视频描述报告

背景介绍

视频描述任务（Video Captioning Task）是一个跨模态的研究领域，旨在为给定的视频生成自然语言描述。该任务结合了计算机视觉和自然语言处理技术，具有广泛的应用场景，例如视频内容检索、辅助盲人导航、视频摘要生成以及人机交互。

任务主要内容

1. **输入**
   一个视频，通常包含一系列连续的图像帧以及可能的音频信号。
2. **输出**
   一段自然语言描述，准确、连贯地表达视频的内容，包括场景、动作、物体、人物及其关系等。
3. **挑战**

- **视频内容理解**：需要识别视频中的物体、场景、动作和复杂事件。
- **时序信息建模**：视频是动态数据，需要捕捉事件的时序关系。
- **语言生成**：生成的描述需要语法正确且语义合理。
- **多模态信息融合**：需要将视觉和听觉信号与语言生成进行高效结合。

技术背景

- **计算机视觉的发展**
  - 随着深度学习的兴起，卷积神经网络（CNN）等技术在图像分类、目标检测和动作识别任务中取得了显著的进展。这些技术为视频描述提供了视觉特征提取的基础。
  - 视频作为动态序列数据，需要建模时间信息。3D-CNN、LSTM、Transformer等时间序列模型应运而生，为理解视频内容提供了有效方法。
- **自然语言处理的进步**
  - RNN（如LSTM、GRU）以及基于自注意力机制的Transformer（如BERT、GPT）显著提高了语言生成的能力。
  - 视频描述任务需要将视频的特征转化为自然语言序列，这种跨模态生成任务依赖于NLP中语义建模和句子生成的技术进步。

- **多模态学习**
  - 视频描述是典型的多模态任务，需要将视觉、听觉甚至文本信号融合在一起。这促进了多模态深度学习的研究，例如联合嵌入空间、注意力机制和多模态预训练模型的开发。

应用背景

- **信息爆炸与需求增长**
  - 随着社交媒体和流媒体平台（如YouTube、TikTok）的视频内容急剧增长，手动标注或分类视频内容的效率低下。自动化的视频描述技术可以帮助快速生成视频元数据，便于检索、推荐和管理。
- **无障碍技术**
  - 对于视觉障碍者，视频描述可以通过语音生成技术，为他们提供更好的信息获取方式。例如，将视频中的重要事件、动作和场景生成语音描述，提升用户体验。
- **安全与监控**
  - 视频监控设备生成的大量数据难以被人工分析。视频描述技术可以对视频内容进行自动总结，用于事件检测、异常分析等。

任务介绍

在本次实验中，我们主要使用的视频的图像信息，利用视觉系统来抽取特征，并进行描述。我们使用训练视频，训练了一个S2VT模型，并用它为测试集的每条视频生成一个描述语句。

训练集中，每个视频提供了5个标签语句。但是观察到，5个语句如果不尽相同，那么其实也就说明用一句话来表述视频其实是有所受限的，如何评估一句话是否准确地描述了视频，其实也是一个开放性问题。

S2VT方法介绍

S2VT 模型（Sequence to Sequence Video to Text）由 Venugopalan 等人在 2015 年首次提出，标志着视频描述生成领域的重要里程碑。这一模型首次将通用的序列到序列（Seq2Seq）框架引入视频描述任务，实现了从视频到文本的端到端生成。

模型核心思想

Seq2Seq 模型是一种深度学习架构，最初用于解决序列到序列的映射问题，例如机器翻译、对话生成等任务。在 S2VT 模型中，这一框架被巧妙地应用于视频描述领域：

1. **编码阶段**：通过一个 LSTM 网络对输入视频帧序列进行编码，将视频帧序列转化为一个固定长度的高维向量表示。
2. **解码阶段**：另一个 LSTM 网络基于编码后的向量表示，生成与视频内容相对应的自然语言描述。

这一流程利用了 LSTM 的强大时序建模能力，使得模型能够处理动态、多帧的视频数据并生成符合语法规范的文本。

输入数据形式

1. **RGB 图像输入**：

- 通过卷积神经网络（CNN）对每一帧图像提取特征，得到每个帧的特征表示。
- 这些特征表示再被输入到 LSTM 网络中进行时序建模。
- RGB 图像保留了丰富的空间信息（例如颜色、纹理和物体形状），是视频描述生成的重要信息源。

2. **光学流输入**：

- 光学流图像表示帧与帧之间的运动信息（例如物体移动和行为特征），是捕捉视频时序信息的关键。
- 相较于RGB图像，光学流专注于动态变化，可以直接输入到 LSTM 网络中，以进一步增强时序信息的建模。

通过结合 RGB 和光学流输入，S2VT 模型可以充分利用静态场景信息和动态运动信息，从而提升描述生成的准确性和流畅性。

模型优势

1. **处理可变长度的输入帧**：

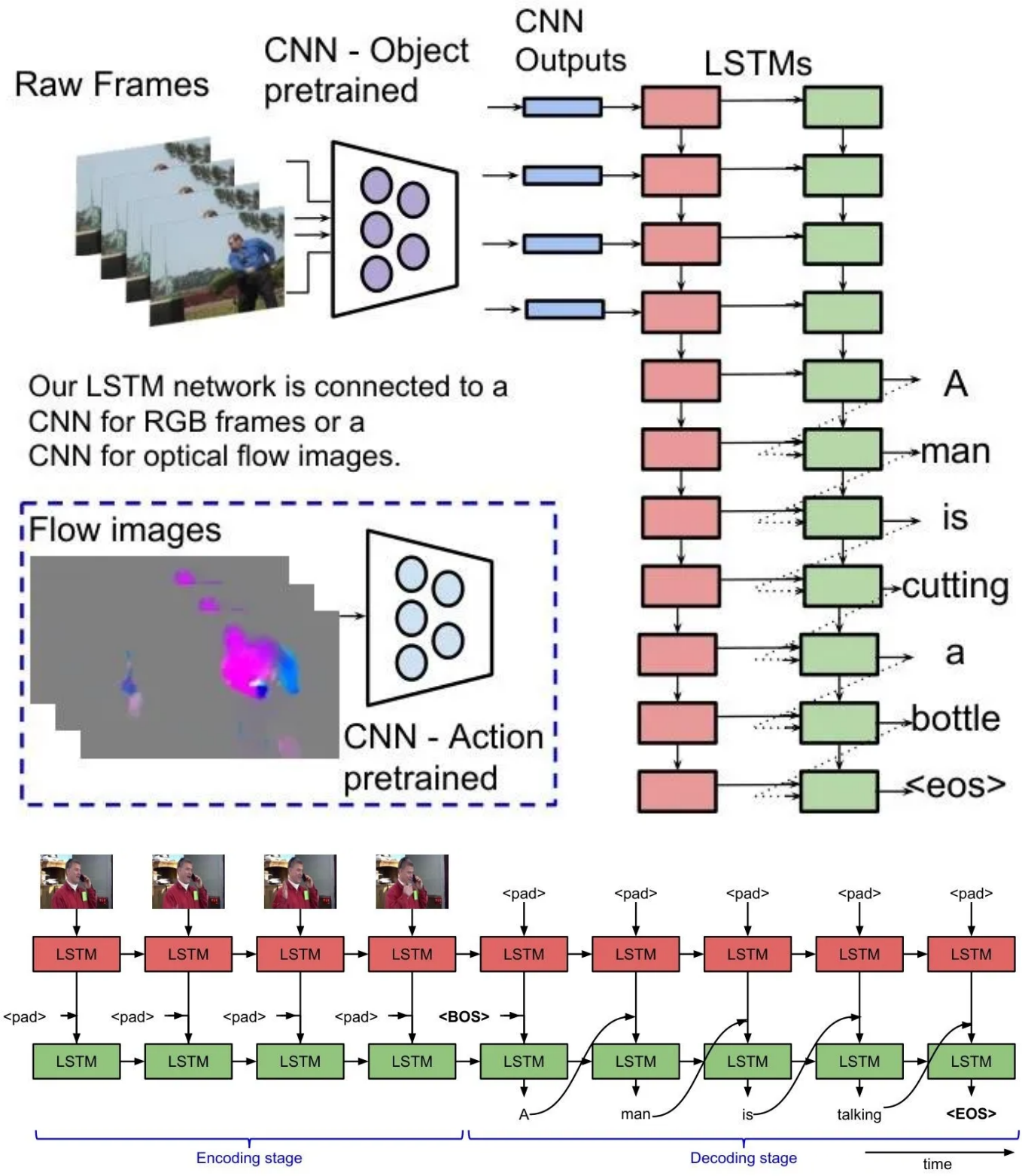
- 视频帧数量通常是可变的，而传统的 RNN 模型在处理变长序列时容易出现梯度消失或梯度爆炸问题。
- LSTM 网络通过引入门控机制（如遗忘门和输入门），有效地缓解了这些问题，使得模型能够稳定地处理长时间序列。

2. **学习视频的时序结构**：

- 视频是典型的时序数据，帧与帧之间存在强相关性。S2VT 模型通过 LSTM 捕捉这种时序依

赖关系，生成的文本描述更加准确且连贯。

- 特别是在复杂场景中，模型可以通过分析帧序列的变化动态描述事件的发展。



S2VT结构图

实验内容

核心代码

VGG抽取特征器 Python

```python
# 读取并处理输入
class VGG16FeatureExtractor(nn.Module):
    def __init__(self, model):
        super(VGG16FeatureExtractor, self).__init__()
        self.features = model.features  # VGG16卷积部分
        self.classifier = model.classifier[:6]  # 保留前6层全连接层

    def forward(self, x):
        # 获取卷积层输出
        x = self.features(x)
        # 展平卷积层的输出为(batch_size, 25088)
        x = x.view(x.size(0), -1)  # batch_size, 512 * 7 * 7 -> batch_size, 25088
        # 将展平后的特征传递给全连接层
        x = self.classifier(x)
        return x

def extract_feats_for_video(file, batch_size, device, save_dir):
    """Extract features for a single video."""
    # 加载预训练的VGG-16模型
    model = VGG16FeatureExtractor(models.vgg16(pretrained=True))
    model.eval()  # 设置为评估模式
    model.to(device)  # 移动模型到GPU或CPU

    preprocess = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        transforms.Resize((224, 224)),
    ])

    # 读取视频
    vid = imageio.get_reader(f"E:/tzy/zy/深度学习/视频描述/dataset/video/{file}", 'ffmpeg')
    curr_frames = []

    for frame in vid:
        # 调整帧大小
        frame = skimage.transform.resize(frame, [224, 224])
        if len(frame.shape) < 3:
            frame = np.repeat(frame, 3).reshape([224, 224, 3])
        frame = Image.fromarray((frame * 255).astype(np.uint8))
        curr_frames.append(preprocess(frame))


```

```
        curr_frames = torch.stack(curr_frames).to(device)  # 将图像移动到 GPU/CP
U


        idx = np.linspace(0, len(curr_frames) - 1, 80).astype(int)  # 获取80帧
        curr_frames = curr_frames[idx]

        curr_feats = []
        for i in range(0, 80, batch_size):
            curr_batch = curr_frames[i:i + batch_size]
            with torch.no_grad():
                features = model(curr_batch)
                curr_feats.append(features.cpu().numpy())

        curr_feats = np.concatenate(curr_feats, axis=0)
        save_path = os.path.join(save_dir, f"{file[:-4]}.npy")
        np.save(save_path, curr_feats)
        print(f"Saved features for {file} to {save_path}")
```

1. **特征提取器的设计**：
   - 基于预训练的VGG16网络构建了一个特征提取器
   - 保留了VGG16的所有卷积层和前6层全连接层
   - 去掉了最后一层全连接层，这样输出的是视频的特征表示，而不是分类结果
2. **视频处理流程**：
   - 首先读取指定路径下的视频文件
   - 对视频进行采样，从整个视频中均匀抽取80帧画面
   - 每一帧都会经过预处理：
     - 调整尺寸为224x224（VGG16的标准输入尺寸）
     - 转换为张量格式
     - 进行标准化处理（使用ImageNet数据集的均值和标准差）
3. **批处理特征提取**：
   - 将采样得到的80帧分成多个批次进行处理
   - 每个批次的帧会同时送入VGG16模型
   - 使用torch.no_grad()确保不计算梯度，提高处理效率
   - 模型输出的是每一帧的4096维特征向量（来自倒数第二个全连接层）
4. **特征保存**：
   - 将所有批次的特征向量合并

- 保存为.npy文件，文件名与视频名对应
- 每个视频最终得到一个shape为(80, 4096)的特征矩阵，表示80帧画面的特征

S2VT模型 Python

```python
class S2VT(nn.Module):
    def __init__(self, vocab_size, batch_size=10, frame_dim=4096, hidden=500, dropout=0.5, n_step=80):
        super(S2VT, self).__init__()
        self.batch_size = batch_size
        self.frame_dim = frame_dim          # 视频特征维度
        self.hidden = hidden                # 隐藏层维度
        self.n_step = n_step                # 时间步长（视频帧数）

        self.drop = nn.Dropout(p=dropout)
        self.linear1 = nn.Linear(frame_dim, hidden)    # 视频特征降维
        self.linear2 = nn.Linear(hidden, vocab_size)   # 输出层，映射到词表大小

        # 编码器LSTM
        self.lstm1 = nn.LSTM(hidden, hidden, batch_first=True, dropout=dropout)
        # 解码器LSTM，输入维度是2*hidden因为包含了视频特征和词嵌入的拼接
        self.lstm2 = nn.LSTM(2*hidden, hidden, batch_first=True, dropout=dropout)

        self.embedding = nn.Embedding(vocab_size, hidden)  # 词嵌入层

    def forward(self, video, caption=None):
        # 重塑视频特征维度
        video = video.contiguous().view(-1, self.frame_dim)
        video = self.drop(video)
        video = self.linear1(video)                    # 视频特征降维
        video = video.view(-1, self.n_step, self.hidden)
        # 在时间维度上填充视频特征
        padding = torch.zeros([self.batch_size, self.n_step-1, self.hidden]).cuda()
        video = torch.cat((video, padding), 1)         # 视频输入
        vid_out, state_vid = self.lstm1(video)        # 通过编码器LSTM

        if self.training:
            # 训练模式
            caption = self.embedding(caption[:, 0:self.n_step-1])  # 对输入的描述文本进行词嵌入
            padding = torch.zeros([self.batch_size, self.n_step, self.hidden]).cuda()
            caption = torch.cat((padding, caption), 1)         # 描述文本填充
            caption = torch.cat((caption, vid_out), 2)         # 将视频特征和描述文本拼接

```

```
            cap_out, state_cap = self.lstm2(caption)  # 通过解码器LSTM
            # cap_out的大小是 [batch_size, 2*n_step-1, hidden]
            cap_out = cap_out[:, self.n_step:, :]    # 只保留生成的描述部分
            cap_out = cap_out.contiguous().view(-1, self.hidden)
            cap_out = self.drop(cap_out)
            cap_out = self.linear2(cap_out)          # 映射到词表大小
            return cap_out
            # 输出大小 [batch_size*79, vocab_size]
        else:
            # 测试模式（生成描述）
            padding = torch.zeros([self.batch_size, self.n_step, self.hidden]).cuda()
            cap_input = torch.cat((padding, vid_out[:, 0:self.n_step, :]), 2)
            cap_out, state_cap = self.lstm2(cap_input)
            # 第二层LSTM的填充输入，80个时间步

            # 生成开始符号<BOS>的批处理张量
            bos_id = word2id['<BOS>']*torch.ones(self.batch_size, dtype=torch.long)
            bos_id = bos_id.cuda()
            cap_input = self.embedding(bos_id)
            cap_input = torch.cat((cap_input, vid_out[:, self.n_step, :]), 1)
            cap_input = cap_input.view(self.batch_size, 1, 2*self.hidden)

            # 生成第一个词
            cap_out, state_cap = self.lstm2(cap_input, state_cap)
            cap_out = cap_out.contiguous().view(-1, self.hidden)
            cap_out = self.drop(cap_out)
            cap_out = self.linear2(cap_out)
            cap_out = torch.argmax(cap_out, 1)
            # 输入"<BOS>"开始生成过程

            caption = []
            caption.append(cap_out)
            # 将生成的词索引添加到caption列表中，每个时间步为每个批次生成一个词

            # 循环生成剩余的词
            for i in range(self.n_step-2):
                cap_input = self.embedding(cap_out)
                cap_input = torch.cat((cap_input, vid_out[:, self.n_step+1+i, :]), 1)
                cap_input = cap_input.view(self.batch_size, 1, 2 * self.hidden)

                cap_out, state_cap = self.lstm2(cap_input, state_cap)
                cap_out = cap_out.contiguous().view(-1, self.hidden)
```

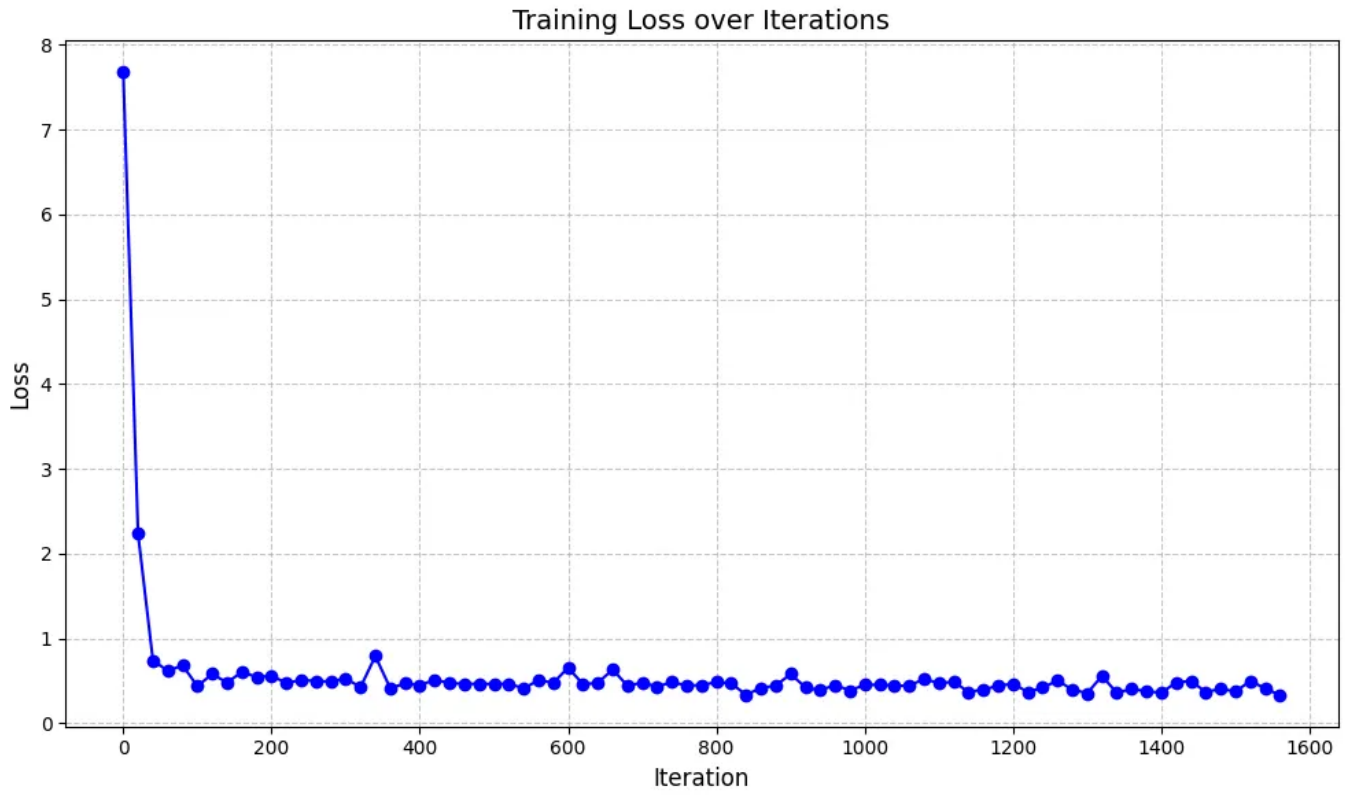
```
                cap_out = self.drop(cap_out)
                cap_out = self.linear2(cap_out)
                cap_out = torch.argmax(cap_out, 1)    # 获取词表中概率最高的词的索引
                caption.append(cap_out)
            return caption  # caption的大小为 [79, batch_size]
```

这个S2VT模型的工作原理可以分为以下几个关键部分：

1. **模型结构设计**：
   - 包含两个LSTM层：编码器LSTM和解码器LSTM
   - 使用词嵌入层将词转换为向量表示
   - 包含视频特征降维层和最终输出层
   - 使用dropout来防止过拟合
2. **视频特征处理**：
   - 首先对输入的视频特征进行降维处理（从4096维降到hidden维）
   - 通过编码器LSTM处理视频序列
   - 在时间维度上进行填充，以匹配描述生成的长度需求
3. **训练模式下的工作流程**：
   - 将输入的描述文本转换为词嵌入表示
   - 将视频特征和描述文本进行拼接
   - 通过解码器LSTM生成预测
   - 输出每个时间步的词表概率分布
   - 可以直接用于计算损失函数进行训练
4. **测试模式下的工作流程**（生成描述）：
   - 首先输入开始符号 `<BOS>`
   - 逐词生成描述：
     - 将上一个生成的词进行词嵌入
     - 与对应时间步的视频特征拼接
     - 通过解码器LSTM预测下一个词
     - 选择概率最高的词作为输出
   - 重复这个过程直到生成完整描述
5. **特点和创新**：
   - 采用了编码器-解码器架构

- 在解码阶段同时利用了视频特征和文本特征
- 可以处理变长的输入视频序列
- 能够生成流畅的描述文本

这个模型实现了一个端到端的视频描述生成系统，将视频理解和自然语言生成有机地结合在一起。在训练时使用教师强制（teacher forcing）策略，而在测试时采用自回归方式生成描述。



训练损失

测试集结果示例

video id	row id	caption
G_23245	2	people digging soil in the river
G_23250	8	rescue workers and a person in the workshop
G_23354	22	in suits and two women are talking
G_23514	38	many cars driving on the road