实验 8：基于 CNN-Transformer 的图像描述实验指导书

一、实验目的

1. 理解图像描述生成任务的基本概念和挑战。
2. 熟悉卷积神经网络（CNN）和 Transformer 模型的原理和应用。
3. 学习如何将 CNN 和 Transformer 模型相结合，以提取图像特征并生成与图像内容相关的自然语言描述。
4. 掌握数据预处理技术，包括图像处理和文本处理，以准备适合模型输入的数据集。
5. 学习如何构建、训练和评估基于 CNN-Transformers 的图像描述生成模型。

二、实验要求

1. 安装深度学习框架 PyTorch 或 TensorFlow，配置和搭建 CNN 和 Transformer 模型所需的神经网络结构。构建 CNN 模型作为图像特征提取器，并加载预训练的权重。构建 Transformer 模型作为图像描述生成器，并将 CNN 模型的输出作为输入（或其他）。
2. 使用 MSCOCO 或其他合适的图像描述数据集，确保图像和描述文本之间具有对应关系。同时完成数据预处理，包括使用图像处理库对图像进行预处理，例如调整大小、裁剪或填充图像；对描述文本进行文本处理，例如分词、去除停用词、构建词汇表等。数据集应包含训练集和测试集，并按照一定比例进行划分。
3. 提供评估指标的数值，包括 BLEU-1、BLEU-4、METEOR、ROUGE-L 和 CIDEr 等，本实验对指标数值不做要求。
4. 如果选择做此实验作业，按规定时间在课程网站提交实验报告、代码以及 PPT。

三、实验原理

1. 卷积神经网络：卷积神经网络是一种广泛应用于计算机视觉任务的深度学习模型。它通过一系列卷积层和池化层，有效地提取图像的局部特征和全局特征。在图像描述生成任务中，我们可以使用预训练的 CNN 模型，如 VGGNet、ResNet 等，作为图像特征提取器。通过将图像输入 CNN 模型，可以获得图像的高级特征表示。如图 1 所示。

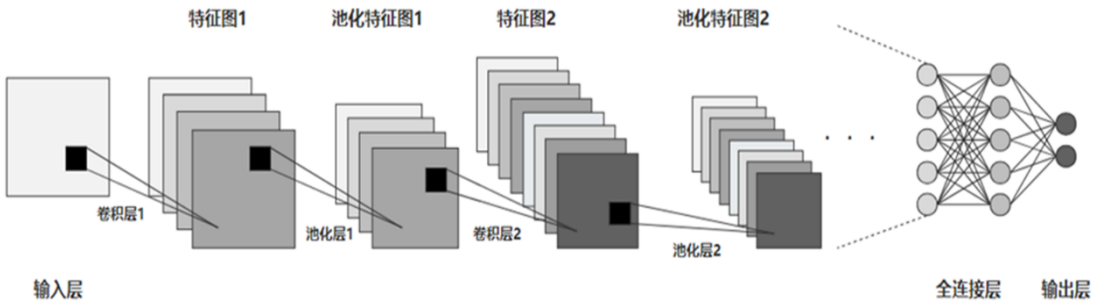

图 1　CNN 大致结构

2. Transformer 模型：Transformer 模型是一种基于自注意力机制的序列建模模型，最初提出用于机器翻译任务。它的核心是自注意力机制，能够在序列中建立全局的依赖关系。Transformer 模型由多个编码器层和解码器层组成，其中编码器负责将输入序列编码为上下文感知的表示，解码器则根据编码器的输出和先前生成的标记来生成下一个标记。在图像描述生成中，我们可以将 Transformer 模型应用于从图像特征到描述文本的映射。如图 2 所示。

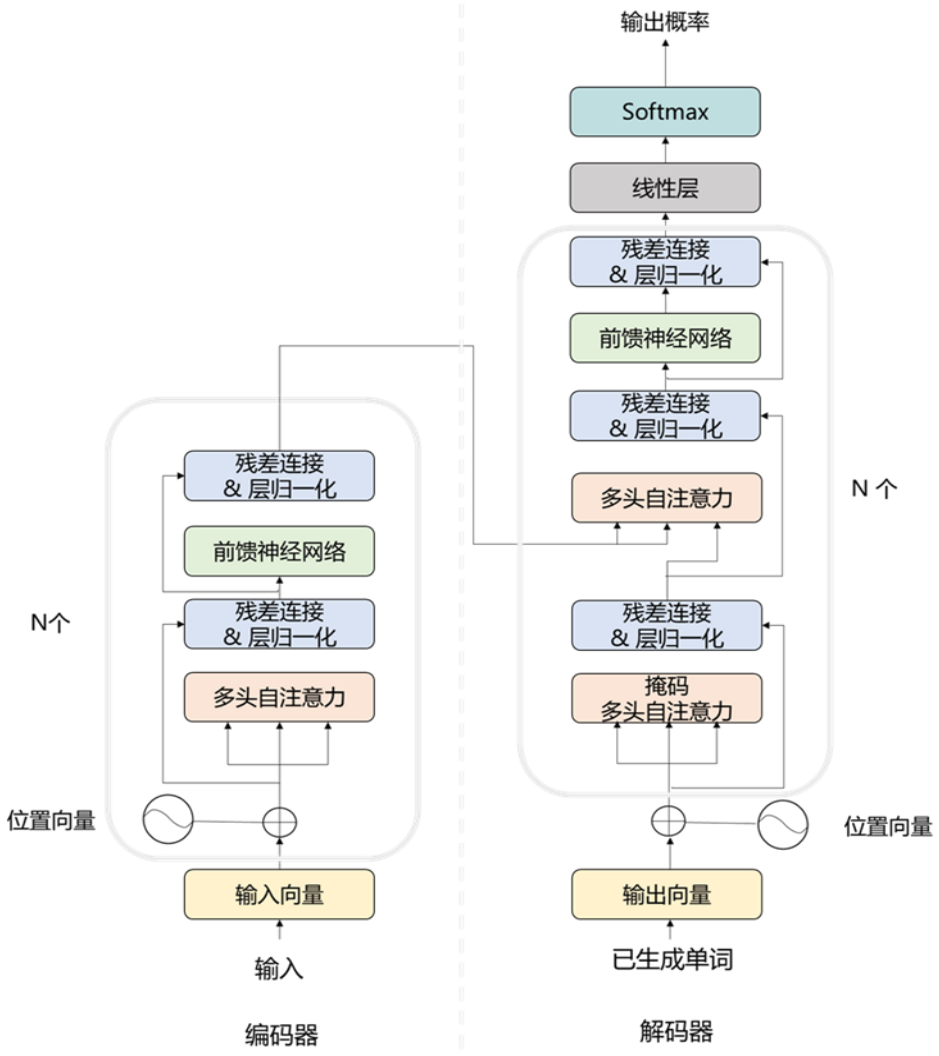

图 2 Transformers 结构

3. CNN-Transformers 模型：CNN-Transformers 模型将 CNN 和 Transformer 模型结合起来，用于图像描述生成任务。首先，通过 CNN 模型对输入图像进行特征提取，得到图像的高级特征表示。然后，将这些特征表示输入到 Transformer 模型中，通过自注意力机制和逐层处理，将图像特征映射到对应的描述文本；最终，模型根据图像内容生成自然语言描述。如图 3 所示。

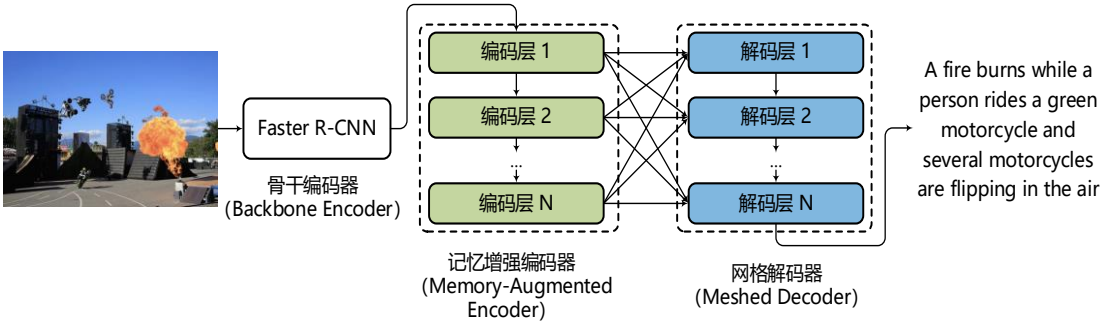

图 3：CNN-Transformers 模型结构（M2Transformer）

4. 在实验中，需要对图像和描述文本进行预处理。对于图像，常见的预处理操作包括调整大小、裁剪或填充图像，以确保输入 CNN 模型的图像具有一致的大小。对于描述文本，可以进行分词、去除停用词、构建词汇表等操作，以便模型能够理解和处理文本数据。

四、实验所用工具及数据集

1. 深度学习框架：
   (1) PyTorch：PyTorch 是一个广泛应用于深度学习的开源框架，提供了灵活的张量操作和自动求导功能，适合构建和训练深度神经网络模型。
   (2) TensorFlow：TensorFlow 是另一个广泛使用的深度学习框架，提供了强大的张量计算和高级建模工具，支持分布式训练和部署。
2. 数据集（以 MSCOCO 为例）：MSCOCO（Microsoft Common Objects in Context）是一个广泛使用的图像描述数据集，由微软公司创建和维护。该数据集旨在为图像理解和图像生成任务提供丰富的图像和对应的描述文本。MSCOCO 数据集的主要特点如下：
   (1) 数据规模：MSCOCO 数据集是一个大规模的数据集，包含超过 120,000 张图像。这些图像涵盖了多个场景和对象类别，包括人物、动物、自然风景、日常物品等。
   (2) 图像和文本对应：每张图像都有多个人工描述文本，每个文本描述约有 5 至 40 个单词。这些描述旨在准确而详细地描述图像中的内容，提供了对图像语义的丰富理解。
   (3) 多样性和复杂性：MSCOCO 数据集中的图像涵盖了多样化和复杂化的场景，

包括多个对象的场景、遮挡、不同视角和光照条件等。这使得该数据集对于训练和评估复杂的图像理解和生成模型很有挑战性。

(4) 标注质量：MSCOCO 数据集的图像和描述文本都经过了仔细的人工标注，以确保数据的质量和准确性。每个描述都是由多个标注者独立编写的，以提供多样化和客观性。

图像结构：MSCOCO 数据集由一组图像组成，每个图像都有一个唯一的标识符（ID）。图像存储在一个层次目录结构中，每个图像文件命名为 <image_id>.jpg。例如，ID 为 000000000001 的图像将被存储为 000000000001.jpg。

注释结构：每个图像的注释存储在一个 JSON 文件中，该文件包含一个字典，具有以下键：

(1) info：数据集的元数据，例如版本和贡献者。

(2) licenses：与图像相关的许可证列表。

(3) images：图像注释列表，其中每个注释是一个字典，具有以下键：

(4) id：图像的唯一 ID。

(5) width 和 height：图像的尺寸。

(6) file_name：图像的文件名。

(7) license：图像的许可证。

(8) flickr_url：图像在 Flickr 上的 URL（如果适用）。

(9) coco_url：图像在 MSCOCO 网站上的 URL。

annotations：注释对象列表，其中每个对象是一个字典，具有以下键：

(1) id：注释的唯一 ID。

(2) image_id：注释所属的图像 ID。

(3) category_id：对象类别的 ID（例如人、车、狗等）。

(4) bbox：对象的边界框坐标（x,y,w,h）。

(5) area：对象的面积，以像素为单位。

(6) segmentation：定义对象分割掩码的多边形坐标列表。

(7) iscrowd：一个标志，指示对象是否是一个 crowd（即一组对象）。

categories：类别对象列表，其中每个对象是一个字典，具有以下键：

(1) id：类别的唯一 ID。

(2) name：类别的名称（例如人、车、狗等）。

(3) supercategory：类别的超类别（例如动物、车辆等）。

MSCOCO 提供了多种类型的注释，包括：

(1) 对象检测：对象的边界框注释。

(2) 对象分割：对象分割掩码的多边形注释。

(3) 图像字幕：图像的自然语言字幕。

(4) 视觉问答：关于图像的问题和答案。

数据集划分：

(1) MSCOCO 数据集通常被划分为三个子集：

(2) 训练集（train2014）：82,783 张图像用于训练。

(3) 验证集（val2014）：40,504 张图像用于验证。

(4) 测试集（test2014）：40,775 张图像用于测试。

五、实验步骤与方法

1. 数据准备

(1) 下载 MSCOCO 数据集并提取图像和注释。

(2) 对图像进行预处理，例如将其 resize 到固定大小（例如 224x224）并 normalize 像素值。样例代码如下：

```
import cv2
import numpy as np
# 加载原始图像
image = cv2.imread('input.jpg')
# 调整大小
resized_image = cv2.resize(image, (224, 224))
# 归一化像素值
normalized_image = resized_image.astype(np.float32) / 255.0
# 打印调整大小后图像的形状和像素值范围
print("Resized Image Shape:", resized_image.shape)
print("Normalized Image Range:", np.min(normalized_image), "-",
np.max(normalized_image))
# 可选择保存调整大小后的图像
cv2.imwrite('resized_image.jpg', resized_image)
# 可选择显示调整大小后的图像
cv2.imshow('Resized Image', resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

(3) 将字幕分词并创建词汇表。（可以加载 Transformers 库的 Autotokenizer 类，通过指定分词器来实现自动分词，样例代码如下：）

```
from transformers import BertTokenizer
# 加载 BERT 分词器
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# 假设字幕文本已经加载到 subtitles 变量中
# 分词
tokens = tokenizer.tokenize(subtitles)
```

```
# 打印分词结果
for token in tokens:
    print(token)
```

(4) 将数据集分割成训练集、验证集和测试集（例如 80%用于训练，10%用于验证，10%用于测试）。

2. 模型架构

(1) 实现一个基于 CNN-Transformer 的模型，该模型由以下部分组成：

(2) 一个卷积神经网络（CNN）用于从图像中提取视觉特征。

(3) 一个 transformer 编码器用于处理视觉特征并生成一系列文本嵌入向量。

(4) 一个 transformer 解码器用于根据词嵌入生成字幕。

(5) CNN 可以是一个预训练的模型，例如 ResNet 或 VGG，而 transformer 可以是一个原始 transformer 架构的变体。

样例模型如下：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import resnet50
from torch.nn import TransformerEncoder, TransformerEncoderLayer
class CNNTransformerModel(nn.Module):
    def __init__(self, cnn_model, emb_dim, num_heads, num_layers):
        super(CNNTransformerModel, self).__init__()
        self.cnn = cnn_model
        self.feature_dim = cnn_model.fc.in_features
        self.positional_encoding = PositionalEncoding(emb_dim)
        encoder_layer = TransformerEncoderLayer(d_model=emb_dim,
nhead=num_heads)
        self.transformer_encoder = TransformerEncoder(encoder_layer,
num_layers=num_layers)
        self.fc = nn.Linear(emb_dim, vocab_size)  # vocab_size 为目标语言的词汇表大小

    def forward(self, images, captions):
        features = self.cnn(images)
        features = features.permute(0, 2, 3, 1)  # 调整维度顺序以适应 Transformer
        features = features.view(features.size(0), -1, self.feature_dim)  # 展平特征张量
        features = self.positional_encoding(features)
        features = self.transformer_encoder(features)
        output = self.fc(features)
        return output
```

```
class PositionalEncoding(nn.Module):
    def __init__(self, emb_dim, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=0.1)
        pe = torch.zeros(max_len, emb_dim)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, emb_dim, 2).float() * (-
math.log(10000.0) / emb_dim))
        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):
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)
```

3. 模型训练

(1) 使用训练集训练模型，目标函数包括：预测字幕和 ground-truth 字幕之间的交叉熵损失。可选：额外的损失函数，例如注意力正则化或基于强化学习的损失函数。

(2) 使用合适的优化器（例如 ADAM）和学习率调度。

(3) 训练模型直到收敛，例如 10-20 个 epoch（自定）。

4. 模型评估

使用验证集评估模型，评估指标包括（可选）：

(1) BLEU 评分（例如 BLEU-1、BLEU-4）

(2) METEOR 评分

(3) ROUGE 评分

(4) CIDEr 评分

根据验证性能调整超参数（例如学习率、批量大小、epoch 数）。

5. 模型测试

(1) 使用测试集评估模型，评估指标与步骤 4 相同。

(2) 将模型的性能与 MSCOCO 数据集上的基线模型进行比较。

6. 可视化和分析

(1) 可视化注意力权重和生成的字幕，更进一步了解模型的推理过程。

(2) 分析模型在不同类型的图像（例如对象、场景、动作）和字幕（例如短、长、描述性）上的性能。