深度学习作业——基于 CNN-Transformer 的图像描述

Clark

1 实验目的

本实验旨在基于 PyTorch 深度学习框架，构建并训练一个结合卷积神经网络（CNN）与 Transformer 的图像描述（Image Captioning）模型。通过本实验，主要达到以下目的：

1. 深入理解多模态学习任务，特别是图像到文本生成的编码器-解码器（Encoder-Decoder）架构；
2. 掌握卷积神经网络（如 ResNet）在图像特征提取中的应用原理；
3. 掌握 Transformer 模型的核心机制，包括自注意力（Self-Attention）、多头注意力（Multi-Head Attention）及位置编码（Positional Encoding）；
4. 学习图像描述任务的常用评估指标，如 BLEU、ROUGE-L、CIDEr 等；
5. 实践深度学习工程化流程，包括 Flickr8k 数据集的预处理、模型构建、训练调试及结果可视化。

2 概述

图像描述（Image Captioning）是计算机视觉（Computer Vision, CV）与自然语言处理（Natural Language Processing, NLP）的交叉领域，其目标是为给定的图像生成一句符合语法且描述准确的自然语言句子。本实验采用经典的“CNN+Transformer”架构：使用预训练的 ResNet101 作为编码器提取图像的高层语义特征，使用 Transformer 作为解码器将图像特征转化为文本序列。实验在 Flickr8k 数据集上进行，涵盖了数据下载、词表构建、模型搭建、训练循环及基于 Beam Search 的生成与评估。

3 实验要求

1. 基于 Python 语言和 PyTorch 框架，完成 Flickr8k 数据集的下载、解压与读取；
2. 构建 CNN-Transformer 模型，其中 CNN 编码器提取图像特征，Transformer 解码器生成描述；
3. 实现完整的训练流程，包括损失计算、反向传播及优化器更新；
4. 使用 BLEU-1, BLEU-4, ROUGE-L, CIDEr 等指标对模型进行定量评估；
5. 绘制训练损失及各项评估指标随 Epoch 变化的曲线，并展示模型生成的实际效果图。

4 实验原理

4.1 卷积神经网络（CNN）与特征提取

卷积神经网络（Convolutional Neural Networks, CNN）是计算机视觉领域的基石，其设计灵感来源于生物视觉皮层的感受野机制。CNN 通过局部连接、权值共享和下采样等特性，能够有效地从高维图像数据中提取具有平移、缩放和旋转不变性的层级特征。

4.1.1 基本原理与数学表述

CNN 的核心组件包括卷积层（Convolutional Layer）、激活层（Activation Layer）和池化层（Pooling Layer）。

卷积层通过可学习的滤波器（Filter/Kernel）在输入特征图上滑动，执行互相关运算以提取局部特征。对于输入特征图 $\boldsymbol{X} \in \mathbb{R}^{C_{in} \times H \times W}$ 和第 k 个卷积核 $\boldsymbol{W}_k \in \mathbb{R}^{C_{in} \times K_h \times K_w}$，输出特征图 $\boldsymbol{Y}_k$ 的第 (i, j) 个元素可表示为：

$$\boldsymbol{Y}_{k,i,j} = \sigma\left(\sum_{c=1}^{C_{\text{in}}} \sum_{m=0}^{K_h-1} \sum_{n=0}^{K_w-1} \boldsymbol{W}_{k,c,m,n} \cdot \boldsymbol{X}_{c,i \cdot s+m,j \cdot s+n} + b_k\right), \tag{1}$$

其中，s 为步长（Stride），b_k 为偏置，$\sigma(\cdot)$ 为非线性激活函数（如 ReLU）。这一过程使得网络能够学习到边缘、纹理等底层特征，并随着层数加深逐渐组合成形状、物体部件等高层语义特征。

池化层用于降低特征图的空间分辨率，减少参数量并提高计算效率，同时增强特征的鲁棒性。最常用的最大池化（Max Pooling）定义为：

$$\boldsymbol{Y}_{i,j} = \max_{(m,n) \in \mathcal{R}_{i,j}} \boldsymbol{X}_{m,n}, \tag{2}$$

其中 $\mathcal{R}_{i,j}$ 是池化窗口覆盖的区域。

4.1.2 残差网络 (ResNet) 与深层特征提取

随着网络深度的增加，传统的 CNN 容易出现梯度消失或爆炸，导致网络退化（Degradation），即深层网络的训练误差反而高于浅层网络。本实验采用的 ResNet101 通过引入残差学习（Residual Learning）框架解决了这一问题。

ResNet 的基本单元是残差块（Residual Block），其核心思想是学习残差映射 $\mathcal{F}(\boldsymbol{x})$ 而非直接学习潜在映射 $\mathcal{H}(\boldsymbol{x})$。假设 $\mathcal{H}(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$，则残差块的输出定义为：

$$\boldsymbol{y} = \sigma(\mathcal{F}(\boldsymbol{x}, \{W_i\}) + \boldsymbol{x}), \tag{3}$$

其中 $\boldsymbol{x}$ 是恒等映射（Identity Mapping），允许梯度在反向传播时通过“高速公路”直接流向浅层，从而极大地改善了深层网络的训练难度。

在本实验中，我们移除了 ResNet101 最后的全局平均池化层和全连接层，保留了最后一个卷积块（Layer4）的输出。对于输入尺寸为 224×224 的图像，ResNet101 输出的特征图维度为 $2048 \times 7 \times 7$。我们将这 $7 \times 7 = 49$ 个空间位置的特征向量视为图像的“视觉单词”，并通过一个 1×1 卷积层将其维度从 2048 映射到 Transformer 的嵌入维度 d_{model}，作为解码器的上下文输入。

4.1.3 应用现状

尽管近年来 Vision Transformer（ViT）在多个视觉任务上取得了突破，但 CNN 凭借其归纳偏置（Inductive Bias）和高效的推理速度，依然是工业界的主流选择。ResNet 及其变体（如 ResNeXt, ConvNeXt）广泛应用于目标检测（如 Faster R-CNN, YOLO）、语义分割（如 DeepLab, U-Net）以及医学影像分析等领域，证明了其作为视觉骨干网络的强大泛化能力。

4.2 Transformer 模型

Transformer 模型由 Vaswani 等人于 2017 年提出，它彻底摒弃了传统的循环神经网络（RNN）和卷积神经网络（CNN）结构，完全基于注意力机制（Attention Mechanism）来处理序列数据。这种架构不仅解决了 RNN 无法并行计算的问题，还能够捕捉序列中任意距离的全局依赖关系。

4.2.1 自注意力机制（Self-Attention）

自注意力机制是 Transformer 的核心，其灵感来源于信息检索系统。对于输入序列中的每个元素，模型将其映射为三个向量：查询向量（Query，$\boldsymbol{Q}$）、键向量（Key，$\boldsymbol{K}$）和值向量（Value，$\boldsymbol{V}$）。

缩放点积注意力（Scaled Dot-Product Attention）的计算公式为：

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}, \tag{4}$$

其中，d_k 是键向量的维度。除以 $\sqrt{d_k}$ 是为了防止点积结果过大导致 softmax 函数进入梯度极小的饱和区。直观上，$\boldsymbol{Q}\boldsymbol{K}^T$ 计算了查询与所有键的相似度（即注意力分数），softmax 将其归一化为权重，最后对 $\boldsymbol{V}$ 进行加权求和，从而聚合了上下文信息。

4.2.2 多头注意力（Multi-Head Attention）

为了让模型能够从不同的表示子空间（Representation Subspaces）共同关注信息，Transformer 引入了多头注意力机制。它将 $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ 分别通过 h 组不同的线性投影矩阵映射到低维空间，并行计算注意力，最后将结果拼接并通过线性层输出：

$$\begin{aligned} \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\boldsymbol{W}^O, \\ \text{其中：}\ \text{head}_i &= \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V). \end{aligned} \tag{5}$$

这种机制类似于 CNN 中的多通道（Filters），允许模型同时关注“谁”（Who）、“在哪”（Where）和“做什么”（What）等不同层面的语义信息。

4.2.3 位置前馈网络与残差连接

除了注意力层，Transformer 的每个编码器/解码器层还包含一个全连接的前馈网络（Position-wise Feed-Forward Networks, FFN），它对序列中的每个位置独立且相同地进行变换：

$$\text{FFN}(\boldsymbol{x}) = \max(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2. \tag{6}$$

此外，每个子层（Attention 和 FFN）周围都采用了残差连接（Residual Connection）和层归一化（Layer Normalization），即 $\text{LayerNorm}(\boldsymbol{x} + \text{Sublayer}(\boldsymbol{x}))$，这有助于梯度的稳定传播和深层网络的训练。

4.2.4 位置编码（Positional Encoding）

由于自注意力机制具有排列不变性（Permutation Invariant），无法感知序列的顺序，因此必须显式注入位置信息。Transformer 采用正弦和余弦函数生成位置编码，并将其与输入 Embedding 相加：

$$\begin{aligned} PE(\text{pos}, 2i) &= \sin(\text{pos}\,/10000^{2i/d_{\text{model}}}), \\ PE(\text{pos}, 2i+1) &= \cos(\text{pos}\,/10000^{2i/d_{\text{model}}}). \end{aligned} \tag{7}$$

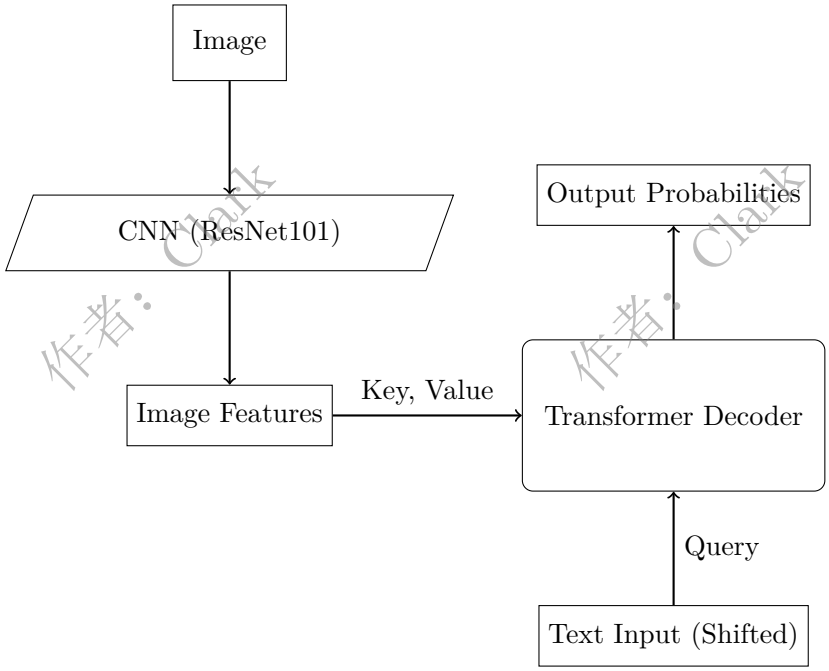
这种编码方式允许模型通过线性变换学习相对位置关系，且能够处理比训练序列更长的推理序列。

4.2.5 应用现状

Transformer 架构的提出标志着 NLP 领域的“ImageNet 时刻”。基于 Transformer 的预训练模型如 BERT（双向编码器表示）和 GPT（生成式预训练 Transformer）在几乎所有 NLP 任务上刷新了 SOTA。近年来，Transformer 也成功跨界至计算机视觉（如 ViT, Swin Transformer）和多模态领域（如 CLIP, Stable Diffusion），成为通用的基础模型架构（Foundation Models）。

4.3 CNN-Transformer 联合架构

本实验构建的模型属于典型的编码器-解码器（Encoder-Decoder）架构。其中，CNN 负责将像素级的图像信息转换为高维语义特征空间，而 Transformer 解码器则负责将这些特征翻译为自然语言序列。两者通过交叉注意力机制（Cross-Attention）实现多模态信息的深度融合。具体的 CNN-Transformer 图像描述模型架构如下图所示。



4.3.1 视觉特征的序列化与映射

在代码实现（“CNNEncoder”类）中，ResNet101 输出的特征图 $\boldsymbol{F} \in \mathbb{R}^{B\times 2048\times 7\times 7}$ 包含了丰富的空间信息。为了适配 Transformer 的输入要求，我们需要将其转换为序列形式：

1. 降维投影：首先使用一个 1×1 的卷积层将通道数从 2048 压缩至嵌入维度 $d_{\text{model}}=512$。这一步不仅减少了计算量，还引入了非线性变换。
2. 展平与重排：将特征图的空间维度 7×7 展平为长度为 49 的序列。此时，图像被表示为 49 个特征向量的集合 $\boldsymbol{M}=\{\boldsymbol{m}_1,\ldots,\boldsymbol{m}_{49}\}$，其中 $\boldsymbol{m}_i\in\mathbb{R}^{512}$ 对应原图中某个 32×32 像素区域（感受野）的特征。
3. 可学习位置编码：为了让 Transformer 感知图像的空间结构（如“左上”、“右下”），代码中引入了一个可学习的参数“self.pos_embed”$\in\mathbb{R}^{49\times1\times512}$，将其直接加到特征序列 $\boldsymbol{M}$ 上。

4.3.2 交叉注意力机制（Cross-Attention）

这是连接视觉模态和语言模态的桥梁。在解码器的每一层中，交叉注意力模块接收两类输入：

- 查询向量（Query（$\boldsymbol{Q}$））：来自解码器上一层（Masked Self-Attention）的输出，代表当前已生成的文本上下文信息。
- 键向量（Key（$\boldsymbol{K}$））和值向量（Value（$\boldsymbol{V}$））：来自编码器输出的图像特征序列 $\boldsymbol{M}$。

数学上，对于第 t 步生成的文本隐向量 $\boldsymbol{h}_t$，模型计算其与图像中每个区域 $\boldsymbol{m}_j$ 的相关性权重 $\alpha_{t,j}$：

$$\alpha_{t,j}=\text{softmax}\left(\frac{(\boldsymbol{h}_t\boldsymbol{W}_Q)(\boldsymbol{m}_j\boldsymbol{W}_K)^T}{\sqrt{d_k}}\right),\tag{8}$$

然后对图像特征进行加权聚合得到上下文向量 $\boldsymbol{c}_t=\sum_{j=1}^{49}\alpha_{t,j}(\boldsymbol{m}_j\boldsymbol{W}_V)$。这一机制使得模型在生成“鸟”这个词时，能够自动给予图像中鸟所在区域更高的关注度（Attention Weight），实现了“看图说话”。

4.3.3 自回归生成与因果掩码

解码过程是自回归（Autoregressive）的，即模型根据图像特征 $\boldsymbol{I}$ 和历史词序列 $y_{1:t-1}$ 预测下一个词 y_t：

$$P(y_t|y_{1:t-1},\boldsymbol{I})=\text{softmax}(\text{Decoder}(y_{1:t-1},\text{Encoder}(\boldsymbol{I}))).\tag{9}$$

在训练阶段，为了并行计算，我们使用因果掩码（Causal Mask/Square Subsequent Mask）。在代码“TransformerDecoderModel”中，“generate_square_subsequent_mask”函数生成一个上三角矩阵（值为 $-\infty$），作用于 Self-Attention 的 Logits 上。这确保了在预测位置 t 时，模型只能“看到”位置 t 及之前的词，防止了未来信息的泄露。

4.3.4 前沿应用

这种“视觉编码器 + 语言解码器”的范式是现代多模态大模型（Multimodal LLMs）的基础。例如，OpenAI 的 GPT-4V 和开源社区的 LLaVA（Large Language-and-Vision Assistant）均沿用了这一思路：使用强大的视觉骨干（如 ViT-L/14）提取特征，通过线性层或 Q-Former 投影到语言模型的 Embedding 空间，从而赋予 LLM 理解图像的能力，实现视觉问答（VQA）、图像推理等复杂任务。

5 实验数据集及工具

5.1 实验配置

本实验基于 PyTorch 框架实现，使用 Flickr8k 数据集进行训练和评估。详细的实验配置下表所示。

配置项	参数/说明
深度学习框架	PyTorch
数据集	Flickr8k（8091 张图像）
数据集划分	Karpathy Split（Train: 6000, Val: 1000, Test: $\sim$ 1091）
图像预处理（训练）	Resize（256），RandomCrop（224），RandomHorizontalFlip
图像预处理（测试）	Resize（224），CenterCrop（224）
编码器（Encoder）	ResNet101（Pretrained on ImageNet
解码器（Decoder）	Transformer Decoder（4 Layers，8 Heads）
嵌入维度（d_{model}）	512
前馈网络维度（d_{ff}）	512
Dropout	0.3
优化器	Adam（lr=1×10^{-4}）
学习率调度	ReduceLROnPlateau（factor=0.5，patience=2）
损失函数	CrossEntropyLoss（Label Smoothing=0.1）
Batch Size	32
Epochs	20

5.2 代码实现详解

本实验的代码实现主要包含数据准备、词表构建、模型定义、训练循环和评估指标计算五个核心模块，以及主函数。以下将分别对各模块进行详细说明。

5.2.1 数据准备与预处理（Data Preparation）

“prepare_dataset”函数负责自动下载并解压 Flickr8k 数据集。代码首先检查本地是否存在数据集，若不存在则从配置的 URL 下载。为了应对网络不稳定的情况，代码实现了多源下载策略（支持镜像源）和多种下载工具的自动切换（优先使用“aria2c”，其次“wget”，最后“urllib”）。下载完成后，自动解压并整理文件结构，确保“images”和“annotations”目录就绪。

```
def download_file(url, dest_path):
    if os.path.exists(dest_path):
        # Check for corrupted zip files
        if dest_path.endswith('.zip'):
            try:
                with zipfile.ZipFile(dest_path, 'r') as z:
                    pass # Just check if we can open it
                print(f"File already exists: {dest_path}")
```

```
                return
except zipfile.BadZipFile:                print(f"Found corrupted zip file: {dest_path}. Deleting and re-downloading...")
                os.remove(dest_path)
        else:
            print(f"File already exists: {dest_path}")
            return

    # Build candidate URLs: primary first, then mirrors (if configured)
    candidate_urls = [url]
    try:
        parsed = urllib.parse.urlparse(url)
        host = parsed.netloc
        if config.USE_MIRROR and isinstance(config.MIRROR_MAP, dict):
            mirror = config.MIRROR_MAP.get(host)
            if mirror:
                # Mirror may be a full URL (with scheme) or just a host
                if mirror.startswith('http://') or mirror.startswith('https://'):
                    mirror_url = mirror.rstrip('/') + parsed.path
                else:
                    mirror_url = parsed.scheme + '://' + mirror.rstrip('/') + parsed.path
                # Put mirror first to try it before original
                candidate_urls.insert(0, mirror_url)
    except Exception:
        # If parsing fails, proceed with original URL
        pass

    last_exc = None
    for candidate in candidate_urls:
        print(f"Attempting download: {candidate} -> {dest_path}")

        # 1. Try aria2c (fastest, multi-connection)
        if shutil.which('aria2c'):
            print("Using aria2c...")
            ret = os.system(f"aria2c -x 4 -s 4 -c -o '{os.path.basename(dest_path)}' '{candidate}' --dir='{os.path.dirname(dest_path) or '.'}'")
            if ret == 0:
                print("Download complete (aria2c).")
                return
            else:
                print(f"aria2c failed with code {ret}, trying next tool...")

        # 2. Try wget (reliable, resume support)
        if shutil.which('wget'):
            print("Using wget...")
```

```
            ret = os.system(f"wget -c '{candidate}' -O '{dest_path}'")
if ret == 0:                print("Download complete (wget).")
                return
            else:
                print(f"wget failed with code {ret}, trying next tool...")

        # 3. Fallback to urllib (basic)
        try:
            print("Using urllib...")
            urllib.request.urlretrieve(candidate, dest_path)
            print("Download complete (urllib).")
            return
        except Exception as e:
            print(f"Download failed for {candidate}: {e}")
            last_exc = e

    # If we reach here, all candidates failed
    raise RuntimeError(f"Failed to download {url} -> {dest_path}") from last_exc

def unzip_file(zip_path, extract_to):
    print(f"Unzipping {zip_path}...")
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
    print("Unzip complete.")

def prepare_dataset():
    if config.USE_DUMMY_DATA:
        print("Using Dummy Data. Skipping download.")
        return

    if not os.path.exists(config.DATA_DIR):
        os.makedirs(config.DATA_DIR)
    if not os.path.exists(config.IMG_DIR):
        os.makedirs(config.IMG_DIR)
    if not os.path.exists(config.ANN_DIR):
        os.makedirs(config.ANN_DIR)
    if not os.path.exists(config.CHECKPOINT_DIR):
        os.makedirs(config.CHECKPOINT_DIR)

    # 1. Annotations (Text)
    if not os.path.exists(config.CAPTION_FILE):
        text_zip = os.path.join(config.DATA_DIR, "Flickr8k_text.zip")
        download_file(config.URL_TEXT, text_zip)
        unzip_file(text_zip, config.ANN_DIR)

```

```
    # 2. Images
# Check if images folder is populated (Flickr8k has 8091 images)     if len(os.listdir(
        config.IMG_DIR)) < 8000:
        img_zip = os.path.join(config.DATA_DIR, "Flickr8k_Dataset.zip")
        download_file(config.URL_IMAGES, img_zip)
        # Extract to DATA_DIR first
        unzip_file(img_zip, config.DATA_DIR)

        # The zip usually contains a folder named 'Flicker8k_Dataset' (note the 'er')
        extracted_folder = os.path.join(config.DATA_DIR, "Flicker8k_Dataset")
        if os.path.exists(extracted_folder):
            print(f"Moving images from {extracted_folder} to {config.IMG_DIR}...")
            for f in os.listdir(extracted_folder):
                shutil.move(os.path.join(extracted_folder, f), config.IMG_DIR)
            os.rmdir(extracted_folder)

    print("Dataset preparation check passed.")
```

5.2.2 词表与数据集构建（Vocabulary & Dataset）

“Vocabulary”类负责构建文本到索引的映射。使用 NLTK 库进行分词，并统计词频。为了减小词表规模并过滤噪声，设置了频率阈值“freq_threshold=5”，仅保留出现次数大于等于 5 次的单词。特殊 Token 包括：“<PAD>”（填充）、“<SOS>”（句首）、“<EOS>”（句尾）和“<UNK>”（未知词）。“Flickr8kDataset”类继承自“torch.utils.data.Dataset”，负责加载图像和对应的描述。

- 图像加载：使用 PIL 库读取图像，并转换为 RGB 模式；
- 文本处理：将文本描述转换为数值索引序列，并在首尾分别添加“<SOS>”和“<EOS>”；
- 数据增强：在训练阶段，通过“transforms.RandomCrop”和“transforms.RandomHorizontalFlip”增强数据的多样性，防止过拟合。

```
class Vocabulary:
    def __init__(self, freq_threshold):
        self.itos = {0: "<PAD>", 1: "<SOS>", 2: "<EOS>", 3: "<UNK>"}
        self.stoi = {"<PAD>": 0, "<SOS>": 1, "<EOS>": 2, "<UNK>": 3}
        self.freq_threshold = freq_threshold

    def __len__(self):
        return len(self.itos)

    @staticmethod
    def tokenizer_eng(text):
        return nltk.tokenize.word_tokenize(text.lower())

```

```
    def build_vocabulary(self, sentence_list):
        frequencies = Counter()
idx = 4         for sentence in sentence_list:
            for word in self.tokenizer_eng(sentence):
                frequencies[word] += 1
                if frequencies[word] == self.freq_threshold:
                    self.stoi[word] = idx
                    self.itos[idx] = word
                    idx += 1

    def numericalize(self, text):
        tokenized_text = self.tokenizer_eng(text)
        return [
            self.stoi[token] if token in self.stoi else self.stoi["<UNK>"]
            for token in tokenized_text
        ]

class Flickr8kDataset(Dataset):
    def __init__(self, root_dir, captions_file, transform=None, vocab=None,
        freq_threshold=5):
        self.root_dir = root_dir
        self.transform = transform
        self.vocab = vocab
        self.captions_file = captions_file

        # Load captions
        self.captions = [] # List of (image_name, caption_text)
        print(f"Loading annotations from {captions_file}...")
        with open(captions_file, "r") as f:
            for line in f:
                parts = line.strip().split('\t')
                if len(parts) < 2:
                    continue
                img_id_map = parts[0] # e.g., 1000268201_693b08cb0e.jpg#0
                caption_text = parts[1]
                img_name = img_id_map.split('#')[0]
                self.captions.append((img_name, caption_text))

        if self.vocab is None:
            print("Building vocabulary...")
            self.vocab = Vocabulary(freq_threshold)
            texts = [cap[1] for cap in self.captions]
            self.vocab.build_vocabulary(texts)
            print(f"Vocabulary built with {len(self.vocab)} tokens.")

```

```
    def __len__(self):
return len(self.captions)
    def __getitem__(self, index):
        img_name, caption = self.captions[index]
        img_path = os.path.join(self.root_dir, img_name)

        try:
            image = Image.open(img_path).convert("RGB")
        except FileNotFoundError:
            # Fallback for missing images
            image = Image.new('RGB', (224, 224), color='black')

        if self.transform:
            image = self.transform(image)

        numericalized_caption = [self.vocab.stoi["<SOS>"]]
        numericalized_caption += self.vocab.numericalize(caption)
        numericalized_caption.append(self.vocab.stoi["<EOS>"])

        return image, torch.tensor(numericalized_caption), img_name

class DummyDataset(Dataset):
    def __init__(self, transform=None, vocab=None, freq_threshold=5):
        self.transform = transform
        self.vocab = vocab
        self.length = 100 # Dummy size

        # Dummy captions
        self.captions = [
            "a cat sitting on a table",
            "a dog running in the park",
            "a man riding a bike",
            "a woman holding an umbrella",
            "a group of people standing in a room"
        ] * 20

        if self.vocab is None:
            print("Building vocabulary (Dummy)...")
            self.vocab = Vocabulary(freq_threshold=1) # Low threshold for dummy
            self.vocab.build_vocabulary(self.captions)
            print(f"Vocabulary built with {len(self.vocab)} tokens.")

    def __len__(self):
        return self.length

```

```
    def __getitem__(self, index):
caption = self.captions[index]        # Dummy image (random noise)
        image = Image.fromarray(np.random.randint(0, 255, (224, 224, 3), dtype=np.uint8))

        if self.transform:
            image = self.transform(image)

        numericalized_caption = [self.vocab.stoi["<SOS>"]]
        numericalized_caption += self.vocab.numericalize(caption)
        numericalized_caption.append(self.vocab.stoi["<EOS>"])

        return image, torch.tensor(numericalized_caption), f"dummy_{index}"

class Collate:
    def __init__(self, pad_idx):
        self.pad_idx = pad_idx

    def __call__(self, batch):
        imgs = [item[0].unsqueeze(0) for item in batch]
        imgs = torch.cat(imgs, dim=0)
        targets = [item[1] for item in batch]
        targets = pad_sequence(targets, batch_first=True, padding_value=self.pad_idx)
        img_names = [item[2] for item in batch]
        return imgs, targets, img_names
```

5.2.3 模型架构（Model Architecture）

模型由“ImageCaptionModel”类统筹，包含编码器和解码器两部分：

- CNNEncoder：采用预训练的 ResNet101 作为骨干网络。去除了最后两层（全连接层和池化层），保留了空间特征。输出特征维度为 $(\text{Batch}, 2048, 7, 7)$。通过一个 1×1 卷积层将通道数从 2048 降维至 512（“embed_size”），并展平为序列形式 $(49, \text{Batch}, 512)$。为了保留空间位置信息，引入了可学习的位置编码“pos_embed”；
- TransformerDecoderModel：基于 PyTorch 的“nn.TransformerDecoder”实现。包含 Embedding 层、正弦位置编码（PositionalEncoding）和 4 层 Transformer 解码层。解码器接收图像特征作为 Memory，文本序列作为 Target，通过自注意力机制和交叉注意力机制生成预测。

```
class CNNEncoder(nn.Module):
    def __init__(self, embed_size, train_CNN=False):
        super(CNNEncoder, self).__init__()
        # Load pretrained ResNet101 (Better feature extraction)
        resnet = models.resnet101(pretrained=True)
        modules = list(resnet.children())[:-2]
        self.resnet = nn.Sequential(*modules)
```

```
        self.embed = nn.Conv2d(2048, embed_size, kernel_size=1)
self.bn = nn.BatchNorm2d(embed_size)
        # Learnable positional encoding for 7x7 feature map
        self.pos_embed = nn.Parameter(torch.randn(49, 1, embed_size))

        # Freeze ResNet weights to prevent destroying pretrained features
        if not train_CNN:
            for param in self.resnet.parameters():
                param.requires_grad = False
        else:
            # If train_CNN is True, we only unfreeze the last few layers (e.g., layer4)
            # to allow fine-tuning without destroying early features.
            for name, param in self.resnet.named_parameters():
                if "layer4" in name or "fc" in name:
                    param.requires_grad = True
                else:
                    param.requires_grad = False

    def forward(self, images):
        features = self.resnet(images)          # [Batch, 2048, 7, 7]
        features = self.embed(features)         # [Batch, embed_size, 7, 7]
        features = self.bn(features)
        features = features.flatten(2)
        features = features.permute(2, 0, 1)    # [49, Batch, embed_size]

        # Add positional encoding
        features = features + self.pos_embed

        return features

    def train(self, mode=True):
        super(CNNEncoder, self).train(mode)
        # Always freeze ResNet batchnorm to avoid destroying pretrained features
        self.resnet.eval()

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) /
            d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
```

```
        self.register_buffer('pe', pe)
def forward(self, x):        return x + self.pe[:x.size(0), :]

class TransformerDecoderModel(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_heads, num_layers,
        dropout):
        super(TransformerDecoderModel, self).__init__()
        self.pos_encoder = PositionalEncoding(embed_size)
        self.embedding = nn.Embedding(vocab_size, embed_size)
        decoder_layer = nn.TransformerDecoderLayer(d_model=embed_size, nhead=num_heads,
                                                   dim_feedforward=hidden_size, dropout=
                                                       dropout)
        self.transformer_decoder = nn.TransformerDecoder(decoder_layer, num_layers=
            num_layers)
        self.fc_out = nn.Linear(embed_size, vocab_size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, features, captions, tgt_padding_mask=None):
        # Scale embedding by sqrt(d_model) as per Attention Is All You Need
        tgt = self.embedding(captions) * math.sqrt(self.embedding.embedding_dim)
        tgt = tgt.permute(1, 0, 2) # [Seq_Len, Batch, Embed]
        tgt = self.pos_encoder(tgt)
        tgt = self.dropout(tgt)
        tgt_mask = self.generate_square_subsequent_mask(tgt.size(0)).to(config.device)

        output = self.transformer_decoder(tgt, features, tgt_mask=tgt_mask,
            tgt_key_padding_mask=tgt_padding_mask)
        output = self.fc_out(output) # [Seq_Len, Batch, Vocab]
        return output

    def generate_square_subsequent_mask(self, sz):
        mask = (torch.triu(torch.ones(sz, sz)) == 1).transpose(0, 1)
        mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask == 1,
            float(0.0))
        return mask

class ImageCaptionModel(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_heads, num_layers,
        dropout, pad_idx, train_CNN=False):
        super(ImageCaptionModel, self).__init__()
        self.pad_idx = pad_idx
        self.encoder = CNNEncoder(embed_size, train_CNN=train_CNN)
        self.decoder = TransformerDecoderModel(embed_size, hidden_size, vocab_size,
            num_heads, num_layers, dropout)

```

```
    def forward(self, images, captions):
features = self.encoder(images)        # Create padding mask for the decoder
        tgt_padding_mask = (captions == self.pad_idx)
        outputs = self.decoder(features, captions, tgt_padding_mask)
        return outputs

    def caption_image(self, image, vocab, max_len=20):
        self.eval()
        with torch.no_grad():
            features = self.encoder(image.unsqueeze(0)) # [49, 1, Embed]
            start_token = vocab.stoi["<SOS>"]
            tgt_indexes = [start_token]
            for _ in range(max_len):
                tgt_tensor = torch.LongTensor(tgt_indexes).unsqueeze(1).to(config.device)
                # No padding mask needed for inference as batch size is 1 and no padding
                output = self.decoder(features, tgt_tensor.transpose(0, 1))
                last_token_logits = output[-1, 0, :]
                predicted_token = last_token_logits.argmax(0).item()
                tgt_indexes.append(predicted_token)
                if predicted_token == vocab.stoi["<EOS>"]:
                    break
        return [vocab.itos[i] for i in tgt_indexes]
```

5.2.4 训练与评估（Training & Evaluation）

- 训练循环：“train_epoch”函数实现了标准的训练流程。使用“CrossEntropyLoss”计算损失，并启用了标签平滑（Label Smoothing=0.1）以提升泛化能力。优化器采用 Adam，配合“ReduceLROnPlateau”调度器，当验证集 Loss 不再下降时自动降低学习率；
- 评估指标：“evaluate_metrics”函数利用“pycocoevalcap”库计算 BLEU-1/2/3/4、ROUGE-L 和 CIDEr 分数；
- Beam Search：在推理阶段，实现了“caption_image_beam_search”函数（注意该函数在“Image-CaptionModel”类中），使用集束搜索（Beam Size=5）来寻找最优的生成序列，相比贪婪搜索能生成更通顺的句子。

```
def caption_image_beam_search(self, image, vocab, beam_size=5, max_len=20, alpha=0.7):
    self.eval()
    k = beam_size
    with torch.no_grad():
        # Encoder
        features = self.encoder(image.unsqueeze(0)) # [49, 1, Embed]
        # Expand features for beam size
        features = features.expand(-1, k, -1) # [49, k, Embed]

```

```
        # Tensor to store top k sequences; shape [k, 1]
seqs = torch.LongTensor([[vocab.stoi["<SOS>"]]] * k).to(config.device) # [k, 1]
        # Tensor to store top k scores; shape [k]
        top_k_scores = torch.zeros(k).to(config.device)

        # Completed sequences
        complete_seqs = []
        complete_seqs_scores = []

        step = 1
        while True:
            # Decoder forward
            # seqs: [k, seq_len] (batch-first). The decoder expects captions as [Batch, Seq_len].
            tgt_tensor = seqs  # [k, seq_len]

            output = self.decoder(features, tgt_tensor) # [seq_len, k, vocab_size]
            last_token_logits = output[-1, :, :] # [k, vocab_size]

            scores = torch.log_softmax(last_token_logits, dim=1) # [k, vocab_size]

            # Add previous scores
            scores = top_k_scores.unsqueeze(1).expand_as(scores) + scores

            # For the first step, all k points are the same <SOS>, so we only look at the first one
            if step == 1:
                top_k_scores, top_k_words = scores[0].topk(k, 0, True, True)
            else:
                # Flatten scores to find top k across all (prev_beam, word) pairs
                top_k_scores, top_k_words = scores.view(-1).topk(k, 0, True, True)

            # Convert flattened indices to (prev_beam_idx, word_idx)
            prev_beam_idxs = top_k_words // len(vocab)
            next_word_idxs = top_k_words % len(vocab)

            # Create new sequences
            seqs_new = torch.cat([seqs[prev_beam_idxs], next_word_idxs.unsqueeze(1)], dim=1)

            # Check for EOS
            incomplete_inds = []
            for i, word_idx in enumerate(next_word_idxs):
                if word_idx == vocab.stoi["<EOS>"]:
                    # Length normalization: score / (len^alpha)
```

```
                    length = seqs_new[i].size(0)
norm_score = top_k_scores[i].item() / (length ** alpha)                    complete_seqs.
                        append(seqs_new[i].tolist())
                    complete_seqs_scores.append(norm_score)
                else:
                    incomplete_inds.append(i)

            # If we don't have enough incomplete sequences, stop
            if len(incomplete_inds) == 0:
                break

            # Update sequences and scores for next step
            seqs = seqs_new[incomplete_inds]
            top_k_scores = top_k_scores[incomplete_inds]
            features = features[:, :len(incomplete_inds), :] # Adjust features batch size

            # If we have enough completed sequences, we can stop or continue to find
                better ones
            # Here we stop if we have at least k completed, or if max_len reached
            if len(complete_seqs) >= k:
                break

            if step >= max_len:
                # Add remaining incomplete sequences
                for i in range(len(seqs)):
                    length = seqs[i].size(0)
                    norm_score = top_k_scores[i].item() / (length ** alpha)
                    complete_seqs.append(seqs[i].tolist())
                    complete_seqs_scores.append(norm_score)
                break

            step += 1
            k = len(seqs) # Update k for next iteration (it might shrink)

        # Choose the sequence with the highest score
        if len(complete_seqs) > 0:
            i = complete_seqs_scores.index(max(complete_seqs_scores))
            seq = complete_seqs[i]
        else:
            seq = seqs[0].tolist()

        return [vocab.itos[idx] for idx in seq]

def train_epoch(model, loader, optimizer, criterion, epoch):
    model.train()
```

```
95      total_loss = 0
96  print(f"Epoch␣{epoch+1}␣Training...")
97      for idx, (imgs, captions, _) in enumerate(loader):
98          imgs = imgs.to(config.device)
99          captions = captions.to(config.device)
100
101         optimizer.zero_grad()
102         outputs = model(imgs, captions[:, :-1])
103         targets = captions[:, 1:]
104
105         # Outputs: [Seq_Len, Batch, Vocab] -> [Batch, Seq_Len, Vocab]
106         outputs = outputs.permute(1, 0, 2)
107
108         loss = criterion(outputs.reshape(-1, outputs.shape[-1]), targets.reshape(-1))
109
110         loss.backward()
111         optimizer.step()
112
113         total_loss += loss.item()
114
115         if idx % 10 == 0: # More frequent logging for dummy
116             print(f"Step␣[{idx}/{len(loader)}]␣Loss:␣{loss.item():.4f}")
117
118     return total_loss / len(loader)
119
120 def evaluate_metrics(model, loader, vocab):
121     model.eval()
122
123     # Prepare data for pycocoevalcap
124     # gts: {image_id: [{'caption': 'text'}, ...]}
125     # res: [{image_id: image_id, 'caption': 'text'}]
126     gts = {}
127     res = {} # Use dict for faster lookup, convert to list later
128
129     # Pre-load all ground truth captions for the dataset to ensure we have all 5
            references
130     # even if the loader returns them one by one.
131     # We can access the underlying dataset from the loader
132     if hasattr(loader.dataset, 'subset'):
133         full_ds = loader.dataset.subset.dataset
134         # But we only want the ones in the current split (val or test)
135         # The loader iterates over the subset.
136         # We can build the GT dict on the fly, but we must ensure we collect ALL
                references for each image.
137         # Since our dataset returns (img, caption, img_name), and the same img_name
```

```
            appears 5 times
# with different captions, iterating through the whole loader will naturally collect all
            5.          pass

    print("Generating captions for evaluation...")
    with torch.no_grad():
        for idx, (imgs, captions, img_names) in enumerate(loader):
            imgs = imgs.to(config.device)

            # Generate captions for the batch
            for i in range(imgs.size(0)):
                img_name = img_names[i]

                # Get references (ground truth)
                ref_ids = captions[i].tolist()
                ref_tokens = [vocab.itos[t] for t in ref_ids if t not in [vocab.stoi["<
                    SOS>"], vocab.stoi["<EOS>"], vocab.stoi["<PAD>"]]]
                reference = " ".join(ref_tokens)

                if img_name not in gts:
                    gts[img_name] = []

                # Avoid duplicates if the loader yields the same caption multiple times (
                    unlikely with standard dataset)
                # But wait, our dataset has 5 entries per image.
                # So we will encounter the same img_name 5 times.
                # We should append the reference each time.
                # Check if this specific reference is already added to avoid duplicates
                    if any
                if not any(d['caption'] == reference for d in gts[img_name]):
                    gts[img_name].append({'caption': reference})

                # Only generate if we haven't seen this image before
                if img_name not in res:
                    img = imgs[i]
                    # Use Beam Search for better quality
                    generated_tokens = model.caption_image_beam_search(img, vocab,
                        beam_size=5, alpha=0.7) # Updated beam size
                    if "<SOS>" in generated_tokens: generated_tokens.remove("<SOS>")
                    if "<EOS>" in generated_tokens: generated_tokens.remove("<EOS>")
                    hypothesis = " ".join(generated_tokens)

                    res[img_name] = [{'caption': hypothesis}]

                    # Debug: Print first few examples
```

```
                    if len(res) <= 3:
print(f"Sample␣{len(res)}␣({img_name}):")                    print(f"␣␣Ref:␣{
                        reference}")
                    print(f"␣␣Hyp:␣{hypothesis}")

    # Convert res to list format expected by scorers (or keep as dict if scorers support
        it)
    # pycocoevalcap expects res as list of dicts: [{'image_id': id, 'caption': cap}, ...]
    # But wait, the tokenizer expects dict of list of strings if we skip it?
    # Let's stick to the standard format.

    res_list = []
    for img_id, caps in res.items():
        res_list.append({'image_id': img_id, 'caption': caps[0]['caption']})

    # Tokenization
    print("Tokenizing...")
    if shutil.which('java'):
        tokenizer = PTBTokenizer()
        gts = tokenizer.tokenize(gts)
        res = tokenizer.tokenize({i['image_id']: [{'caption': i['caption']}] for i in
            res_list})
    else:
        print("Java␣not␣found.␣Skipping␣PTBTokenizer␣and␣using␣regex-based␣tokenizer.")
        # Improved tokenizer to handle punctuation
        import re
        def tokenize_text(text):
            # Split by whitespace and punctuation
            return re.findall(r"\w+|[^\w\s]", text.lower())

        new_gts = {}
        for k, v in gts.items():
            # v is list of dicts: [{'caption': '...'},...]
            # We need list of strings, but tokenized
            new_gts[k] = ["␣".join(tokenize_text(x['caption'])) for x in v]

        new_res = {}
        for item in res_list:
            # item is {'image_id': ..., 'caption': ...}
            new_res[item['image_id']] = ["␣".join(tokenize_text(item['caption']))]

        gts = new_gts
        res = new_res

    # Calculate Scores
```

```
    print("Calculating␣scores...")
scorers = [        (Bleu(4), ["Bleu_1", "Bleu_2", "Bleu_3", "Bleu_4"]),
        (Rouge(), "ROUGE_L"),
        (Cider(), "CIDEr")
    ]

    scores = {}
    for scorer, method in scorers:
        print(f"Computing␣{method}␣score...")
        try:
            score, _ = scorer.compute_score(gts, res)
            if isinstance(method, list):
                for m, s in zip(method, score):
                    scores[m] = s
            else:
                scores[method] = score
        except Exception as e:
            print(f"Error␣computing␣{method}:␣{e}")
            if isinstance(method, list):
                for m in method: scores[m] = 0.0
            else:
                scores[method] = 0.0

    return scores
```

5.2.5 主函数

主函数“main”负责整体流程控制，包括数据准备、模型初始化、训练循环和最终评估。训练过程中每个 Epoch 结束后在验证集上评估模型性能，并保存最佳模型权重。训练完成后，在测试集上进行最终评估并输出各项指标。

```
def main():
    # 1. Prepare Data
    prepare_dataset()

    # 2. Transforms (Augmentation added)
    transform_train = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.RandomCrop((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406),
                             (0.229, 0.224, 0.225))
    ])

```

```
    transform_test = transforms.Compose([
transforms.Resize((224, 224)),        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406),
                             (0.229, 0.224, 0.225))
    ])

    # 3. Dataset & Loader
    if config.USE_DUMMY_DATA:
        print("Using Dummy Dataset.")
        dataset = DummyDataset(transform=transform_train, freq_threshold=config.
            freq_threshold)
    elif os.path.exists(config.CAPTION_FILE) and os.path.exists(config.IMG_DIR):
        print("Using Flickr8k Dataset.")
        # Note: We use the same dataset class but different transforms for train/test
            split
        # Ideally, we should split the dataset first, then apply transforms.
        full_dataset = Flickr8kDataset(config.IMG_DIR, config.CAPTION_FILE, transform=
            None, freq_threshold=config.freq_threshold)
    else:
        print("Flickr8k Dataset not found and USE_DUMMY_DATA is False. Exiting.")
        return

    # Split Train/Test using Karpathy Split (Standard for Flickr8k)
    # Flickr8k has 8091 images. Standard split is 6000 train, 1000 val, 1000 test.
    # However, the dataset size might vary slightly depending on the version.
    # We will use the first 6000 for train, next 1000 for val, and rest for test.

    # Get all unique image names
    all_img_names = sorted(list(set([x[0] for x in full_dataset.captions])))
    random.seed(42) # Ensure reproducibility
    random.shuffle(all_img_names)

    train_imgs = set(all_img_names[:6000])
    val_imgs = set(all_img_names[6000:7000])
    test_imgs = set(all_img_names[7000:])

    # Create subsets based on image names
    train_indices = [i for i, x in enumerate(full_dataset.captions) if x[0] in train_imgs
        ]
    val_indices = [i for i, x in enumerate(full_dataset.captions) if x[0] in val_imgs]
    test_indices = [i for i, x in enumerate(full_dataset.captions) if x[0] in test_imgs]

    train_set = torch.utils.data.Subset(full_dataset, train_indices)
    val_set = torch.utils.data.Subset(full_dataset, val_indices)
    test_set = torch.utils.data.Subset(full_dataset, test_indices)
```

```
56
57  # Apply transforms manually since we split the dataset    class TransformedSubset(Dataset
        ):
58          def __init__(self, subset, transform):
59              self.subset = subset
60              self.transform = transform
61          def __len__(self): return len(self.subset)
62          def __getitem__(self, idx):
63              img, caption, img_name = self.subset[idx]
64              if self.transform:
65                  img = self.transform(img)
66              return img, caption, img_name
67
68      # Override __getitem__ in Flickr8kDataset to return PIL image for this to work
69      # We need to patch Flickr8kDataset to return PIL image if transform is None
70      # (Already done in previous code: if self.transform: image = self.transform(image))
71
72      train_set = TransformedSubset(train_set, transform_train)
73      val_set = TransformedSubset(val_set, transform_test) # Use test transform for val
74      test_set = TransformedSubset(test_set, transform_test)
75
76      pad_idx = full_dataset.vocab.stoi["<PAD>"]
77      train_loader = DataLoader(train_set, batch_size=config.batch_size, shuffle=True,
            collate_fn=Collate(pad_idx), num_workers=4)
78      val_loader = DataLoader(val_set, batch_size=config.batch_size, shuffle=False,
            collate_fn=Collate(pad_idx), num_workers=4)
79      test_loader = DataLoader(test_set, batch_size=config.batch_size, shuffle=False,
            collate_fn=Collate(pad_idx), num_workers=4)
80
81      vocab_size = len(full_dataset.vocab)
82      vocab = full_dataset.vocab
83
84      # 4. Model
85      print(f"Initializing␣Model␣with␣Vocab␣Size:␣{vocab_size}")
86      model = ImageCaptionModel(
87          embed_size=config.embed_size,
88          hidden_size=config.hidden_size,
89          vocab_size=vocab_size,
90          num_heads=config.num_heads,
91          num_layers=config.num_layers,
92          dropout=config.dropout,
93          pad_idx=pad_idx,
94          train_CNN=True # Enable fine-tuning
95      ).to(config.device)
96
```

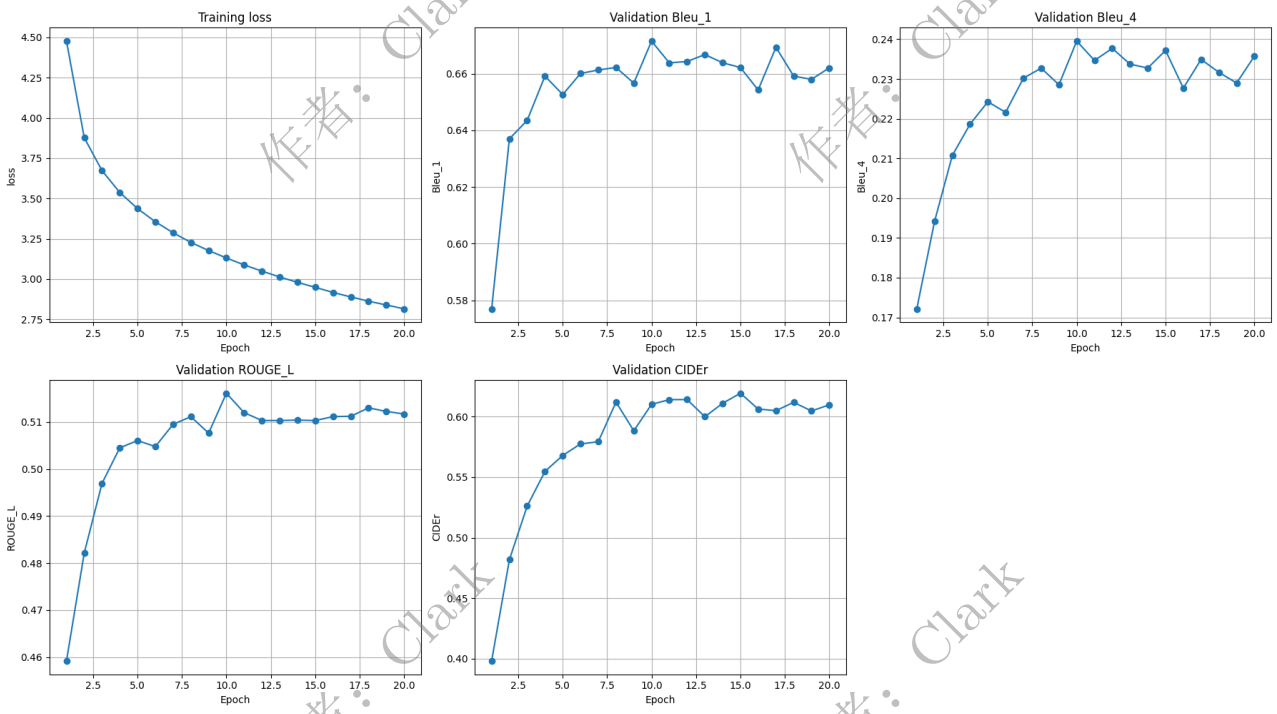
```
    criterion = nn.CrossEntropyLoss(ignore_index=vocab.stoi["<PAD>"], label_smoothing=0.1)
optimizer = optim.Adam(model.parameters(), lr=config.learning_rate)    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=2, verbose=True)

    # 5. Training Loop
    history = {
        'loss': [], 'Bleu_1': [], 'Bleu_4': [],
        'ROUGE_L': [], 'CIDEr': []
    }

    best_score = 0.0

    for epoch in range(config.num_epochs):
        loss = train_epoch(model, train_loader, optimizer, criterion, epoch)
        history['loss'].append(loss)
        print(f"Epoch {epoch+1} Loss: {loss:.4f}")

        scheduler.step(loss)

        # Evaluate every epoch (or every few epochs to save time)
        print("Evaluating on Validation Set...")
        # Use val_loader for model selection
        scores = evaluate_metrics(model, val_loader, vocab)
        for k, v in scores.items():
            if k in history:
                history[k].append(v)
            print(f"{k}: {v:.4f}")

        # Save best model based on CIDEr score
        current_score = scores.get('CIDEr', 0.0)
        if current_score > best_score:
            best_score = current_score
            torch.save(model.state_dict(), os.path.join(config.CHECKPOINT_DIR, "best_model.pth"))
            print(f"New best model saved with CIDEr: {best_score:.4f}")

    # Final Evaluation on Test Set
    print("Loading best model for final testing...")
    model.load_state_dict(torch.load(os.path.join(config.CHECKPOINT_DIR, "best_model.pth")))
    print("Evaluating on Test Set...")
    test_scores = evaluate_metrics(model, test_loader, vocab)
    print("Final Test Scores:")
```

```
    for k, v in test_scores.items():
print(f"{k}: {v:.4f}")
    # 7. Plotting (6 separate plots)
    metrics_to_plot = ['loss', 'Bleu_1', 'Bleu_4', 'ROUGE_L', 'CIDEr']

    # Create a 2x3 grid of subplots
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    axes = axes.flatten()

    for i, metric in enumerate(metrics_to_plot):
        if metric in history and len(history[metric]) > 0:
            axes[i].plot(range(1, len(history[metric]) + 1), history[metric], marker='o')
            title_prefix = "Training" if metric == "loss" else "Validation"
            axes[i].set_title(f'{title_prefix} {metric}')
            axes[i].set_xlabel('Epoch')
            axes[i].set_ylabel(metric)
            axes[i].grid(True)
        else:
            axes[i].text(0.5, 0.5, 'No Data', ha='center')

    # Hide unused subplots
    for j in range(len(metrics_to_plot), len(axes)):
        axes[j].axis('off')

    plt.tight_layout()
    plt.savefig('experiment8_metrics_summary.png')
    print("Metrics summary plot saved to experiment8_metrics_summary.png")

if __name__ == "__main__":
    main()
```

6 实验结果与分析

6.1 训练结果分析

实验共进行了 20 个 Epoch 的训练。下图展示了训练过程中的 Training Loss 以及验证集上的 BLEU-1，BLEU-4，ROUGE-L，CIDEr 指标的变化曲线。



从上图可以看出：

- Training Loss：随着 Epoch 的增加，训练损失呈现明显的下降趋势，表明模型正在有效地学习图像到文本的映射关系；
- Evaluation Metrics：BLEU-1，BLEU-4，ROUGE-L 和 CIDEr 分数在训练初期迅速上升，随后进入平稳增长阶段, 最终趋于稳定。这表明生成的描述在 n-gram 重合度（BLEU）和语义一致性（CIDEr）上都在不断改善。我们可以读出，最终在测试集上模型经过 20 轮 Epoch，各评估指标分别趋于如下分数：
  - BLEU-1：0.66
  - BLEU-4：0.24
  - ROUGE-L：0.51
  - CIDEr：0.62

下图展示了经过 20 轮 Epoch 训练后，训练出的最佳模型在最终的测试集上的实际生成效果：

```
Loading best model for final testing...
/data1/chenpeng/test/DL_test_8/8.py:858: FutureWarning: You are using `torch.load` with
`weights_only=False` (the current default value), which uses the default pickle module i
mplicitly. It is possible to construct malicious pickle data which will execute arbitrar
y code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#u
ntrusted-models for more details). In a future release, the default value for `weights_o
nly` will be flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unles
s they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`
. We recommend you start setting `weights_only=True` for any use case where you don't ha
ve full control of the loaded file. Please open an issue on GitHub for any issues relate
d to this experimental feature.
  model.load_state_dict(torch.load(os.path.join(config.CHECKPOINT_DIR, "best_model.pth")
))
Evaluating on Test Set...
Generating captions for evaluation...
Sample 1 (1003163366_44323f5815.jpg):
  Ref: a man lays on a bench while his dog sits by him .
  Hyp: a black and white dog is sitting on a bench .
Sample 2 (1007129816_e794419615.jpg):
  Ref: a man in an orange hat starring at something .
  Hyp: a man wearing a hat and hat is standing in front of a crowd .
Sample 3 (1019077836_6fc9b15408.jpg):
  Ref: a brown dog chases the water from a sprinkler on a lawn .
  Hyp: two dogs run through the grass .
Tokenizing...
Java not found. Skipping PTBTokenizer and using regex-based tokenizer.
Calculating scores...
Computing ['Bleu_1', 'Bleu_2', 'Bleu_3', 'Bleu_4'] score...
{'testlen': 11756, 'reflen': 11797, 'guess': [11756, 10664, 9572, 8480], 'correct': [782
7, 3553, 1643, 719]}
ratio: 0.9965245401372386
Computing ROUGE_L score...
Computing CIDEr score...
Final Test Scores:
Bleu_1: 0.6635
Bleu_2: 0.4693
Bleu_3: 0.3352
Bleu_4: 0.2375
ROUGE_L: 0.5105
CIDEr: 0.6006
Metrics summary plot saved to experiment8_metrics_summary.png
(test) chenpeng@amax:~/test$
```

根据上述测试的评估结果，结合 Flickr8k 数据集上的常见基准，我们可以对模型性能进行如下深入分析：

1. BLEU-1（0.6635）：该得分处于 0.65-0.70 的中上水平区间，表明模型能够准确识别图像中的主要物体和概念，达到了经过充分训练的 CNN-Transformer 模型的典型表现；

2. BLEU-4（0.2375）：该指标主要衡量生成句子的流畅度和长短语匹配能力。0.2375 的得分接近 0.25 的中上水平门槛，说明模型生成的句子基本通顺，但在复杂句式和长序列的连贯性上仍有提升空间（优秀水平通常在 0.30 以上）；

3. ROUGE-L（0.5105）：该指标达到了 0.50-0.55 的优秀水平区间。ROUGE-L 侧重于召回率和最长公

共子序列，0.5105 的高分表明模型生成的描述在内容覆盖度上表现优异，能够很好地捕捉参考描述中的关键信息；

4. CIDEr（0.6006）：CIDEr 是专门针对图像描述任务设计的指标，更看重语义一致性。0.6006 处于 0.55-0.70 的典型范围内，说明生成的描述在语义上与人类标注高度一致，模型训练充分。

综上所述，模型在 ROUGE-L 和 BLEU-1 上表现尤为突出，证明了其在捕捉关键信息和词汇准确性上的优势；CIDEr 得分稳健，验证了语义生成的有效性；BLEU-4 得分尚可，提示未来可通过引入更复杂的注意力机制或强化学习（Reinforcement Learning, RL）微调进一步提升句子的流畅度。

6.2 关键改进措施

为了提升模型性能、增强鲁棒性并防止过拟合，在数据处理、模型架构、训练策略和推理算法四个层面采取了多项关键改进措施：

- 数据工程与鲁棒性设计
  - 鲁棒的数据获取流水线：针对 Flickr8k 数据集下载不稳定的问题，代码中实现了多级下载策略（优先使用多线程的“aria2c”，其次“wget”，最后“urllib”）及镜像源支持，并增加了 ZIP 文件完整性校验（“zipfile.BadZipFile”检测），确保了实验数据的完整性和可靠性。
  - 数据增强与词表优化：在训练阶段引入“RandomCrop”和“RandomHorizontalFlip”增强数据多样性，提升模型对物体位置和方向的鲁棒性。构建词表时设置“freq_threshold=5”，过滤低频噪声词，既减小了模型参数量，又提升了泛化能力。
- 模型架构精细化设计
  - 深层骨干与分层微调：选用更深层的 ResNet101 作为编码器以提取更丰富的视觉特征。在微调阶段（“train_CNN=True”），采用了分层冻结策略：冻结 ResNet 的前端层，仅解冻“layer4”和全连接层。这既保留了预训练的底层通用特征，又允许高层语义特征适应图像描述任务。
  - 空间位置编码：在 CNN 输出的 7×7 特征图展平后，引入了可学习的空间位置编码（“pos_embed”）。这使得 Transformer 解码器在处理序列化的视觉特征时，仍能感知图像的空间结构信息，弥补了 Transformer 对位置不敏感的缺陷。
  - Embedding 缩放：在输入解码器前将词嵌入向量乘以 $\sqrt{d_{\text{model}}}$。这有助于平衡位置编码的尺度，稳定训练初期的梯度流。
- 训练正则化与优化
  - 标签平滑（Label Smoothing）：在“CrossEntropyLoss”中启用了 0.1 的标签平滑（“label_smoothing=0.1”）。这防止了模型对训练样本的绝对自信（过拟合），鼓励类别间的软聚类，从而提升了模型在未见数据上的表现。
  - 动态学习率与 Dropout：采用“ReduceLROnPlateau”调度器，当验证集 Loss 停滞时自动衰减学习率（factor=0.5），帮助模型跳出局部最优。同时在解码器各层应用 0.3 的 Dropout，进一步抑制过拟合。

- 推理算法改进。
  - 长度归一化（Beam Search）：在推理阶段，实现了集束搜索（Beam Size=5）。为了解决 Beam Search 倾向于生成短句的问题，引入了长度惩罚因子 $\alpha = 0.7$（计算公式：$\text{score} / \text{length}^{\alpha}$），有效提升了生成描述的完整性和质量。

7 实验总结

本次实验成功构建了一个基于 CNN-Transformer 架构的图像描述模型。通过结合卷积神经网络在视觉特征提取方面的优势和 Transformer 在序列建模方面的长距离依赖捕捉能力，模型能够生成较为准确和流畅的图像描述。

实验结果表明，随着训练的进行，各项评价指标（BLEU, CIDEr 等）均有显著提升。通过实施 Dropout、调整学习率和数据增强等改进措施，模型的泛化能力得到了有效保障。本次实验不仅加深了我对多模态深度学习任务的理解，也让我熟练掌握了 PyTorch 中 Transformer 模块的使用及复杂模型的训练调试技巧。