形状，其中batch是批量大小，num\_patches是序列长度或补丁数量，第三个维度是嵌入维度   1. QKV投射：  |  | | --- | | qkv = self.qkv(x).chunk(3, -1) |   通过qkv线性层将输入x投射到一个更高维的空间，该空间的维度是原始嵌入维度的三倍（因为需要同时表示查询（Q）、键（K）和值（V））。chunk(3, -1)方法将投射后的张量在最后一个维度上分成三块，分别对应Q、K、V。   1. 转置多头  |  | | --- | | q, k, v = map(self.transpose\_multi\_head, qkv) |   使用map函数和前面定义的transpose\_multi\_head方法对Q、K、V进行转置和重塑，以便为每个头单独计算注意力。   1. 注意力计算：  |  | | --- | | attn = paddle.matmul(q, k, transpose\_y=True)  attn = self.softmax(attn self.scale) |   首先计算查询（Q）和键（K）的转置之间的点积，得到注意力分数。然后，这些分数乘以缩放因子self.scale（通常是1 / sqrt(dk)，其中dk是键的维度），并通过softmax函数进行归一化，得到注意力权重。   1. 加权求和：  |  | | --- | | out = paddle.matmul(attn, v) |   使用注意力权重对值（V）进行加权求和，得到每个头的输出。   1. 重塑和投射：  |  | | --- | | out = out.transpose([0, 2, 1, 3])  out = out.reshape([batch, num\_patches, -1])  out = self.proj(out) |   首先，通过转置操作将头的维度移动到正确的位置。然后，使用reshape方法将所有头的输出合并成一个张量，其形状与原始输入的形状相匹配（除了嵌入维度可能不同）。最后，通过proj线性层将合并后的张量投射回原始的嵌入维度。   1. 测试LayerNorm   **导入库和设置随机种子：**  ·导入PaddlePaddle和NumPy库。  ·设置NumPy的随机种子以确保可复现性。  **生成随机数据：**  ·使用NumPy生成一个形状为(2, 2, 3)的随机浮点数数组x\_data，并将其转换为PaddlePaddle张量x。  **定义LayerNorm层：**  ·定义了两个LayerNorm层：layer\_norm\_1和layer\_norm\_2。  ·layer\_norm\_1的归一化维度是3，意味着它将沿着最后一个维度（大小为3的维度）进行归一化。  ·layer\_norm\_2的归一化维度是(2,3)，但这里存在一个问题，因为LayerNorm通常不支持多个维度的归一化（它通常只归一化一个维度），除非是在特定的使用场景中（如某些类型的卷积层之后）。在PaddlePaddle中，如果你传入一个大于1的元组作为归一化维度，通常只会取元组的最后一个元素作为归一化维度。  **通过LayerNorm层传递数据：**  将张量x传递给两个LayerNorm层，并分别得到输出layer\_norm\_out\_1和layer\_norm\_out\_2。  **手动计算Layer Normalization：**  ·首先，对一维数组x进行Layer Normalization的手动计算，计算均值t、标准差v，并进行归一化。  ·然后，尝试对二维数组y进行类似的计算，但这里有一个问题：Layer Normalization是对每个样本（或特征映射）的每个特征进行独立归一化的，而不是对整个二维数组进行归一化。   1. 实现EncoderLayer   1.初始化方法:  embed\_dim=768: 嵌入维度，通常与模型的隐藏层大小相对应。这里的默认值是 768，这是许多 Transformer 模型（如 BERT）中常用的维度。  num\_heads=12: 多头注意力机制中的“头”数。多头注意力允许模型同时关注输入序列的不同部分。  mlp\_ratio=4.0: 用于计算多层感知器（MLP）中隐藏层的大小。这个比率通常与嵌入维度相乘来确定 MLP 中间层的大小。  在初始化方法中，还定义了几个关键的神经网络层：  self.attn\_norm 和 self.mlp\_norm：Layer Normalization 层，用于标准化输入的分布，有助于模型训练的稳定性和收敛速度。  self.attn：注意力机制层，是 Transformer 模型中的核心组件，用于计算输入序列中不同位置的相关性。  self.mlp：多层感知器，用于在注意力层之后进一步转换和处理数据。  2.前向传播方法:  该方法描述了数据通过一个编码器层时的流程。  首先，输入数据 x 通过一个 Layer Normalization 层（self.attn\_norm），然后进行注意力计算（self.attn）。  注意力层的输出与原始输入 x（这里用 h 保存了原始输入）相加，形成所谓的“残差连接”。这种结构有助于梯度在反向传播时更有效地流动。  接下来，数据再次通过一个 Layer Normalization 层（self.mlp\_norm），然后传递给 MLP（self.mlp）。  MLP 的输出再次与之前的和（即注意力层的输出加上原始输入）相加，形成另一个残差连接。  最后，编码器层的输出是这个加了两次残差连接的结果。  **Encoder 类**  1.初始化方法:  除了嵌入维度 embed\_dim，还接受一个 depth 参数，表示要堆叠的编码器层的数量。  在循环中，根据 depth 参数创建相应数量的 EncoderLayer 实例，并将它们添加到一个列表中。  这个列表随后被转换为一个 nn.LayerList，这是一个特殊的 PaddlePaddle 容器，用于保存多个网络层。  最后，定义了一个额外的 Layer Normalization 层（self.norm），用于在所有编码器层处理完数据后进行最终的标准化。  2.前向传播方法:  输入数据 x 依次通过每个编码器层。  每个编码器层的输出都作为下一个层的输入。  在通过所有编码器层后，数据通过一个最终的 Layer Normalization 层（self.norm）。  编码器的最终输出是经过这个最终标准化处理的数据。   1. 实现Positionembedding   **初始化方法 \_\_init\_\_**  1.参数定义：  ·image\_size：输入图像的大小，默认为224x224。  ·patch\_size：每个图像块的大小，默认为16x16。  ·in\_channels：输入图像的通道数，对于RGB图像默认为3。  ·embed\_dim：嵌入维度，即每个图像块被转换成的向量维度，默认为768。  ·dropout：dropout率，用于防止过拟合。  2.计算图像块数量： 通过(image\_size // patch\_size) (image\_size // patch\_size)计算出图像被划分成多少个图像块（patches）。对于默认的224x224图像和16x16的图像块大小，结果是14x14=196个图像块。  3.图像块嵌入： 使用nn.Conv2D卷积层实现图像块嵌入。这个卷积层的核大小、步长都与patch\_size相同，因此它会将每个patch\_size x patch\_size的图像块转换成一个embed\_dim维的向量。  4.Dropout层： 用于防止模型过拟合。  5.类标记（Class Token）： 创建一个可学习的类标记，它是一个embed\_dim维的向量。在ViT模型中，这个类标记用于在最后的分类任务中代表整个图像的信息。  6.位置嵌入（Position Embedding）： 创建一个可学习的位置嵌入矩阵，其形状为[1, num\_patches + 1, embed\_dim]。+1是为了考虑类标记的位置。位置嵌入用于向模型提供每个图像块的位置信息，因为Transformer模型本身不具备处理序列位置信息的能力。  **前向传播方法 forward**  1.扩展类标记： 将类标记从形状[1, 1, embed\_dim]扩展到与输入图像批次大小相匹配的形状[N, 1, embed\_dim]。  2.图像块嵌入： 使用在初始化方法中定义的卷积层对输入图像进行图像块嵌入。  3.调整张量形状： 将嵌入后的张量从形状[N, embed\_dim, h', w']展平并转置为[N, h'w', embed\_dim]，以便与位置嵌入相加。  4.添加类标记： 在嵌入的图像块序列前面添加类标记。  5.添加位置嵌入： 将位置嵌入加到嵌入的图像块和类标记上。这一步是向模型提供位置信息的关键。  6.返回结果： 返回添加了位置嵌入的图像块嵌入和类标记的张量。   1. 实现MLP模块   **初始化方法 \_\_init\_\_**  1.参数定义：  ·embed\_dim：嵌入维度，表示输入和输出的特征维度。  ·mlp\_ratio：MLP的扩展比例，用于确定MLP中间层的维度。  ·dropout：dropout率，用于在训练过程中随机将部分神经元的输出置零，以防止过拟合。  2.网络层定义：  · self.fc1：第一个全连接层（Linear层），将输入维度从embed\_dim扩展到int(embed\_dim mlp\_ratio)。这里使用int()函数确保维度是整数。  ·self.fc2：第二个全连接层，将扩展后的维度重新映射回embed\_dim。  ·self.act：激活函数层，使用的是GELU（Gaussian Error Linear Unit）激活函数。  ·self.dropout：dropout层，用于在训练过程中随机置零部分神经元的输出。  3.前向传播方法 forward  ·全连接层1：x = self.fc1(x)，将输入x通过第一个全连接层进行维度扩展和特征变换。  ·激活函数：x = self.act(x)，对第一个全连接层的输出应用GELU激活函数，增加非线性。  ·Dropout层1：x = self.dropout(x)，在激活函数之后应用dropout，随机置零部分神经元的输出，以防止过拟合。  ·全连接层2：x = self.fc2(x)，将经过dropout处理的输出通过第二个全连接层，将维度重新映射回embed\_dim。  ·Dropout层2：x = self.dropout(x)，在第二个全连接层之后再次应用dropout。  ·返回结果：最后返回经过两层全连接网络、激活函数和dropout处理后的输出x。   1. 实现VIT模块   **初始化方法 \_\_init\_\_**  参数定义：  ·image\_size：输入图片的大小，默认为224x224。  ·patch\_size：将图片分割成patches的大小，默认为16x16。  ·in\_channels：输入图片的通道数，默认为3（RGB）。  ·num\_classes：目标类别的数量，默认为1000。  ·embed\_dim：编码后的嵌入维度，即每个patch的特征向量长度，默认为768。  ·depth：Encoder堆叠的数量，即Transformer的层数，默认为3。  ·num\_heads：Multi-Head Attention中的头数，但在代码中并未直接使用此参数。  ·mlp\_ratio：MLP（多层感知机）的维度缩放比例，用于确定MLP中间层的维度，但在代码中并未直接使用此参数。  ·qkv\_bias：是否使用QKV投射层的偏置，但在代码中并未直接使用此参数。  ·dropout：dropout率，用于防止过拟合。  ·attention\_dropout：Attention层的dropout率，但在代码中并未直接使用此参数。  ·droppath：DropPath正则化率，但在代码中并未直接使用此参数。  模块定义：  ·self.patch\_embedding：使用PatchEmbedding类（未在代码中给出）将输入图片分割成patches，并编码成特征向量，同时加入class token和位置编码。  ·self.encoder：使用Encoder类（未在代码中给出）堆叠多个Transformer编码器层，用于对patches的特征向量进行编码。  ·self.classfier：一个线性层，用于将class token的特征向量映射到目标类别的概率分布。  **前向传播方法 forward**  ·输入处理：  输入x是形状为[N, C, H, W]的图片张量，其中N是批次大小，C是通道数，H和W是图片的高度和宽度。  ·Patch Embedding：  使用self.patch\_embedding将输入图片x编码成patches的特征向量，并加入class token和位置编码。输出x的形状变为[N, num\_patches + 1, embed\_dim]，其中num\_patches是patches的数量，+1是class token。  ·Encoder：  使用self.encoder对patches的特征向量进行编码，输出仍然是形状为[N, num\_patches + 1, embed\_dim]的张量。  ·分类：  提取class token的特征向量（即x[:, 0]），并使用self.classfier将其映射到目标类别的概率分布。这里假设class token是patches特征向量序列中的第一个元素。  ·输出：  返回class token经过分类头映射后的结果，形状为[N, num\_classes]。   1. 测试VIT模块得出的结果   测试VIT结果如下：      以上的的参数可拿出部分参数进行分析：  **1.Conv2D-1:**  输入形状: [4, 3, 224, 224] 表示有4个样本，每个样本是一个3通道（RGB）的224x224图像。  输出形状: [4, 768, 14, 14] 表示经过卷积层后，特征图的尺寸缩小了（因为卷积和池化的效果），通道数变为768，高度和宽度变为14x14。但通常，在ViT中这一步可能是通过一个具有较大步长的卷积或者一个patch embedding层来完成，直接将图片切割成固定大小的patches，并将这些patches映射到高维空间。这里的卷积层可能起到了类似的作用。  参数数量: 590,592 是这个卷积层的参数总数。  **2.PatchEmbedding-1:**  这一层负责将图片切割成patches，并将这些patches嵌入到高维空间。从输出形状可以看出，每个样本被切割成了197个patches（可能是14x14的patches，但需要具体实现来确定），每个patch被嵌入到了一个768维的空间。  参数数量: 152,064 可能是与位置编码相关的参数或者patch embedding过程中的一些变换参数。  3.后面的层主要构成了Transformer的编码器部分，包括多个重复的模块，如Attention-3, Mlp-1, EncoderLayer-1等。每个编码器层通常包括自注意力机制（由Softmax操作实现）和前馈神经网络（Mlp）。此外，LayerNorm层用于层归一化，GELU是激活函数，Dropout用于防止过拟合。  ·输入/输出形状: 在整个编码器部分，输入和输出的形状保持不变，都是[4, 197, 768]，意味着模型在处理过程中保持了patches的数量和嵌入维度。  ·参数数量: 各层的参数数量显示了模型的复杂度。例如，Linear层具有大量的参数，因为它们负责在高维空间中进行线性变换。  ·值得注意的是，Softmax和Dropout层没有参数，它们的功能是基于输入数据进行非线性变换或正则化，而不需要学习额外的权重。    **以上运行结果说明的有：**  **1.参数数量：**  Total params: 22,769,896：模型总共有22,769,896个参数。这些参数是模型在训练过程中需要学习的权重和偏置项。  Trainable params: 22,769,896：可训练的参数数量与总参数数量相同，意味着模型中的所有参数都是可训练的，即在训练过程中会更新它们的值。  Non-trainable params: 0：没有不可训练的参数。不可训练的参数通常是在模型初始化时就已经确定的值，或者是在训练过程中不会改变的参数。  **2.内存使用情况：**  Input size (MB): 2.30：单个输入样本占用的内存大小约为2.30MB。这通常与输入图像的尺寸、通道数以及数据类型（如float32）有关。  Forward/backward pass size (MB): 324.29：前向传播和反向传播过程中模型占用的内存大小约为324.29MB。这包括模型参数、中间激活值、梯度等。由于反向传播需要存储前向传播中的中间结果来计算梯度，因此这部分内存需求通常比前向传播要大。  Params size (MB): 86.86：模型参数占用的内存大小约为86.86MB。这是模型所有参数（权重和偏置项）在内存中所占用的空间。  Estimated Total Size (MB): 413.45：估计的总内存需求约为413.45MB。这是输入大小、前向/反向传播大小和参数大小的总和，但可能不包括其他潜在的内存开销（如框架开销、缓存等）。  **3.字典输出：**  这个字典包含了total\_params和trainable\_params两个键，它们的值分别与之前给出的总参数数量和可训练参数数量相同。这个字典可能是由某个库（如PyTorch或TensorFlow）生成的，用于方便地访问这些统计信息。   1. **模型评估**   I.模型训练  以下是运行环境JupyterLab的简单示意：    以下是部分运行结果（运行配置没有GPU，使用的是免费使用8小时普通用户版本，运行速度极慢，在此处设置epochs为100，但是运行6小时后跑取的结果只运行至第二轮）      所以应当考虑到自行搭载环境在本地运行该程序：配置好必要环境后，**在本地运行该程序**，使用paddlepaddle-gpu库，正在运行的截图如下：  运行配置：  CUDA 11.2  CuDNN 8.6  依赖动态库:zlibwapi.dll  硬件：  Windows10  GPU: RTX3060    显卡的运行情况如下：    估算得知，运行一轮epoch的时间约为2分钟，预计3个多小时完成训练  等待该程序运行结束（epochs设置为100）  运行结束，得到的output文件如下：    以下是生成训练结果的代码：   |  | | --- | | if epoch % 20 == 0:          paddle.save(vit.state\_dict(), "./output/vit\_{}.pdparams".format(epoch))  writer.close() |   **II.模型评估：**  从终端给出的信息来看，可以从以下角度取分析本模型的情况：  训练损失（Loss）和准确率（Accuracy）：  在epoch 272的不同batch\_id中，损失（loss）值在1.9到2.2之间波动，这表示模型在训练数据上的预测误差仍然相对较高。准确率（acc）在0.28125（即28.125%）到0.4375（即43.75%）之间变化，显示出模型在单个批次上的性能不稳定，并且整体性能有待提高。  验证损失（Loss）和准确率（Accuracy）：  在验证集上，损失值（loss）分别为1.6367和1.8938，相较于训练损失略低，但仍处于较高水平。验证集的准确率在两个批次中分别为0.625（即62.5%）和0.3548（即35.48%），显示出模型在验证集上的性能也有待提高。特别是第二个批次的准确率较低，可能是模型泛化能力不足的体现。  过拟合与欠拟合：  从目前的信息来看，模型可能正处于欠拟合的状态。这是因为训练集上的准确率并不高，同时验证集上的准确率也没有达到理想的水平。  另一个可能的迹象是，训练损失和验证损失之间的差距并不是特别大，这通常也暗示着模型可能尚未充分学习数据的特征。  所以可能需要采取以下措施：  尝试增加训练周期（epochs），让模型有更多的机会学习数据的特征。  检查学习率（learning rate）设置，如果学习率过低，可能会导致模型训练缓慢；如果学习率过高，可能会导致模型在训练过程中震荡而无法收敛。  考虑使用更复杂的模型结构或调整模型的超参数，以更好地适应数据的特征。  引入数据增强（data augmentation）技术，增加训练样本的多样性和数量，有助于提高模型的泛化能力。  **六、实验结果对比及结果可视化**  （五号宋体）   1. 测试代码   以下为测试代码部分：   |  | | --- | | # 测试  # 加载模型  state\_dict = paddle.load("output/vit\_40.pdparams")  vit.set\_state\_dict(state\_dict)  vit.eval()  with paddle.no\_grad():      for batch\_id, data in enumerate(test\_loader):          img = data[0]          label = data[1]          y = vit(img)          corr = metric.compute(y, label)          metric.update(corr)          res = metric.accumulate()  print('测试集Acc:{}'.format(res)) |   运行该代码部分，得到准确率：  运行0\_epoch准确率：0.02    运行20\_epoch准确率：0.20    运行40\_epoch准确率：0.20    运行60\_epoch准确率：0.26    运行80\_epoch准确率：0.35    进行测试图片，预测结果很不一致：      准确率仍然不高，所以考虑增加epochs的轮数，再次进行训练    训练完毕，以下为训练300轮的结果：      测试样例的结果如下：    可以看到其准确度仍然很低  可以看到，即便提高epochs轮数也不能使得准确率线性提高，其准确率仍然位于20%~40%之间，即可能产生了震荡，所以考虑到了ViT本身的特性导致的限制，由于是完全初始化权重从头训练，因此测试集的精度较低，ViT比较难训练，需要在大规模数据集上预训练才可以达到较好的效果。  **附录 A主要实现代码**  **附件1 输入图片识别出对应图片类型的代码**   |  | | --- | | # 测试  # 加载预训练模型参数  state\_dict = paddle.load("output/vit\_80.pdparams")  vit.set\_state\_dict(state\_dict)  # 设置为评估模式  vit.eval()  # 预处理步骤  transform = Compose([      Resize(224),                  # 输入图片需要调整为224x224      CenterCrop(224),      ToTensor(),      Normalize(mean=[0.485, 0.456, 0.406], std=[                0.229, 0.224, 0.225]),  # ImageNet标准化  ])  # 用户输入的图片路径  img\_path = 'test2.jpg'  # 用户需要替换为实际的图片路径  # 加载图片并预处理  img = Image.open(img\_path).convert('RGB')  img\_tensor = transform(img).unsqueeze(0)  # 添加batch维度  # 预测  with paddle.no\_grad():      y = vit(img\_tensor)      probs = paddle.nn.functional.softmax(y, axis=1).numpy()      predicted\_class\_idx = np.argmax(probs)        label\_map = {          0: 'apple',          1: 'banana',          2: 'beetroot',          3: 'bell pepper',          4: 'cabbage',          5: 'capsicum',          6: 'carrot',          7: 'cauliflower',          8: 'chilli pepper',          9: 'corn',          10: 'cucumber',          11: 'eggplant',          12: 'garlic',          13: 'ginger',          14: 'grapes',          15: 'jalepeno',          16: 'kiwi',          17: 'lemon',          18: 'lettuce',          19: 'mango',          20: 'onion',          21: 'orange',          22: 'paprika',          23: 'pear',          24: 'peas',          25: 'pineapple',          26: 'pomegranate',          27: 'potato',          28: 'raddish',          29: 'soy beans',          30: 'spinach',          31: 'sweetcorn',          32: 'sweetpotato',          33: 'tomato',          34: 'turnip',          35: 'watermelon'      }  # 输出预测结果  predicted\_label = label\_map[predicted\_class\_idx]  print(f"Predicted fruit or vegetable: {predicted\_label}") | | |
|  | |