

ACTION SPOTTING AND CAPTIONING OF SOCCER MATCHES USING OBJECT-SCENE RELATIONAL GRAPHS AND LANGUAGE LSTM

A PROJECT REPORT

Submitted by

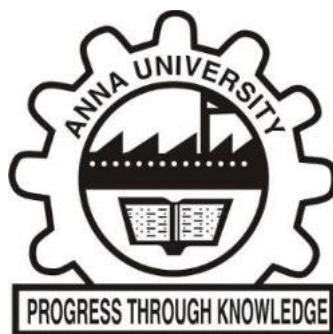
AKSHARA M S **2019103004**

HARINI E R **2019103019**

AKSHAYALAKSHMI V K 2019103505

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING



COLLEGE OF ENGINEERING GUINDY CAMPUS
ANNA UNIVERSITY :: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY :: CHENNAI 600 025
BONAFIDE CERTIFICATE

Certified that this project report “**ACTION SPOTTING AND CAPTIONING OF SOCCER MATCHES USING OBJECT-SCENE RELATIONAL GRAPHS AND LANGUAGE LSTM**” is the bonafide work of “**Akshara M S (2019103004), Harini E R (2019103019) and Akshayalakshmi V K (2019103505)**” who carried out the project work under my supervision, for the fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported here in does not form part of any thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates

Place: Chennai

Date: 05/06/2023

Dr. K. S. Easwarakumar

Professor

Department of Computer Science and Engineering

Anna University, Chennai - 25

COUNTERSIGNED

Dr. S. Valli

Head of the Department

Department of Computer Science and Engineering,

Anna University, Chennai – 600 025

ACKNOWLEDGEMENT

We express our deep gratitude to our guide, **Dr. K. S. Easwarakumar**, Professor, Department of Computer Science and Engineering for guiding us through every phase of the project. We appreciate his thoroughness, tolerance and ability to share his knowledge with us. We would also like to thank him for his kind support and for providing necessary facilities to carry out the work.

We are extremely grateful to **Dr. S. Valli**, Professor & Head of the Department, Department of Computer Science and Engineering, Anna University, Chennai – 25, for extending the facilities of the Department towards our project and for her unstinting support.

We express our thanks to the panel of reviewers **Dr. S. Bose**, Professor, Department of Computer Science and Engineering and **Dr. P. Geetha**, Professor Department of Computer Science and Engineering for their valuable suggestions and critical reviews throughout the course of our project.

We express our thanks to all other teaching and non-teaching staff who helped us in one way or other for the successful completion of the project. We would also like to thank our parents, family and friends for their indirect contribution in the successful completion of this project.

AKSHARA M S

HARINI E R

AKSHAYALAKSHMI V K

ABSTRACT

Soccer is one of the most prominent sports followed by many people worldwide. Various immersive and non-immersive technologies have been incorporated to enhance the overall experience of the audience. One such enhancement is the automatic captioning of soccer matches. Present video captioning techniques leverage the optical and motion features of the video frames majorly to generate the appropriate caption. But the same method cannot be applied for soccer since the differentiability of appearance features of various participating elements is low. Hence, we propose a novel captioning system that generates captions for soccer matches from visual features and other temporal features along with hierarchical attention. Specifically, our project aims to automatically generate captions for soccer matches using a combination of object scene relational graph and spatial-temporal attention LSTM. Temporal significance of frames is obtained through pooling layers and the audio segments. The caption generation segment involves a hierarchical LSTM with spatial temporal attention mechanism to enable us focus on the most relevant objects and actions at each time step. The proposed system improves upon existing captioning techniques by incorporating object tracking and temporal attention in a single framework. It has achieved a Meteor score of 46.9 and Rouge-L of 39.5 which is comparable to the state-of-the-art results on the benchmark dataset.

திட்டப்பணி சுருக்கம்

உலகளவில் பலர் பின்பற்றும் முக்கிய விளையாட்டுகளில் கால்பந்து ஒன்றாகும். பார்வையாளர்களின் ஒட்டுமொத்த அனுபவத்தை மேம்படுத்த பல்வேறு தொழில்நுட்பங்கள் இணைக்கப்பட்டுள்ளன. அத்தகைய ஒரு மேம்பாடு கால்பந்து போட்டிகளின் தானியங்கி தலைப்பு ஆகும். தற்போதைய காணொளி-தலைப்பு நுட்பங்கள் பொருத்தமான தலைப்பை உருவாக்க காணொளிச்சட்டங்களின் ஒளி மற்றும் அசைவு அம்சங்களை முக்கியமாகப் பயன்படுத்துகின்றன. ஆனால் அதே முறையை கால்பந்துக்கு பயன்படுத்த முடியாது, ஏனெனில் பல்வேறு பங்கேற்புக் கூறுகளின் தோற்ற அம்சங்களின் வேறுபாடு குறைவாக உள்ளது. எனவே, காட்சி அம்சங்கள் மற்றும் பிற தற்காலிக அம்சங்களிலிருந்து, படிநிலை கவனத்துடன் கால்பந்து போட்டிகளுக்கான தலைப்புகளை உருவாக்கும் புதிய தலைப்பு அமைப்பை நாங்கள் முன்மொழிகிறோம். குறிப்பாக, பொருள்-காட்சி தொடர்புடைய கோட்டுரு மற்றும் இடஞ்சார்ந்த-தற்காலிக கவனம் பொருந்திய நீண்ட குறுகிய-கால நினைவகப்பிணையம் ஆகியவற்றின் கலவையைப் பயன்படுத்தி கால்பந்து போட்டிகளுக்கான தலைப்புகளைத் தானாகவே உருவாக்குவதை எங்கள் திட்டம் நோக்கமாகக் கொண்டுள்ளது. காணொளிச்சட்டங்களின் தற்காலிக முக்கியத்துவம் ஒன்றிணைக்கும்

அடுக்குகள் மற்றும் ஒலிப்பிரிவுகள் மூலம் பெறப்படுகிறது. தலைப்பு உருவாக்கப் பிரிவில், ஒவ்வொரு கால கட்டத்திலும் மிகவும் பொருத்தமான பொருள்கள் மற்றும் செயல்களில் கவனம் செலுத்துவதற்கு இடஞ்சார்ந்த தற்காலிக கவனப்பொறிமுறையுடன் கூடிய படிநிலை நீண்ட குறுகிய-கால நினைவகப்பிணையத்தை உள்ளடக்கியது. முன்மொழியப்பட்ட அமைப்பு, பொருள் கண்காணிப்பு மற்றும் தற்காலிக கவனத்தை ஒரே கட்டமைப்பில் இணைப்பதன் மூலம் தற்போதுள்ள தலைப்பு நுட்பங்களை மேம்படுத்துகிறது. இது 46.9 என்ற மெட்டியோர் மதிப்பெண்ணையும், 39.5 என்ற ரூக்-எல் மதிப்பெண்ணையும் அடைந்துள்ளது, இது தரநிலை தரவுத்தொகுப்பில் உள்ள அதிநவீன முடிவுகளுடன் ஒப்பிடத்தக்கது.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT – ENGLISH	iv
	ABSTRACT – TAMIL	v
	LIST OF TABLES	x
	LIST OF FIGURES	xi
	LIST OF SYMBOLS	xiv
1	INTRODUCTION	1
1.1	OVERVIEW	1
1.2	OVERALL OBJECTIVES	4
1.3	PRESENT ISSUES	4
1.4	PROBLEM STATEMENT	5
1.5	ORGANIZATION OF THE THESIS	5
2	LITERATURE SURVEY	6
3	SYSTEM DESIGN	10
3.1	SYSTEM ARCHITECTURE	10
3.2	FRAME FEATURE EXTRACTION	10
3.2.1	DATA PREPARATION	10
3.2.2	FRAME FEATURE EXTRACTION	11

3.3	OBJECT SCENE GRAPH CONSTRUCTION	13
3.4	AUDIO FEATURE EXTRACTION	17
3.5	ACTION SPOTTING	18
3.6	CAPTION GENERATION	19
3.6.1	CAPTIONING MODULE	20
3.6.2	ATTENTION LSTM	20
3.6.3	TEMPORAL ATTENTION LAYER	20
3.6.4	SPATIAL ATTENTION LAYER	21
3.6.5	LANGUAGE LSTM	21
4	IMPLEMENTATION AND RESULTS	24
4.1	DATASET	24
4.2	EXPERIMENTAL SETUP	24
4.3	IMPLEMENTATION AND RESULTS	25
4.3.1	FRAME FEATURE EXTRACTION	25
4.3.2	OBJECT SCENE-GRAPH CONSTRUCTION	27
4.3.3	AUDIO FEATURE EXTRACTION	34
4.3.4	ACTION SPOTTING	37
4.3.5	CAPTION GENERATION	41

4.4	PERFORMANCE METRICS	45
4.5	COMPARATIVE ANALYSIS	49
4.6	TEST CASES AND VALIDATION	51
5	CONCLUSION AND FUTURE WORK	55
5.1	CONCLUSION	55
5.2	FUTURE WORK	55
	REFERENCES	56

LIST OF TABLES

4.1	Performance measured with the denoted metrics	49
4.2	Comparison of different approaches	50
4.3	Expected and Predicted outputs for each test case	51

LIST OF FIGURES

3.1	System Architecture	10
3.2	Frame feature extraction	11
3.3	Object scene-graph construction	13
3.4	Audio feature extraction	17
3.5	Action spotting	18
3.6	Caption generation	19
4.1	Video Clipping	25
4.2	Output of Video clipping	25
4.3	Extracted 30s video clips	26
4.4	Implementation of frame feature extraction	26
4.5	Execution of frame feature extraction	27
4.6	Obtained frame features	27
4.7	Implementation of object detection and instance segmentation	28
4.8	Driver code for object detection	29
4.9	Object detection and Instance segmentation done on a sample frame	29
4.10	Object Appearance feature extraction	30

4.11	Output for appearance feature extraction	31
4.12	Extracted appearance features	31
4.13	Implementation of graph construction	32
4.14	A Sample scene and its constructed scene graph	32
4.15	Implementation of GCN	33
4.16	GCN training	33
4.17	Node embeddings obtained for a single frame of a video clip	34
4.18	Implementation of VGGish	35
4.19	Extracting embeddings from VGGish	35
4.20	Output of audio feature extraction	36
4.21	Audio features in files	36
4.22	Extracted audio features	36
4.23	VGGish Embedding obtained for the audio of a video clip	37
4.24	Audio Embedding after dimensionality reduction	37
4.25	Loss function for NetVLAD++ model	38
4.26	Implementation of NetVLAD++ model	38
4.27	Extracted NetVLAD++ features	39

4.28	Action spotted for two different test data (a correctly and a wrongly predicted instance)	39
4.29	Temporal uniformity observed in the NetVLAD++ embedding of two different clips of same action	40
4.30	Implementation of temporal attention layer	42
4.31	Implementation of Spatial attention layer	42
4.32	The Whole hierarchical encoder-decoder model	43
4.33	Time-step wise implementation of attention mechanisms	43
4.34	Obtaining final model state from the attention layer	44
4.35	Compiling and training the model	44
4.36	Testing the model	44
4.37	Obtained captions for a test sample	45
4.38	Obtained captions	45
4.39	Average mAP	46
4.40	BLEU Score	47
4.41	Meteor Score	48
4.42	Rouge-L Score	48
4.43	Comparison of different approaches	50

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

CNN	Convolutional Neural Network
GCN	Graph Convolutional Network
RCNN	Convolutional Neural Network
VGG	Visual Geometry Group
UMAP	Uniform Manifold Approximation and Projection
LSTM	Long Short-Term Memory
LLM	Large Language Models
VLAD	Vector of Locally Aggregated Descriptors

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Caption generation from videos has become a recent trend in computer vision. Because of its broad range of actions, several studies have been conducted to test novel methods for the same.

Video captioning involves the process of adding text captions to a video to make it accessible to people who are deaf or hard of hearing, as well as to those who may not be able to turn on the audio due to their surroundings or preferences.

Challenges in this task are two-fold: video content understanding and sequence modeling of uttering words to successive video frames.

There are various methods for video captioning, including automatic speech recognition (ASR) technology and human transcription. ASR technology uses machine learning algorithms to recognize and transcribe speech in real-time, while human transcription involves a person manually transcribing the speech in the video.

While ASR technology has improved significantly in recent years, it still has some limitations and may not accurately transcribe certain words or phrases.

In addition to making videos accessible to more people, video captioning also has other benefits. It can improve search engine optimization of a video by making it more searchable and discoverable online.

The goal of sports video captioning is to create captions from clips of original match videos. Sports video captioning has become increasingly important in recent days as more people rely on online video content to follow their favorite sports teams and athletes. Captions can provide additional information such as player names, stats, and scores, which can help viewers stay

informed and engaged with the game. They can also be a useful tool for analyzing and studying game footage, as captions can help viewers focus on specific moments and details in the video.

Sports video captioning differs from normal video captioning in many ways owing to the presence of new challenges. Sports videos are typically fast-paced, with quick movements and changes happening on-screen. This can make it challenging to capture and convey all the key moments and highlights accurately in the captions.

Each sport has its own specialized terminology and jargon, which can be challenging for human transcribers to understand and transcribe accurately.

Sports videos are often broadcast live, which requires real-time captioning to provide immediate access to the captions for viewers. This can be challenging for both ASR technology and human transcribers, as they must keep up with the fast-paced action and deliver accurate captions in real-time

Normal captioning need not group players (people) or identify places accurately. Some words which have significant meaning in the sports domain may be considered insignificant and would be discarded most of the times in normal captioning.

Soccer captioning also require significant information about the movement of the ball and actions based on the motion of ball and players. It should be able to differentiate among the two teams especially important activities like goal and own goal.

Soccer captioning is a challenging task that requires the integration of multiple computer vision and natural language processing techniques. These techniques must be able to detect and track individual players, recognize and classify soccer actions, and generate fluent and coherent natural language descriptions of the events taking place on the field.

There have been several notable works in the field of soccer captioning that have made significant contributions to the development of this technology.

One of the earliest works in this field is the SoccerNet dataset, which was released in 2019 and has since become a widely-used benchmark for soccer captioning research. The dataset contains over 500 full-length soccer matches and includes annotations for various soccer events, such as goals, shots, and fouls.

Another notable work is the SoccerNet-v2 dataset, which was released in 2020 and builds upon the original SoccerNet dataset by including additional annotations for events such as passes, tackles, and interceptions. The dataset also includes a new task of long-term forecasting, which involves predicting events that will occur several seconds in the future.

Overall, the development of soccer captioning technology has been an active area of research in recent years, with certain notable works contributing to this field. These works have advanced the state-of-the-art in soccer captioning and have opened up new possibilities for automated soccer analysis and broadcasting.

Notable modelling works in soccer video captioning rely only on the NetVLAD++ and related pooling strategies to address these issues. They obtain frame features directly and use them for classification. A very few papers have experimented trajectory based or object-based tracking and captioning. Some papers have implemented optical feature-based reasoning. These approaches, though were found to be effective, do not take into account the temporal importance of each frame of a clip in contributing to a caption.

Hence, we aim to implement a novel object-scene graph-based caption generation method to incorporate spatial importance of various objects of a frame and a temporally-aware pooling technique to contribute to the temporal awareness of the video. We also use audio features as a significant temporal feature of a video clip. Along with the mentioned features, we develop a

hierarchical encoder-decoder architecture with spatial and temporal attention mechanisms to generate the caption.

1.2 OVERALL OBJECTIVES

- To represent video frames and object interactions through object-scene relational graphs.
- To identify significant soccer actions, present on the video clippings.
- To generate meaningful and domain-specific captions using the spotted action.
- To analyze the performance metric of the proposed model with the state-of-the-art models.

1.3 PRESENT ISSUES

Current video captioning methods rely on extracting appearance features from various objects of the frame and use it as an input to the encoder. This method doesn't effectively work on Soccer videos owing to the small size of players on the field, small size of the ball and their fast-paced movements.

Extracting the activity features alone can also not serve as a sufficient feature to classify actions since the number of conventional actions performed is limited (eg, activity recognition on soccer videos would detect that a player is kicking a ball)

Another major issue rises due to the occlusion of ball and some players by other players in the field. Moreover, the temporal importance of occurrence of events is not concentrated

Thus, to meaningfully generate domain related captions, both appearance and temporally significant features need to be encoded.

1.4 PROBLEM STATEMENT

Action spotting and captioning of soccer matches play a very vital role in automatic identification of highlights from a video and easier extraction of videos with certain highlights. Current action spotting techniques identify an action majorly using object's appearance features followed by its pooling with frame appearance features, thereby neglecting the role of directly modelled interactions. Hence, we extend the current spotting techniques by introducing scene-relational graphs to include relationship between the identified players and use temporally significant features such as the audio features and the features obtained from temporally pooled layers to generate fine-grained captions with more domain-specific words.

1.5 ORGANIZATION OF THE THESIS

The thesis is organized as followed. Chapter 2 discusses about the existing methods in the literature utilized for captioning systems. Chapter 3 introduces the architecture diagram along with the details of each of the module. It elaborates on each of the component module along with the obtained output. Chapter 4 discusses on the performance metric used in the work and the values we obtained. A comparative analysis of the results is also found there. Chapter 6 concludes the present work while discussing on the criticisms and scope for future work.

CHAPTER 2

LITERATURE SURVEY

The video captioning methodologies can be categorized into two, template-based methods and deep learning-based methods [1]. In template-based methods, a predefined set of specific grammar rules are used. Sentences are split into subject, object and verb and are associated with video data. In deep learning-based methods encoder-decoder framework that includes a semantic concept detector is used.

In [15] Wu et al proposed a CNN based encoder-decoder framework for video captioning. They append inter-frame differences to each CNN-extracted frame feature to get a more discriminative representation; then with that as the input, they encode each frame to be a more compact feature by a one-layer convolutional mapping, which could be taken as a reconstruction network.

In the decoding stage, they first fuse visual and lexical feature. Then they stack multiple dilated convolutional layers to form a hierarchical decoder. As long as term dependencies could be captured by a shorter path along the hierarchical structure, the decoder could alleviate the loss of long-term information.

They use MCF to fuse the visual and lexical features, and use multiple dilated convolution layers to construct the decoder.

The framework is added with a discriminative enhancer to increase the feature variance between two frames. The discriminative compact feature is taken as input to reduce long term dependencies.

In [5] Lianli Gao et al proposed a fused GRU with semantic temporal attention pipeline that can explicitly incorporate high-level visual concepts for video captioning. These concepts boost the performance of captioning. The pipeline consists of an encoder network that encodes the visual input into a

feature vector as the visual representation and a decoder network generates a natural sentence by decoding this visual representation.

In [16] Bairui Wang proposed an encoder-decoder-reconstructor architecture to leverage the forward flow and backward flow together for video captioning.

In[12], Masoomah Nabati et al model incorporated a content-based beam search algorithm that uses an object detector and a structured dictionary of sentences into a model. This is aided by a multi stage refinement algorithm that removes incorrect and unstructured sentences.

Sports video captioning differs from the usual video captioning methods owing to the presence of a number of challenges. Primarily, the video encoding module should effectively identify groups of players and field positions accurately to evaluate any given video clip. Secondly, the importance of domain-specific words needs to be emphasized in the process of creating captions (left and right can be considered general words in conventional methods and can be discarded but they play a vital role in sports captioning).

Existing action spotting attempts in soccer videos are mainly based on NetVLAD and NetVLAD++ [7]. In [7], the authors developed a pooling module called NetVLAD++ to insert temporal context in both past and future contexts of an action. It uses two NetVLAD pooling layers to obtain an encoded representation of video slices. The encodings are then fed to a classification layer to identify the action present in the clip. This system has simple architecture and less dimensions are required due to the pooling. Thus this is considered the standard benchmark for soccer action spotting

Further works on NetVLAD++ for action spotting have also been supported by the works of [19] and [8].

Qi et al[14] contrasts the importance of modelling object interactions and trajectories in action spotting and proposes a trajectory-based attention model to identify actions. The model uses a hierarchical decoder to generate captions from the key frames by capturing the spatial and temporal features of various frames of the clipping.

A recent work by Hua et al., [9] uses object-scene relational graphs to find the interactions of objects and their interactions with the environment. The study also uses a trajectory-based feature extraction module to study the objects motion features. They are combined and fed into an LSTM to produce the caption via Adversarial Reinforcement training.

Zhou et al., [19] extracts the same video features aggregated from a variety of models and uses the NetVLAD++ pooling to identify the action. This can provide more fine-tuned results as the value is aggregated over a variety of backbone models. But since the sampling frequency of features is too low it is difficult to compute the tight average mAP.

Studies have also been initiated to find the effectiveness of graph-based models in soccer action spotting. In [3], the authors constructed a relation graph between players, objects and the environment frame-wise to identify the action. Hammoudeh et al., [8] experimented captioning from action spotting using a transformer-based model and generated captions with the weighted vectors of those actions. Here the flow-level features improve the effectiveness of the segmentation and caption generation, a new loss function that gives importance to domain specific strong words. But on the other hand, this system requires as ground truth, the complete caption of the clip, which may not be available many times, conversion from commentary to caption gives raise to human related bias errors.

Thus, in order to generate fine-grained, semantically meaningful captions, the action present in a video needs to be identified and should be weighted more

as compared to the other describing words. This study aims at building a novel soccer captioning system built on top of action spotting and modelling player's interactions and attributes using object scene-graphs.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

The architecture of the proposed system is shown in Figure 3.1

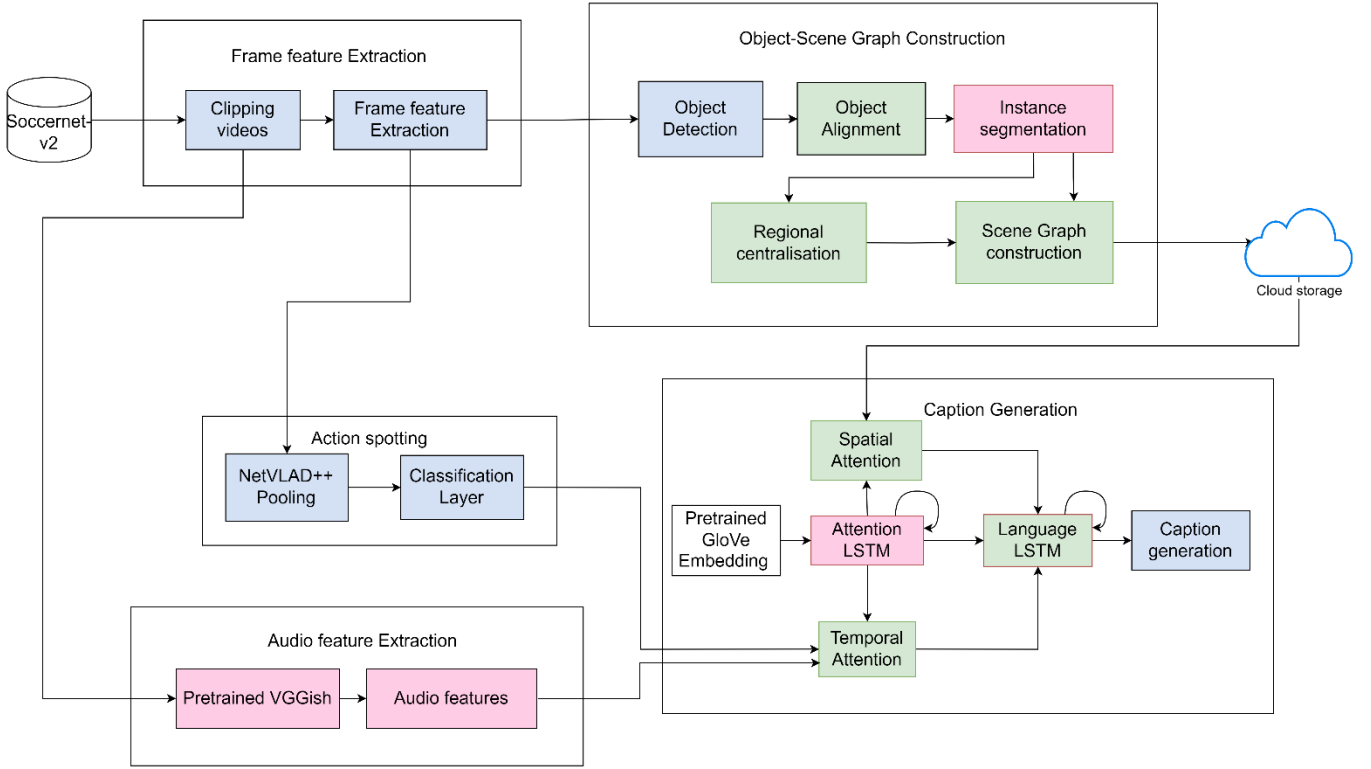


Figure 3.1 System Architecture

The complete framework can be divided into 5 main modules.

3.2 FRAME FEATURE EXTRACTION

3.2.1 Data preparation

Firstly, the complete match video is clipped into non-overlapping frames of 30 seconds each with one frame as a striding frame in between two clips. The frames are extracted with a sampling rate of 2 frames per second by uniformly skipping certain frames every second from the video that was originally sampled at 25fps. The module design for frame feature extraction is shown in Figure 3.2

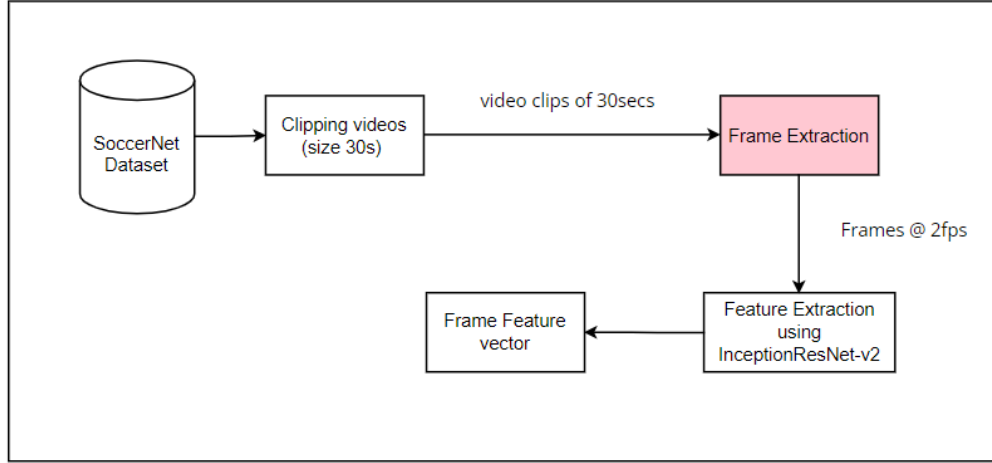


Figure 3.2 Frame feature extraction

Input: A match's half-time video

Output: Frames extracted per 30 second of the video sampled at 2 fps. The output is of dimension (N, 60, H, W) where,

N – Number of 30 second clippings generated

H – Height of each frame

W – Width of each frame

3.2.2 Frame feature extraction

From the frames extracted, the global frame attributes are extracted from the pretrained 2DCNN model. A ResNet-152 model pretrained on ImageNet is used for extracting the frame features learned by the CNN from the layer before its classification layer to better encode the attributes. Thus, a 1532-dimensional frame feature vector is obtained for each frame.

Input: A 30-second match clipping

Output: Frame features extracted for each frame from 30-second clippings of the matches. The received output is of dimension (60, 1, 1532) for each clipping.

Algorithm: Frame Feature Extraction

Video Clipping

for each game:

identify start and end timesteps to slice for a duration of 30 seconds each

obtain the sliced clips

Frame selection

for each video clip

initialize placeholder 'frames'

extract the 25 frames for each second (since the original video is sampled at 25 frames per second)

initialize start frame and set 'skip_factor' to 12

initialize 'present_frame' to the starting frame

until 'present_frame' is valid

append 'present_frame' to 'frames'

advance 'present_frame' to the frame after 'skip factor' number of frames

Frame feature extraction

initialize 'model' to an InceptionResnetV2 model

load its pretrained weights

initialize the placeholder 'clips' to hold frame features of all clips

for each video clip

initialize a placeholder 'features' for frame features

for each frame

extract appearance features from FC7 layer of the model

append to the 'features'

append 'features' to 'clips'

3.3 OBJECT SCENE GRAPH CONSTRUCTION

The module design for object scene graph creation is shown in Figure 3.3

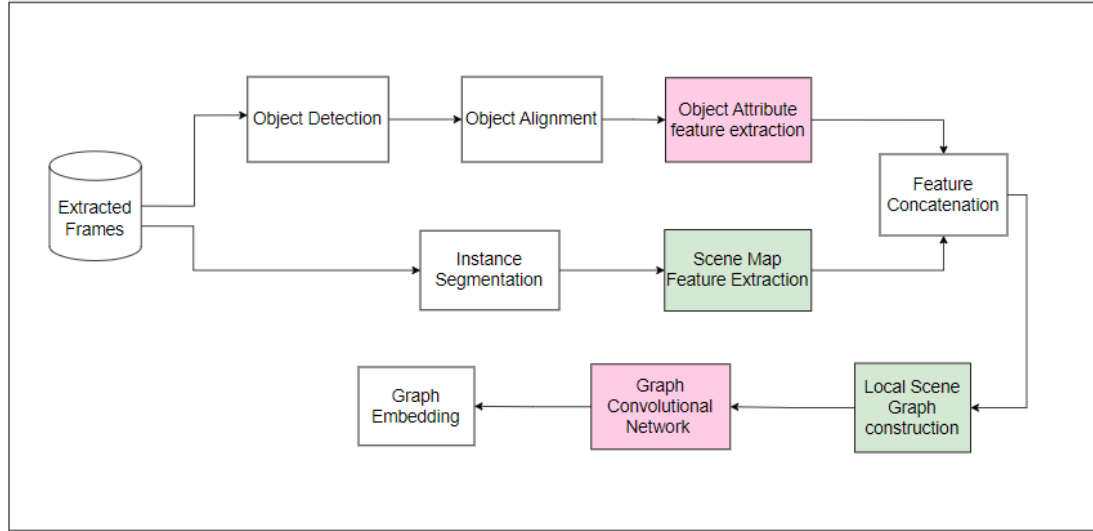


Figure 3.3 Object scene-graph construction

The frames thus extracted from the first step are fed to be segmented by an instance segmentor after finding the bounding boxes of objects in it. The segmentor identifies various segments in the input frame through the bounding boxes of the objects in the frame. The segmentation results, which are represented as pixel value maps are then processed to produce an object-scene correlation matrix. Object-appearance features are also extracted from the region of interests corresponding to the object boundary boxes through a pretrained-2DCNN.

The objects obtained from object detection are aligned together in one frame as local aligned features. The objects in the first frame are considered as anchors. The objects in different frames are aligned to these anchors according to the maximum similarity, using the cosine similarity as a measure.

The object scene graph is then constructed using the centroids of the bounding boxes as the position of object nodes and the acquired appearance and

scene correlation features as their attributes. Nodes that are nearer to each other are connected by edge. This graph is fed into a pre-trained graph convolutional network to acquire the local scene graph embedding of the frame.

Input: Video frames

Output: Local scene graph embeddings for each frame of the complete clip.

Algorithm: Object Scene Graph Creation

Object detection

```
initialise 'model' as MaskTrackRCNN  
load pretrained weights of model  
for each clip  
    initialize placeholder 'frame_objects'  
    for each frame  
        initialize placeholder 'object_bboxes'  
        identify aggregate regions from frame  
        for each identified region  
            identify regions of interest containing objects  
            identify bounding boxes for the region  
            append coordinates to 'object_bboxes'  
        append 'object_bboxes' to 'frame_objects'  
    append 'frame_objects' to 'frames'
```

Instance segmentation

```
initialize 'model' as MaskTrackRCNN  
load pretrained weights of model  
for each clip
```

initialize placeholder 'frame_scenes'

for each frame

initialize placeholder 'scene_maps'

for each bounding box in the frame:

identify region of interest

create binary mask of region with respect to frame

initialise placeholder 'scene_feature'

divide the frame into 9 equal regions

mean pool the values for each region

append these values to 'scene_feature'

append 'scene_feature' to 'scene_maps'

append 'scene_maps' to 'frame_scenes'

Object feature creation

initialise 'model' as MaskTrackRCNN

load pretrained weights of model

for each clip

for each frame

for each object

obtain object appearance features from the last layer of the model

Add scene feature and bounding box coordinate to the obtained feature to receive the final object features of an object

Graph Construction

for each clip

for each frame

initialize an empty graph 'G'

for each detected object

create a node 'N' in 'G'

set the attributes of 'N' as the previously obtained object features

for every pair of nodes in G

if distance < 200 pts

add edge 'E' connecting 'N1' and 'N2'

Obtaining Graph Embedding

initialize model as a 'GatedGCN'

train model with the present graphs

for each clip

initialize placeholder 'embeddings'

for each graph 'G'

obtain the features from last layer of model when fed with 'G'

append features to 'embeddings'

Frameworks

- (i) **Object Detector** – MaskTrackRCNN with ResNeXt101 backbone
- (ii) **Instance segmentor** – MaskTrackRCNN with ResNeXt101 backbone
- (iii) **Graph embedding** – Graph Convolutional Network

3.4 AUDIO FEATURE EXTRACTION

Figure 3.4 shows the design of the module Audio Feature Extraction

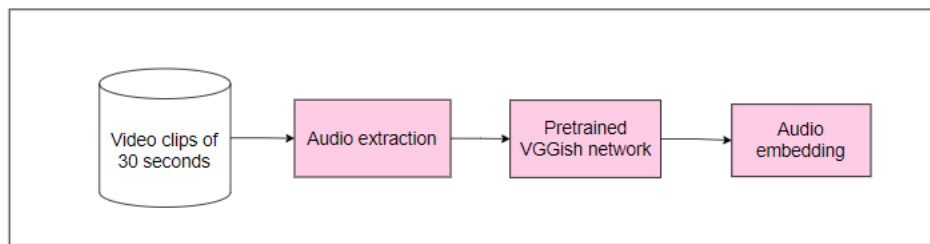


Figure 3.4 Audio Feature Extraction

For each 30 second clippings, the audio from the clip is extracted. The VGGish model which is pretrained on a large corpus of audio clips is loaded. Then each audio is passed into the VGGish network to get the audio embedding.

Since the audio embedding has a dimensionality of (2906,128) for ease of computation, the dimensionlity is reduced to (512,) using UMAP

Frameworks

- (i) Audio Feature Extraction: VGGish Model

Input : The 30 second clips

Output : The audio embedding for each 30 second video clip

Algorithm : Audio feature extraction

initialise model as 'VGGish'

load pretrained weights of the 'model'

for each clip

extract audio of the clip

preprocess the audio to obtain the spectrogram

feed the spectrogram into ‘model’ to receive the audio embeddding

3.5 ACTION SPOTTING

Figure 3.5 shows the design of the module Action spotting

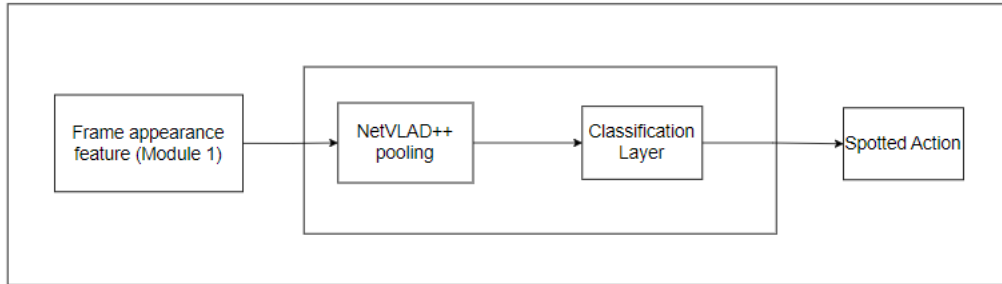


Figure 3.5 Action Spotting

Additionally, the frame appearance features extracted at Module 1 are fed to a NetVLAD++ pooling layer to effectively identify temporal features surrounding the past and future contexts of a frame. The output from this pooling layer is fed to a soft-max classification layer to predict the action present in the video clip.

Input: Frame appearance features from module 1

Output: Label of the soccer action spotted in the clip

Frameworks:

- (i) **Action spotting:** NetVLAD++ Pooling

Algorithm: Action Spotting

Temporal feature extraction

initialise model as ‘NetVLAD’

for each clip

obtain center frame of the clip

obtain NetVLAD features from last layer of the model for the frames before the center frame

obtain the same features for the frames after the center frame

combine them both to create the temporally pooled feature of the clip

Action spotting

for each clip

obtain the frame appearance features of the clip

obtain the temporally pooled features of the clip

feed these features to a softmax layer to obtain the classified action

3.6 CAPTION GENERATION

Figure 3.6 shows the module design of caption generation

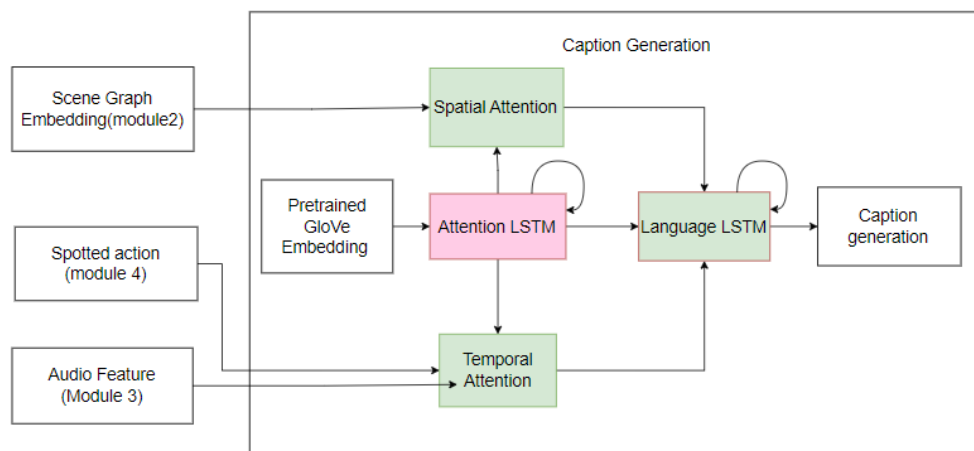


Figure 3.6 Caption Generation

Input: Video Embedding of the object-scene graph, classification output and motion features

Output: Caption of the particular clip

3.6.1 CAPTIONING MODULE

The captioning module consists of an encoder-decoder network consisting of LSTMs and attention layers. The model has 4 layers

- An Attention LSTM
- Temporal attention layer
- Spatial attention layer
- Language LSTM

3.6.2 ATTENTION LSTM

The attention LSTM does the primary encoding of the video sequence and produces a globally aware hidden state at time step t .

Inputs: Mean-pooled video feature, the hidden state of language LSTM in the previous time step, the corresponding word to be generated in the previous time step and the temporal features consisting of NetVLAD++ pooling features and the VGGish Audio features.

Output: hidden state of Attention LSTM at time t

3.6.3 TEMPORAL ATTENTION LAYER

The temporal attention layer produces a global context vector that encodes the temporal awareness of the video at timestep t . Temporal awareness highlights the frame(time step) when a significant action had occurred. The layer produces

temporal weights for each frame denoting the temporal importance contributed by each frame at time t .

Inputs: Video features, hidden state of Attention LSTM

Outputs: Temporal weights (with dimension as number of Samples \times timesteps \times frames) and Global context vector (Linear product of the temporal weights and features of the corresponding frames)

3.6.4 SPATIAL ATTENTION LAYER

The spatial attention layer finds the local spatial regions that are important at time step t . After the temporal attention layer, the objects' features of various frames are linearly multiplied with the temporal weights to produce a local frame feature. Spatial attention layer works on this feature to produce spatial weights and the local context vector.

Input: Locally aligned object features from the scene-graph embeddings, hidden state of attention lstm

Output: Spatial weights (a tensor with dimension as number of samples \times timeSteps \times number of objects) and Local context vector (Linear product of spatial weights and local frame feature)

3.6.5 LANGUAGE LSTM

The Language LSTM corresponds to the language modelling part of the captioning model. The hierarchical-decoder takes in the previously computed features as input and predicts the word for each timestep t .

Input: Hidden state of attention LSTM, local context vector, global context vector

Output: Word at time t

These words are accumulated for all time steps t and are converted back to words following a mapping from the embedding space to the word space.

Algorithm: Caption Generation

initialize the attention model with pre-trained object detector and scene graph generator, and the language LSTM with a pre-trained word embedding model.

pad the captions with a random word to contain 30 words each

convert the caption text into a sequence of word embeddings using the pre-trained Glove word embedding model.

for each time step:

pass the pooled frame appearance features the current word embedding through the attention LSTM.

Update the hidden state of attention LSTM

Pass the obtain hidden state, appearance features of the current frame along with the temporally pooled features to temporal attention layer

Obtain the temporal attention weights and compute the global context vector as the dot product of the obtained weights and the video features

Create the local frame object descriptors by weight-pooling object features from graph embeddings with temporal attention weights

Pass the local descriptors to the spatial attention layer to obtain the spatial attention weights

Create the local context vector as the dot product of spatial attention weights and object descriptors.

Pass the global context vector, local context vector and the hidden state of attention LSTM to language LSTM

Pass the output of the LSTM to a SoftMax layer to obtain the word embedding of the current generated word

Obtain its equivalent actual word through inverse conversion in Glove

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 DATASET

We used the SoccerNet-v2, one of the largest corpora of its kind for our project. It is a large-scale corpus of soccer videos containing around 500 hours of untrimmed matches and over 30,000 manual annotations for various tasks like action spotting, player tracking. The action spotting labels from the dataset would be used for the scope of this project.

17 classes of actions are labelled in the dataset [“ball out of play”, “throw in”, “foul”, “indirect free kick”, “clearance”, “shot off target”, “corner”, “substitution”, “kick off”, “yellow card”, “red card”, “off side”, “direct free kick”, “goal”, “penalty”, “yellow then red card”, “whistle”].

We used the 12 single-camera games for experimentation. The 12 games contained annotated actions from 7 classes along with its accompanying captions. The available data was split into two subsets, each for training and testing. We used the testing subset to evaluate the performance of our model.

4.2 EXPERIMENTAL SETUP

The experimental setup involved utilizing Google Colab as the primary programming environment. The project relied on essential tools such as NumPy and Spacy toolkit for data preprocessing and linguistic analysis. PyTorch was used for implementing and training the deep learning models required for soccer captioning, ensuring accurate and descriptive results

4.3 IMPLEMENTATION AND RESULTS

4.3.1 FRAME FEATURE EXTRACTION

Initially, for the ease of processing, the complete match video is clipped into videos of 30 seconds as shown in Figure 4.1 and Figure 4.2. The clips extracted from the complete matched video is stored in files as shown in the Figure 4.3.

```
from moviepy.video.io.VideoFileClip import VideoFileClip
import math

def cut_video_into_clips(video_file_path, clip_length):
    video = VideoFileClip(video_file_path)
    video_duration = video.duration
    clip_count = math.ceil(video_duration / clip_length)

    for i in range(clip_count):
        start = i * clip_length
        end = min(video_duration, (i+1) * clip_length)
        clip = video.subclip(start, end)
        filename = video_file_path[:-4] + "clip" +str((i+1))+".mkv"
        print(filename)
        clip.write_videofile(filename, codec='libx264')

    video.close()

for file in os.listdir(r'D:\Soccernet\england_epl\'):
    if file.endswith('.mkv'):
        path = "D:\\Soccernet\\england_epl\\" + file
        cut_video_into_clips(path, 30)
```

Figure 4.1 Video Clipping

```
(opencv) C:\Users\Akshaya>python C:\Users\Akshaya\FYP\VideoClipper.py
D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip1.mkv
Moviepy - Building video D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip1.mkv.
MoviePy - Writing audio in 2_224pclip1TEMP_MPY_wvf_snd.mp3
MoviePy - Done.
Moviepy - Writing video D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip1.mkv
Moviepy - Done !
Moviepy - video ready D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip1.mkv
D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip2.mkv
Moviepy - Building video D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip2.mkv.
MoviePy - Writing audio in 2_224pclip2TEMP_MPY_wvf_snd.mp3
MoviePy - Done.
Moviepy - Writing video D:\Soccernet\england_epl\2014-2015\2015-02-21 - 18-00 Chelsea 1 - 1 Burnley\2_224pclip2.mkv
```

Figure 4.2 Output of Video clipping

1_224p	19-02-2023 11:45	MKV File	1,68,828 KB
1_224pclip1	26-02-2023 01:27	MKV File	1,706 KB
1_224pclip2	26-02-2023 01:27	MKV File	1,751 KB
1_224pclip3	26-02-2023 01:27	MKV File	1,924 KB
1_224pclip4	26-02-2023 01:27	MKV File	1,884 KB
1_224pclip5	26-02-2023 01:28	MKV File	1,928 KB
1_224pclip6	26-02-2023 01:28	MKV File	1,811 KB
1_224pclip7	26-02-2023 01:28	MKV File	1,787 KB
1_224pclip8	26-02-2023 01:28	MKV File	1,770 KB
1_224pclip9	26-02-2023 01:28	MKV File	1,830 KB

Figure 4.3 Extracted 30s video clips

For each 30 second clipping, 60 frames are extracted and frame appearance features are extracted using InceptionResnetV2 as shown in Figure 4.4. The features extracted for a single 30s clipping and its corresponding dimension are shown in the Figure 4.5 and Figure 4.6.

```

17 def extract_features(image):
18     # convert the image pixels to a numpy array
19     image = img_to_array(image)
20     # reshape data for the model
21     image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
22     # prepare the image for the Inception Model
23     image = preprocess_input(image)
24     # get extracted features
25     features = model.predict(image)
26     print(features.shape)
27     # save to file
28     return features

```

Figure 4.4 Implementation of frame feature extraction

Output

```
✓ 43s 1 onVideo("/content/1_224pcip1.mkv")
↳ /content/1_224pcip1.mkv
1/1 [=====] - 5s 5s/step
(1, 1536)
1/1 [=====] - 0s 391ms/step
(1, 1536)
1/1 [=====] - 0s 389ms/step
(1, 1536)
1/1 [=====] - 0s 378ms/step
(1, 1536)
1/1 [=====] - 0s 405ms/step
(1, 1536)
1/1 [=====] - 0s 398ms/step
(1, 1536)
1/1 [=====] - 0s 393ms/step
(1, 1536)
```

Figure 4.5 Execution of frame feature extraction

```
✓ 0s [18] 1 features = np.load('/content/1_224pcip1.npy')
      2 print(features.shape)

      (63, 1, 1536)

✓ 0s 1 features
↳ array([[[[0.13326928, 0.18053521, 0.28421617, ..., 0.22675547,
            0.11818172, 0.03394912]],

           [[0.25255817, 0.15852663, 0.28003427, ..., 0.21126975,
            0.10078876, 0.01582667]],

           [[0.23672116, 0.09376782, 0.29847506, ..., 0.34679002,
            0.12155506, 0.05004011]]],

          ...])
```

Figure 4.6 Obtained frame features

4.3.2 OBJECT-SCENE GRAPH CONSTRUCTION

a. Object detection and instance segmentation

The frames are fed to an instance segmentor to get the bounding boxes of objects in it. The detected objects from a sample frame is shown in Figure

4.9. Object-appearance features are also extracted corresponding to the object boundary boxes through a pretrained-2DCNN. Here, a MaskRCNN model with ResNet101 backbone is used to detect objects and obtain their masks. The implementation of the segmentor model is given in Figure 4.7. The driver code to feed each video into the model is shown in Figure 4.8.

```
class Detector:
    def __init__(self, model_type = "OD"):
        self.cfg = get_cfg()
        self.model_type = model_type
        if model_type == "OD":
            self.cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"))
            self.cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")

        elif model_type == "IS":
            self.cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml"))
            self.cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml")

        elif model_type == "LVIS":
            self.cfg.merge_from_file(model_zoo.get_config_file("LVISv0-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml"))
            self.cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("LVISv0-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml")

        elif model_type == "PS":
            self.cfg.merge_from_file(model_zoo.get_config_file("COCO-PanopticSegmentation/panoptic_fpn_R_101_3x.yaml"))
            self.cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-PanopticSegmentation/panoptic_fpn_R_101_3x.yaml")

        self.cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
        self.cfg.MODEL.DEVICE = "cpu"

        self.predictor = DefaultPredictor(self.cfg)
        self.backbone = build_backbone(self.cfg)
        self.pooler = ROIPooler(output_size = 1, pooler_scales = [1,1],
                                sampling_ratio=self.cfg.MODEL.ROI_DENSEPOSE_HEAD.POOLER_SAMPLING_RATIO,
                                pooler_type = self.cfg.MODEL.ROI_DENSEPOSE_HEAD.POOLER_TYPE)
        self.model = build_model(self.cfg)
        DetectionCheckpointinter(self.model).load(self.cfg.MODEL.WEIGHTS)
        self.model.densepose_pooler = ROIPooler()
        self.model.eval()
```

Figure 4.7 Implementation of object detection and instance segmentation

```

beforeCNN.py  ISL_data.py  SoccerNetDownloader.py  FrameFeatureExtractor.py  JsonReader.py
def onVideo(self, videoPath):
    cap = cv2.VideoCapture(videoPath)
    print(videoPath)
    if (cap.isOpened() == False):
        print("Error opening file")
        return
    i = 7
    frame_skip = 12
    frame_count = 0
    frame_id = 0
    (success, image) = cap.read()
    arr = []
    while success:
        if i > frame_skip - 1:
            if self.model_type != "PS":
                image = cv2.resize(image, (120, 120))
                predictions = self.predictor(image)
                attrs = createNodeAttr(predictions, image)
                for attr in attrs:
                    arr.append(attr)
                print(predictions["instances"][0]._fields["pred_boxes"])
                viz = Visualizer(image[:,:,:-1], metadata = MetadataCatalog.get(self))
                output = viz.draw_instance_predictions(predictions["instances"].to("cpu"), se

            else:
                predictions, segmentInfo = self.predictor(image)["panoptic_seg"]
                viz = Visualizer(image[:,:,:-1], metadata = MetadataCatalog.get(self))
                output = viz.draw_panoptic_seg_predictions(predictions.to("cpu"), se

                cv2.imshow("Result", output.get_image()[:,:,:-1])
                i = 0
                frame_count += 1
                key = cv2.waitKey(1) & 0xFF
                if key == ord("q"):
                    break
                i += 1
                (success, image) = cap.read()
    print(frame_count)
    filePath = videoPath[:-4] + ".csv"
    savetxt(filePath, np.array(arr), delimiter=',')

```

Figure 4.8 Driver code for object detection

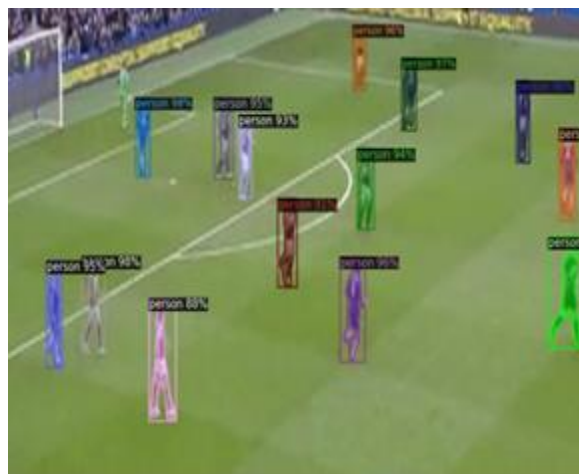


Figure 4.9 Object detection and Instance segmentation done on a sample frame

The object appearance features of each of the detected object is obtained from the FC7 layer of the MaskRCNN model. The obtained $1024 * 1024$ feature matrix is dimensionally reduced by PCA. Its scene correlation features are obtained by converting the whole image into sub-parts of size $40 * 40$ and obtaining the mean value of each of such segment. It is obtained as detailed in Figure 4.10 to give the features stored in files as shown in Figure 4.11.

```
def Centralise(seg):
    seg = seg.reshape(120, 120)
    a = seg[0:40, 0:40].flatten()
    b = seg[0:40, 3:80].flatten()
    c = seg[0:40, 6:120].flatten()
    d = seg[40:80, 0:40].flatten()
    e = seg[40:80, 3:80].flatten()
    f = seg[40:80, 6:120].flatten()
    g = seg[80:120, 0:40].flatten()
    h = seg[80:120, 3:80].flatten()
    i = seg[80:120, 6:120].flatten()
    arr = [np.mean(a), np.mean(b), np.mean(c), np.mean(d), np.mean(e), np.mean(f), np.mean(g), np.mean(h), np.mean(i)]
    return arr

def createNodeAttr(predictions, image):
    items = predictions["instances"]
    attrs = []
    extra = 15 - len(predictions["instances"])
    for i in range(len(predictions["instances"])):
        obj = predictions["instances"][i]
        centers = obj._fields["pred_boxes"].get_centers().numpy()[0]
        x = centers[0]
        y = centers[1]
        x1, y1, x2, y2 = int(obj._fields["pred_boxes"].tensor.numpy()[0][0]), int(obj._fields["pred_boxes"].tensor.numpy()[0][1]), int(obj._
        object_image = image[x1:x2, y1:y2]
        object_image = cv2.resize(object_image, (1024, 1024))
        object_tensor = torchvision.transforms.functional.to_tensor(object_image)
        fc7_features = model.roi_heads.box_head.fc7(object_tensor)
        fc7 = np.mean(fc7_features.detach().numpy(), axis = 0).reshape(1024, 1024)
        pca = PCA(n_components=1)
        feat = pca.fit_transform(fc7)
        print(np.array(feat).shape)

    seg = obj._fields["pred_masks"].numpy().astype("int")
    seg = Centralise(seg)
    attr = []
    attr.append(x)
    attr.append(y)
    for i in seg:
        attr.append(i)
    for i in feat:
        attr.append(i)
    attr = np.array(attr)
    # (attr)
    attrs.append(attr)
    . . .
```

Figure 4.10 Object Appearance feature extraction

Output





 1_224pclip1	26-02-2023 10:35	Microsoft Excel Co...	24,135 KB
 1_224pclip2	26-02-2023 11:37	Microsoft Excel Co...	24,140 KB
 1_224pclip3	26-02-2023 12:38	Microsoft Excel Co...	24,263 KB
 1_224pclip4	26-02-2023 13:43	Microsoft Excel Co...	24,107 KB

Figure 4.11 Output for appearance feature extraction

Where each file contains the node attributes for 25 objects detected at each frame. A sample of the features obtained is shown in Figure 4.12.

[illegible]

Figure 4.12 Extracted appearance features

b. Scene-Graph Construction

The object scene graph is then constructed using the centroids of the bounding boxes as the position of object nodes and the acquired appearance and scene correlation features as their attributes. Nodes that are nearer to each other are connected by edge. The implementation of the graph construction is shown in Figure 4.13. An illustration of a sample scene and its corresponding scene-graph is shown in Figure 4.14.

```

1 def isEdge( a, b ):
2     dist = np.linalg.norm(np.array((a[0], a[1])) - np.array((b[0], b[1])))
3     if(int(dist) <= 20):
4         return True
5     else:
6         return False
7 index = 0
8 #G1.add_nodes_from(node)
9 def addNode (G1, node):
10     for index, row in node.iterrows():
11         G1.add_nodes_from([
12             (index, {"features" : row})
13         ])
14     return G1

```

```

1 def addEdge(G1, node):
2     s = node.iloc[0].name
3     e = node.iloc[0].name + 15
4     for i in range(14):
5         for j in range(i+1, 15):
6             if(isEdge(node.iloc[i], node.iloc[j])):
7                 G1.add_edge(s + i, s + j)
8     return G1

```

```

1 graphs = []
2 frames = len(node1.index)//15
3 for i in range(frames):
4     start = i * 15
5     end = (i+1) * 15
6     G1 = nx.Graph()
7     node = node1.iloc[start:end]
8     G1 = addNode(G1, node)
9     G1 = addEdge(G1, node)
10    graphs.append(G1)

```

Figure 4.13 Implementation of graph construction

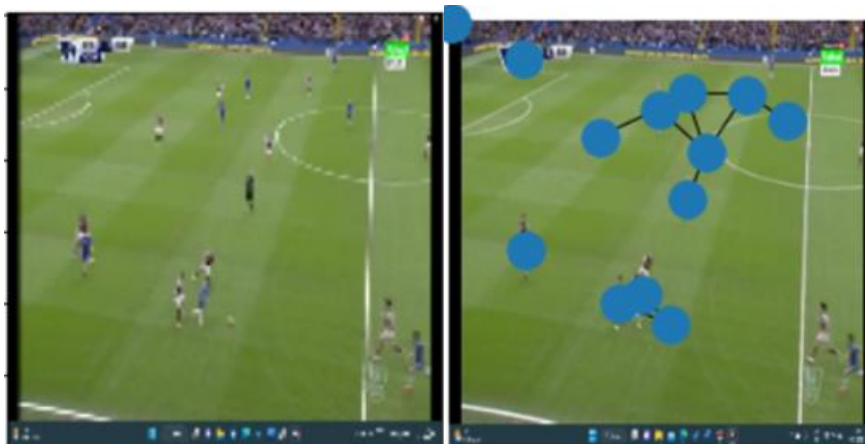


Figure 4.14 A Sample scene and its constructed scene graph

The constructed scene graphs are then fed into a GCN and trained. Figure shows the implementation of the model along with its training loss graph shown in Figure 4.15 and Figure 4.16. The trained model is then used to obtain the embeddings of each node. Figure 4.17 shows the node embeddings and its dimension obtained for a single frame of a video clip.

```

1 emb = []
2 for G in sgraphs:
3     fullbatch_generator = FullBatchNodeGenerator(G, sparse=False)
4     gc_n_model = GCN(layer_sizes=[512], activations=["relu"], generator=fullbatch_generator)
5
6     corrupted_generator = CorruptedGenerator(fullbatch_generator)
7     gen = corrupted_generator.flow(G.nodes())
8     infomax = DeepGraphInfomax(gc_n_model, corrupted_generator)
9     x_in, x_out = infomax.in_out_tensors()
10
11     model = Model(inputs=x_in, outputs=x_out)
12     model.compile(loss=tf.nn.sigmoid_cross_entropy_with_logits, optimizer=Adam(lr=1e-3))
13     epochs = 100
14     es = EarlyStopping(monitor="loss", min_delta=0, patience=10)
15     history = model.fit(gen, epochs=epochs, verbose=0, callbacks=[es])
16     plot_history(history)
17     x_emb_in, x_emb_out = gc_n_model.in_out_tensors()
18
19 # for full batch models, squeeze out the batch dim (which is 1)
20 x_out = tf.squeeze(x_emb_out, axis=0)
21 emb_model = Model(inputs=x_emb_in, outputs=x_out)
22 train_gen = fullbatch_generator.flow(G.nodes())
23 embeddings = emb_model.predict(train_gen)
24 emb.append(embeddings)

```

Figure 4.15 Implementation of GCN

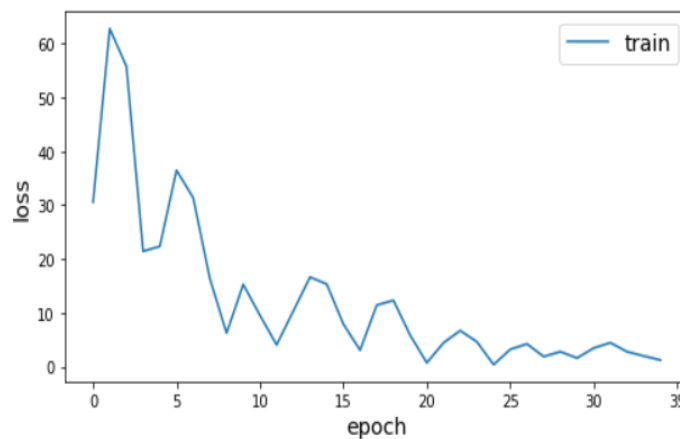


Figure 4.16 GCN training

```
[19] 1 embeddings.shape
(15, 512)

1 embeddings
array([[1.4529854, 2.727112, 0.00569063, ..., 0., 3.1542618,
0. ],
[2.6991212, 4.042334, 0., ..., 1.8349307, 3.5021293,
0. ],
[2.2310362, 4.0169144, 1.3888812, ..., 0., 5.105372,
0. ],
...,
[0., 0., 0., ..., 0.0093656, 0.,
0. ],
[0., 0., 0., ..., 0.0093656, 0.,
0. ],
[0., 0., 0., ..., 0.0093656, 0.,
0. ]], dtype=float32)
```

Figure 4.17 Node embeddings obtained for a single frame of a video clip

4.3.3 AUDIO FEATURE EXTRACTION

This module extracts the audio feature of every clip, which aids in temporal attention. The VGGish model is utilised for audio feature extraction. VGGish is a model for audio feature extraction, specifically for extracting rich and compact representations of audio signals. VGGish is trained on a large dataset of audio clips, and it maps each clip to a compact 128-dimensional embedding, which captures the significant characteristics of the audio. Figure 4.18 and Figure 4.19 show the implementation of VGGish model for the mentioned purpose. The output of the model is shown in Figure 4.20 and Figure 4.21.


```

def CreateVGGishNetwork(hop_size=0.96): # Hop size is in seconds.
    """Define VGGish model, load the checkpoint, and return a dictionary that points
    to the different tensors defined by the model.
    """
    vggish_slim.define_vggish_slim()
    checkpoint_path = 'vggish_model.ckpt'
    vggish_params.EXAMPLE_HOP_SECONDS = hop_size
    vggish_slim.load_vggish_slim_checkpoint(sess, checkpoint_path)
    features_tensor = sess.graph.get_tensor_by_name(
        vggish_params.INPUT_TENSOR_NAME)
    embedding_tensor = sess.graph.get_tensor_by_name(
        vggish_params.OUTPUT_TENSOR_NAME)
    layers = {
        'conv1': 'vggish/conv1/Relu',
        'pool1': 'vggish/pool1/MaxPool',
        'conv2': 'vggish/conv2/Relu',
        'pool2': 'vggish/pool2/MaxPool',
        'conv3': 'vggish/conv3/conv3_2/Relu',
        'pool3': 'vggish/pool3/MaxPool',
        'conv4': 'vggish/conv4/conv4_2/Relu',
        'pool4': 'vggish/pool4/MaxPool',
        'fc1': 'vggish/fc1/fc1_2/Relu',
        # 'fc2': 'vggish/fc2/Relu',
        'embedding': 'vggish/embedding',
        'features': 'vggish/input_features',
    }
    g = tf.compat.v1.get_default_graph()
    for k in layers:
        layers[k] = g.get_tensor_by_name( layers[k] + ':0')
    return {'features': features_tensor,
            'embedding': embedding_tensor,
            'layers': layers,
    }

```

Figure 4.18 Implementation of VGGish

```

def EmbeddingsFromVGGish(vgg, x, sr):
    """Run the VGGish model, starting with a sound (x) at sample rate
    (sr). Return a dictionary of embeddings from the different layers
    of the model."""
    # Produce a batch of log mel spectrogram examples.
    input_batch = vggish_input.waveform_to_examples(x, sr)
    # print('Log Mel Spectrogram example: ', input_batch[0])
    layer_names = vgg['layers'].keys()
    tensors = [vgg['layers'][k] for k in layer_names]
    results = sess.run(tensors,
                       feed_dict={vgg['features']: input_batch})
    resdict = {}
    for i, k in enumerate(layer_names):
        resdict[k] = results[i]
    return resdict

```

Figure 4.19 Extracting embeddings from VGGish

Output

```
MoviePy - Writing audio in temp.wav
MoviePy - Done.
Original Dimension of VGGish Embedding: (2907, 128)
Dimensionality Reduction using UMAP: (512,)
Saved embedding for 1clip24.mkv
MoviePy - Writing audio in temp.wav
MoviePy - Done.
Original Dimension of VGGish Embedding: (2907, 128)
Dimensionality Reduction using UMAP: (512,)
Saved embedding for 1clip25.mkv
```

Figure 4.20 Output of audio feature extraction









Name	↑
 1clip1.pkl	
 1clip2.pkl	
 1clip3.pkl	
 1clip4.pkl	

Figure 4.21 Audio features in files

Where each pickle file contains the audio embedding corresponding to each 30 second video clipping.

```
Shape of data: (512,)
[5.2936616 2.1408992 4.5012126 6.212135 7.852923 4.574814 4.518768
 4.33571 4.8300433 4.8818583 3.4380035 3.7035277 4.326533 4.90554
 5.6735063 5.873532 3.9104989 5.9612756 5.698144 4.5784492 5.256299
 6.583692 5.2194805 4.205787 4.3912206 3.8809497 5.0264077 4.280067
 4.331485 4.396414 5.0948706 5.507389 5.4075174 7.812728 4.2809396
 5.0979033 6.008837 5.619746 4.9043055 3.6332512 6.2277303 5.80565
 6.662073 4.792535 4.301498 6.066127 5.6928596 3.7555249 5.6505136
 4.2960944 3.5755877 4.847421 6.9183564 5.067169 5.3044753 5.114305
 4.361894 4.9813695 5.0211167 3.2900472 5.013101 4.07981 5.378156
 4.0358458 4.110969 5.161842 4.319124 5.5956016 5.9361157 3.403451]
```

Figure 4.22 Extracted audio features

For each of the extracted 30 second clipping, the audio extraction model extracts feature of dimension (2096, 128). This dimension is then reduced using UMAP

to a lower dimensional space of (512,) as shown in Figure 4.22. Figure 4.23 and Figure 4.24 depicts the audio before and after dimensionality reduction.

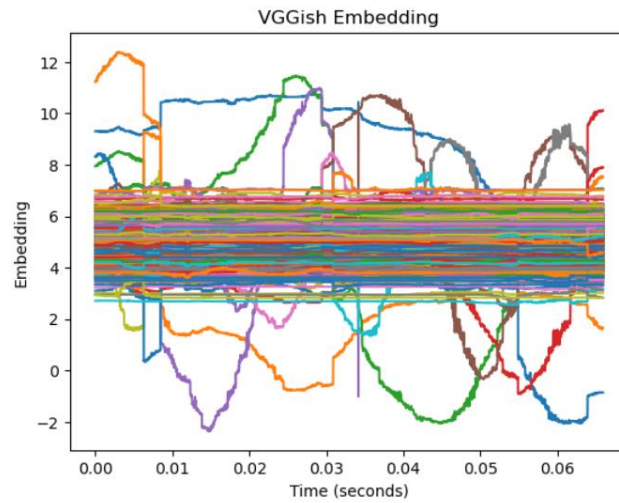


Figure 4.23 VGGish Embedding obtained for the audio of a video clip

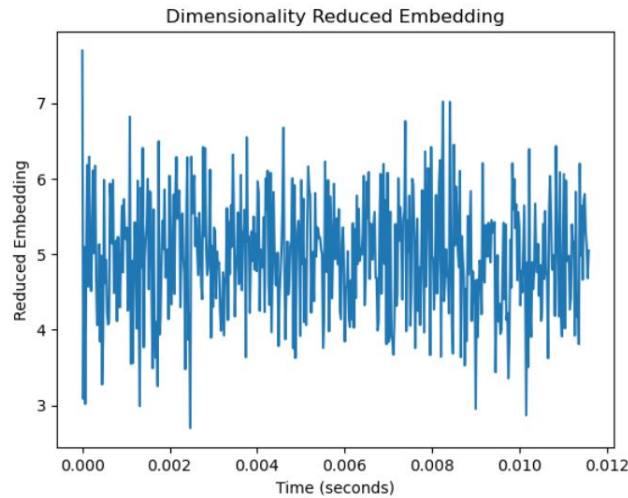


Figure 4.24 Audio Embedding after dimensionality reduction

4.3.4 ACTION SPOTTING

Temporal features and action spotted in each clip are obtained to give temporal significance. The NetVLAD++ pooling module is utilised for temporal pooling. The pooled temporal features are fed into a softmax layer to predict the

action present in a soccer clip. Figure 4.26 shows the baseline architecture of the NetVLAD++ model along with its loss function as shown in Figure 4.25. Figure 4.28 shows the action spotted for two different test data. Since the NetVLAD++ features themselves are sufficient to capture temporal features in the clip, the features from the last layer of this model are extracted to form the temporal features of a particular clip as shown in Figure 4.27. Figure 4.29 shows the temporal uniformity observed in the NetVLAD++ embedding of two different clips containing same action

```
class NLLLoss(torch.nn.Module):
    def __init__(self):
        super(NLLLoss, self).__init__()

    def forward(self, labels, output):
        # return torch.mean(labels * -torch.log(output) + (1 - labels) * -torch.log(1 - output))
        return torch.mean(torch.mean(labels * -torch.log(output) + (1 - labels) * -torch.log(1 - output)))
```

Figure 4.25 Loss function for NetVLAD++ model

```
class Model(nn.Module):
    def __init__(self, weights=None, input_size=512, num_classes=17, vocab_size=64, window_size=15, framerate=2, batch_size = 30):
        """
        INPUT: a Tensor of shape (batch_size,window_size,feature_size)
        OUTPUTS: a Tensor of shape (batch_size,num_classes+1)
        """
        super(Model, self).__init__()

        self.window_size_frame=window_size * framerate
        self.input_size = input_size
        self.num_classes = num_classes
        self.framerate = framerate
        self.vlad_k = vocab_size
        self.fc = nn.Linear(input_size*self.vlad_k, self.num_classes+1)
        self.feature_extractor = None

        if not self.input_size == 512:
            self.feature_extractor = nn.Linear(self.input_size, 512)
            input_size = 512
            self.input_size = 512

        self.pool_layer_before = NetVLAD(cluster_size=int(self.vlad_k/2), feature_size=self.input_size,
                                         add_batch_norm=True)
        self.pool_layer_after = NetVLAD(cluster_size=int(self.vlad_k/2), feature_size=self.input_size,
                                         add_batch_norm=True)

        self.fc = nn.Linear(input_size*self.vlad_k, self.num_classes+1)
        self.drop = nn.Dropout(p=0.4)
        self.sigm = nn.Sigmoid()
        self.weights = self.load_weights(weights)
```

Figure 4.26 Implementation of NetVLAD++ model

```

print("Resulting NetVLAD++ features :")
print(logs.shape)
print(logs)

Resulting NetVLAD++ features :
(262, 512)
[[-0.04178167 -0.03397905 -0.05800464 ...  0.01391139  0.01691547
  0.01567022]
 [-0.00245055  0.01131966 -0.00714697 ...  0.00379008  0.0090063
  0.00670483]
 [-0.03931544 -0.05761556 -0.04399014 ...  0.00464507  0.00567276
  0.0018271 ]
 ...
 [ 0.0444003   0.05385896  0.05924065 ...  0.01772056  0.02022889
  0.01889981]
 [ 0.06529777  0.07528863  0.08492568 ...  0.01349564  0.01547739
  0.01508992]
 [-0.01279522  0.02070657 -0.01420929 ...  0.00344606  0.00633641
  0.00412727]]

```

Figure 4.27 Extracted NetVLAD++ features

```

#Testing for a random Sample
ind = np.random.randint(len(features))

print("Actual : ", le.inverse_transform([int(targ[ind])]))
print("Predicted : ", le.inverse_transform(np.array(res)[ind]))

Actual :  ['na']
Predicted :  ['na']

```

```

#Testing for a random Sample - Faulty case
ind = np.random.randint(len(features))

print("Actual : ", le.inverse_transform([int(targ[ind])]))
print("Predicted : ", le.inverse_transform(np.array(res)[ind]))

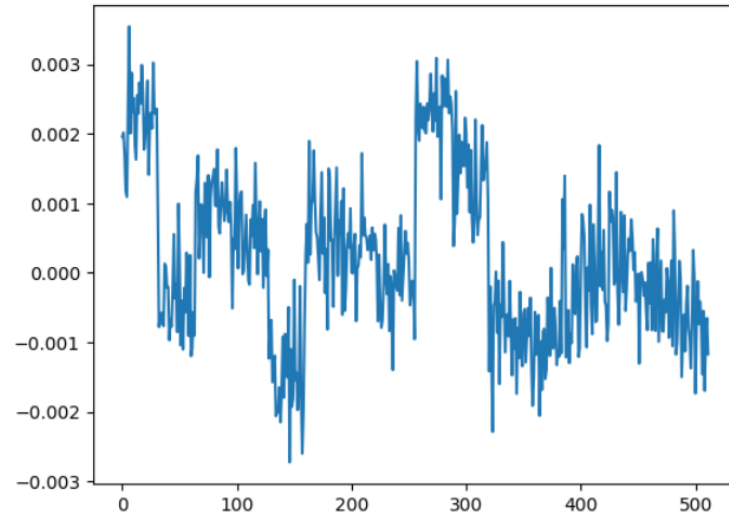
Actual :  ['corner']
Predicted :  ['na']

```

Figure 4.28 Action spotted for two different test data (a correctly and a wrongly predicted instance)

```
import matplotlib.pyplot as plt
plt.plot(logs[18])
action = le.inverse_transform(np.array(int(targ[18])).reshape(1,1))
print("Action : ", action)

/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_label.py:155:
  y = column_or_1d(y, warn=True)
Action : ['y-card']
```



```
import matplotlib.pyplot as plt
plt.plot(logs[10])
action = le.inverse_transform(np.array(int(targ[10])).reshape(1,1))
print("Action : ", action)

/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_label.py:155:
  y = column_or_1d(y, warn=True)
Action : ['y-card']
```

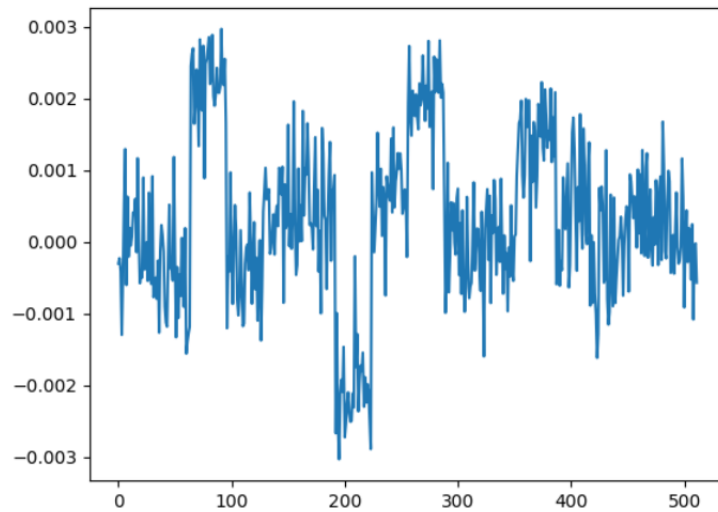


Figure 4.29 Temporal uniformity observed in the NetVLAD++ embedding of two different clips of same action

4.3.5 CAPTION GENERATION

Caption Generation uses a combination of attention LSTM and language LSTM to generate captions. The below mentioned actions are taken in order, before feeding the captions to the model.

a. Text Preprocessing:

Sentences which are longer than 30 words are truncated and all words are converted into lower case. For sentences shorter than 30 words, a zero embedding is padded to fill its content. We build a vocabulary on words with at least 2 occurrences. We embed each of such word to a 300-D word vector initialized with GloVe by spaCy toolkits. Thus, the embedding of a video clip is of dimension (30, 300).

b. Video feature collation

In order to feed to the captioning model, the following video features are collated.

- Video frame appearance features
- Spatial features of the clip (Object-scene graph embeddings)
- Temporal features of the clip (NetVLAD++ and Audio)

c. Temporal Attention Layer

The collated video features are first fed into an attention layer as shown in Figure 4.32, Figure 4.33 and Figure 4.34. The features are then fed into the temporal attention layer whose implementation is shown in Figure 4.30 to capture the temporal importance of each of the frame in a clip.

```

class temp_attention(Layer):
    def __init__(self, return_sequences=True):
        self.return_sequences = return_sequences
        super(temp_attention,self).__init__()

    def build(self, input_shape):
        self.W=self.add_weight(name="att_v_weight", shape=(1536,60), initializer="normal", trainable = True)
        self.U=self.add_weight(name="att_h_weight", shape=(512,60), initializer="normal", trainable = True)
        self.b=self.add_weight(name="att_bias", shape=(input_shape[1],1), initializer="normal", trainable = True)
        super(temp_attention,self).build(input_shape)

    def call(self, x):
        u = x[:, :, :512]
        v = x[:, :, 512:]

        e = K.tanh(K.dot(v,self.W)+K.dot(u,self.U))
        a = K.softmax(e, axis=1)

        context = tf.expand_dims(K.sum(a, axis=2), axis=2)

        output = v*context

        if self.return_sequences:
            return output, a
        return K.sum(output, axis=1), K.sum(a, axis=1)

```

Figure 4.30 Implementation of temporal attention layer

d. Spatial Attention Layer

Followed by temporal attention, the scene-graph embeddings along with the temporal weights from the previous layer is fed to the spatial attention layer whose implementation is shown in Figure 4.31 to capture the spatial importance of each of the objects in a frame.

```

1 #Spatial Attention
2
3 class spat_attention(Layer):
4     def __init__(self, return_sequences=True):
5         self.return_sequences = return_sequences
6         super(spat_attention,self).__init__()
7
8     def build(self, input_shape):
9         self.W=self.add_weight(name="att_v_weight", shape=(1035,1), initializer="normal", trainable = True)
10        self.U=self.add_weight(name="att_h_weight", shape=(512,15), initializer="normal", trainable = True)
11        self.b=self.add_weight(name="att_bias", shape=(input_shape[1],1), initializer="normal", trainable = True)
12        super(spat_attention,self).build(input_shape)
13
14
15    def call(self, u,v):
16
17        temp = K.dot(v,self.W)
18        temp = tf.transpose(temp, perm=[0,2,1])
19        e = K.tanh(temp+K.dot(u,self.U))
20        a = K.softmax(e, axis=0)
21        a = tf.transpose(a, perm=[0,2,1])
22
23        output = a*v
24        op = tf.reduce_mean(output, axis=1)
25        op = tf.expand_dims(op, axis=1)
26        if self.return_sequences:
27            return op
28        return K.sum(op, axis=1)

```

Figure 4.31 Implementation of Spatial attention layer


```

3 #Define the input shape and number of timesteps
4 vid_shape = (60, 1536)
5 net_shape = (1, 512)
6 aud_shape = (1, 512)
7 h_lang_shape = (1, 512)
8 word_shape = (1, 300)
9 local_obj_shape = (60,15,1035)
10
11
12 # Define the LSTM layer
13 attn_lstm_layer = tf.keras.layers.LSTM(units=512, return_sequences=True, return_state=True)
14 t_attn = temp_attention()
15 s_attn = spat_attention()
16 dense_layer = tf.keras.layers.Dense(units=1)
17 lang_lstm_layer = tf.keras.layers.LSTM(units=512, return_sequences=True, return_state=True)
18 softmax = tf.keras.layers.Dense(units=300, activation='softmax')
19
20 # Define the input tensor
21 vid_in = tf.keras.Input(shape=vid_shape)
22 net_in = tf.keras.Input(shape=net_shape)
23 aud_in = tf.keras.Input(shape=aud_shape)
24 h_in = tf.keras.Input(shape = h_lang_shape)
25 word_in = tf.keras.Input(shape = word_shape)
26 local_obj_in = tf.keras.Input(shape=local_obj_shape)
27
28 # Define initial hidden state and cell state tensors
29 initial_h = tf.zeros((NUM_SAMPLES, 512))
30 initial_c = tf.zeros((NUM_SAMPLES, 512))

```

Figure 4.32 The Whole hierarchical encoder-decoder model

```

36 for i in range(30):
37     # Get the input for the current timestep
38     v = vid_in[:, i, :]
39     v = tf.expand_dims(v,axis=1)
40     #print(v.shape)
41     x = tf.keras.layers.concatenate([v, aud_in, net_in, h_in, word_in])
42
43     if(i != 0):
44         x = tf.keras.layers.concatenate([v, aud_in, net_in, tf.expand_dims(state_h, axis=1), output])
45
46     # Pass the input and previous hidden and cell states to the LSTM layer
47     output1, state_h, state_c = attn_lstm_layer(x, initial_state=[initial_h, initial_c])
48
49     #Output 1 - 512
50     print("Out 1 : ", output1.shape)
51
52     conc = tf.keras.layers.concatenate([v, output1])
53     output2, a = t_attn(conc)
54
55     #Output 2 - 300
56     print("Out 2 : ", output2.shape)
57
58     #Calculation here, getting l = a*v across frames; resulting in a 15x200 size tensor
59     #For now lets take one sample alone
60
61     a = tf.squeeze(a, axis=1)
62     a = tf.expand_dims(tf.expand_dims(a, axis=2), axis=3)
63

```

Figure 4.33 Time-step wise implementation of attention mechanisms

```

64 inter = local_obj_in * a
65 l = tf.reduce_mean(inter, axis=1)
66 print(l.shape)
67
68 #l = local_obj_in[:,i,:,:]
69
70 output3 = s_attn(output1, l)
71 print("Out 3 : ", output3.shape)
72
73 input = tf.keras.layers.concatenate([output1, output2, output3])
74 foutput, state_h, state_c = lang_lstm_layer(input)
75
76 output = softmax(foutput)
77
78 # Update the initial hidden and cell states for the next timestep
79 initial_h = state_h
80 initial_c = state_c
81
82 # Append the output for the current timestep to the list of outputs
83 outputs.append(output)
84
85 # Concatenate the outputs for all timesteps into a single tensor
86 outputs = tf.keras.layers.concatenate(outputs, axis=1)
87 # Define the model
88 model = tf.keras.Model(inputs=[vid_in, h_in, aud_in, net_in, word_in, local_obj_in], outputs=outputs)

```

Figure 4.34 Obtaining final model state from the attention layer

e. Compiling and training the model

The model is then trained with the inputs after initialising its loss and other required hyperparameters as shown in Figure 4.35.

```

[31] 1 model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
      2 model.fit([V, h_lang, aud, net_v, T, loc_obj], T[:, :30, :], batch_size=54, epochs=10)

```

Figure 4.35 Compiling and training the model

f. Testing the model

The captions for the video clips are obtained after testing the trained model as shown in Figure 4.36.

```

▶ 1 predicted = model.predict([V, h_lang, aud, net_v, w, loc_obj], batch_size=54)
↳ 1/1 [=====] - 40s 40s/step

[35] 1 print(predicted.shape)
(54, 30, 300)

```

Figure 4.36 Testing the model

Figure 4.37 depicts the captions obtained for a test sample. Figure 4.38 shows the captions generated for 10 random test samples

```
1 results = embeddings_to_captions(predicted)
2 print(results[12])

['goal', 'great', 'box', 'player', 'ball', 'deep', 'box', 'goal', 'goal', 'box', 'box', 'game', 'player', 'box', 'shot', 'left', 'box', 'game',
```

```
1 print(data[12])

goal superb work from player who plays a vital role in the buildup he squares it to player team who beats player with a brilliant shot into the
```

Figure 4.37 Obtained captions for a test sample

```
predicted_sentences

[player team look like he player player player player to take player corner kick ',
'a to has be made player is replaced by player team ',
'referee blows player whistle and the second half starts ',
'player team player player asking for some medical attention with his it to the be of player injury is yet player be player ',
'a player player team an an player player of the referee to an his whistle player we to to score player player ',
'player team has tested player player of referee referee and player into player player for a player late player player scores an in a to free kick for team ',
'player blows his whistle and player team is shown a second yellow card for player foul his an can not be we he has lot of player player an player ',
'a cross player player team player player side of the player player into the player but player to player player to player as the aim was poor ',
'player referee thwarts a to shot towards the we side of referee goal after his we solo run but player thwarts the referee with referee of save the score an ',
'player team an on a player on the edge of the player and player shot goes a whisker over the cross player ball player out player play ',
'penalty save player team sends the penalty towards the player an side player the goal but player put off a we save to an he ']
```

Figure 4.38 Obtained captions

4.4 PERFORMANCE METRICS

We used the following performance metrics to evaluate the model.

(i) Average mAP

Average mAP (Mean Average Precision) is a metric used to measure the performance of object detection and segmentation systems. Here, mAP refers to the average precision per class on action spotting that lies within a tolerance range T averaged over all the classes. [6]

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} A P_K \quad (1)$$

where,

AP_K - Average precision of k classes

n - number of classes

Here, Average mAP is used as a metric for the sub-task action spotting. We achieved an Average-mAP of 72.4% for our sub-task action spotting as shown in Figure 4.39.

```
[ ] map = calculate_map(res, top)
    print("MAP: ", map)

MAP: 0.7241379310344828
```

Figure 4.39 Average mAP

(ii) BLEU

BiLingual Evaluation Understudy measures numerical translation closeness between the ground truth and generated output. Grammatical errors or small changes in word order might get penalty under this metric. This is best suited for small sentences. [12]

$$BLEU = BP \cdot \exp(\sum_{n=1}^N w_n \log p_n) \quad (2)$$

where,

BP - brevity penalty

N - No. of n-grams

w_n - Weight for each modified precision, by default N is 4,
 w_n is $1/4=0.25$

P_n - Modified precision

We obtained a BLEU-4 score of 3.7 with the proposed system as shown in Figure 4.40.

```

from datasets import load_metric
bleu = load_metric("bleu")
b = bleu.compute(predictions=[[results]], references=[[sentences]])
print("Bleu Score : ", b['bleu'])

```

3.7

Figure 4.40 BLEU Score

(iii) METEOR

Metric for evaluation of translation with explicit ordering. The main difference between this and BLEU is that it combines the recall with the precision metric. It utilizes bigrams of words and synonyms. This is similar to bleu but is relaxed from the limitation of strict matching. [2]

$$F_{mean} = \frac{10PR}{R+9P} \quad (3)$$

$$M = F_{mean(1-p)} \quad (4)$$

where,

P - Precision

R - Recall

p - Chunk Penalty

The Meteor score for the proposed approach is 46.9 as shown in Figure 4.41.

```
from datasets import load_metric
meteor = load_metric("meteor")
m = meteor.compute(predictions=[[results]], references=[[sentences]])
print("Meteor : ", m)
```

```
Downloading builder script: 5.34k/? [00:00<00:00, 165kB/s]
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
Meteor: {'meteor': 0.46917513435223673}
```

Figure 4.41 Meteor Score

(iv) Rouge L Score

Rouge is a metric denoting the ability of summarization of a machine. Rouge is calculated as the quotient of number of matched unigrams to the total number of unigrams present. [10]. ROUGE-L measures the longest common subsequence (LCS) between our model output and reference

$$ROUGE - L = \frac{\text{length of the longest common subsequence shared between the reference and candidate}}{\text{number of unigrams in the candidate}} \quad (5)$$

Our approach achieves a Rouge-L score of 39.5 as shown in Figure 4.42.

```
rouge = load_metric("rouge")
rouge.compute(predictions=[[results]], references=[[sentences]])
```

```
Downloading builder script: 5.65k/? [00:00<00:00, 214kB/s]
{'rouge1': AggregateScore(low=Score(precision=0.46296296296297, recall=0.6690454950936664, fmeasure=0.547245530828165), mid=Score(precision=0.46296296296297, recall=0.6690454950936664, fmeasure=0.547245530828165), high=Score(precision=0.46296296296297, recall=0.6690454950936664, fmeasure=0.547245530828165)),
'rouge2': AggregateScore(low=Score(precision=0.2452130945027795, recall=0.35446428571428573, fmeasure=0.289886820007302), mid=Score(precision=0.2452130945027795, recall=0.35446428571428573, fmeasure=0.289886820007302), high=Score(precision=0.2452130945027795, recall=0.35446428571428573, fmeasure=0.289886820007302)),
'rougeL': AggregateScore(low=Score(precision=0.3950617283950617, recall=0.5709188224799286, fmeasure=0.4669828529733674), mid=Score(precision=0.3950617283950617, recall=0.5709188224799286, fmeasure=0.4669828529733674), high=Score(precision=0.3950617283950617, recall=0.5709188224799286, fmeasure=0.4669828529733674)),
'rougeLsum': AggregateScore(low=Score(precision=0.3950617283950617, recall=0.5709188224799286, fmeasure=0.4669828529733674), mid=Score(precision=0.3950617283950617, recall=0.5709188224799286, fmeasure=0.4669828529733674), high=Score(precision=0.3950617283950617, recall=0.5709188224799286, fmeasure=0.4669828529733674))}
```

Figure 4.42 Rouge-L Score

The performance of our model as measured with these metrics are given in Table 4.1.

Table 4.1 Performance measured with the denoted metrics

TASK	METRIC	RESULT
Action Spotting	Average-mAP	0.724
Caption generation	Bleu-4	3.7
Caption generation	Meteor	46.9
Caption generation	Rouge-L	39.5

4.5 COMPARATIVE ANALYSIS

We compare the acquired metrics to two existing systems – one in which a transformer model was utilized along with various visual features such as the high-level object appearance features from the CNN and optical flow features [8]. The other system corresponds to an LSTM decoder model using single anchored dense video captioning, following a spotting and captioning training phases. We specifically compare the method with two different temporal pooling techniques – NetVLAD++ and NETRVLAD [11]. The results are given in Table 4.2 and Figure 4.43.

Table 4.2 Comparison of different approaches

APPROACH	BLEU-4	METEOR	ROUGE-L
VISUAL FEATURES + TRANSFORMER	9.8	-	-
I3D + NETVLAD++	2.9	20.7	18.3
I3D + NETRVLAD	3.1	21.2	20.7
OUR APPROACH	3.7	46.9	39.5

From the results, we found that our method performs better in terms of Meteor and Rouge than the other methods.

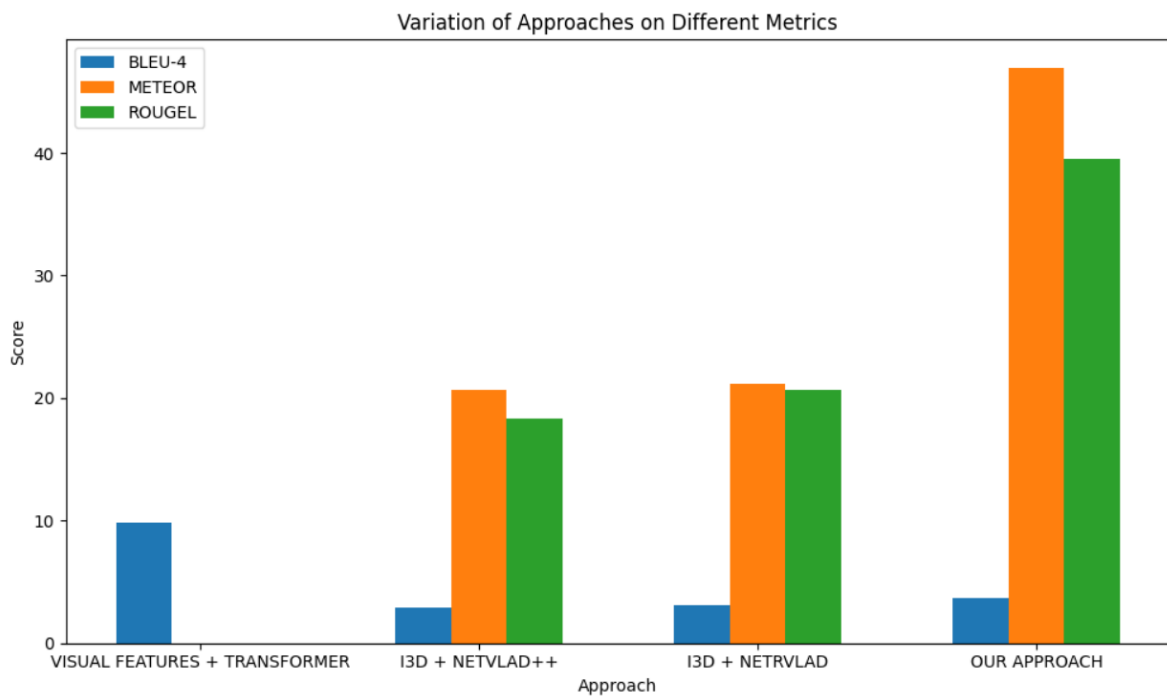


Figure 4.43 Comparison of different approaches

But the proposed method seems to perform low in terms of Bleu. Bleu denotes the frequency of co-occurrence of n-grams and is highly sensitive to positional variance. Since our model produces such n-grams less occasionally, it seems to have been performed low when compared with this metric. However, since the recall rate of the model (as measured with Rouge) appears better, a minimal fine tuning of the model for improved coherence of words can make it perform better.

4.6 TEST CASES AND VALIDATION

The proposed system is experimented with 12 complete single camera games localized with both labelled and non-labeled captions. The metrics were measured on a test subset of this set. Specifically, we chose 11 test cases covering all the mentioned labelled and non-labelled captions.

Labelled captions include captions among 8 classes {corner, substitution, whistle, injury, penalty, y-card, yr-card, penalty-missed}

Non-labelled captions denote the captions describing a sequence without any associated action classes. These captions are annotated with the action ‘na’. The expected and obtained captions for the mentioned test cases are shown in Table 4.3

Table 4.3 Expected and Predicted outputs for each test case

Test Case ID	Expected Output	Received Output	Recalled no. of words
T_01	[PLAYER] ([TEAM]) looks like he will be the one to take the corner kick.	player team look like he player player player player to take player corner kick	9

T_02	A substitution has been made. [PLAYER] is replaced by [PLAYER] ([TEAM]).	a to has be made player is replaced by player team	9
T_03	[REFEREE] blows his whistle and the second half starts.	referee blows player whistle and the starts	6
T_04	[PLAYER] ([TEAM]) is clearly asking for some medical attention with his painful gestures. The extent of his injury is yet to be discovered.	player team player player asking for some medical attention with his it to the be of player injury is yet player be player	12
T_05	A player from [TEAM] pulled an opponent down, forcing the referee to blow his whistle. PENALTY! Wonderful opportunity to score for [TEAM].	a player player team an an player player of the referee to an his whistle player we to to score player player	7
T_06	[PLAYER] ([TEAM]) has tested the patience of referee [REFEREE] and goes into the book for a previous late challenge. Another situation results in a	player team has tested player player of referee referee and player into player player for a player late player player scores	8

	direct free kick for [TEAM].	an in a to free kick for team	
T_07	[REFEREE] blows his whistle and [PLAYER] ([TEAM]) is shown a second yellow card for his foul. His manager will not be pleased. He has lot of time to think about it as he walks off.	player his whistle and player team shown a second yellow for player foul his an can not be we he has lot of player player an player	12
T_08	A cross by [PLAYER] ([TEAM]) from the side of the pitch flies into the box, but fails to find its intended target as the aim was poor.	a cross player player team player player side of the player player into the player but player to player player to player as the aim was poor	11
T_09	[PLAYER] ([TEAM]) unleashes a promising shot towards the right side of the goal after his wonderful solo run, but [PLAYER] thwarts the effort with a fabulous save. The score remains the same. Corner kick. [TEAM] will have an opportunity.	player referee thwarts a to shot towards the we side of referee goal after his we solo run but player thwarts the referee with referee of save the score an	15

T_10	[PLAYER] ([TEAM]) pounced on a rebound on the edge of the box and his shot goes a whisker over the crossbar. The ball goes out of play	player team an on a player on the edge of the player and player shot goes a whisker over the cross player ball player out player play	19
T_11	Penalty saved! [PLAYER] ([TEAM]) sends the penalty towards the left hand side of the goal, but [PLAYER] pulls off a wonderful save to deny him.	penalty save player team sends the penalty towards the player an side player the goal but player put off a we save to an he	16

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In our project, we presented a novel approach to automatically generate captions for soccer matches using object-scene relational graph and spatial-temporal LSTM. The proposed system leverages object detection, and spatial-temporal attention to generate more accurate and informative captions. We evaluated our approach on benchmark datasets and demonstrated that it outperforms existing captioning techniques, achieving state-of-the-art results. Our outcomes indicate that the proposed method can effectively understand the activities that occur and provide richer information of the whole scene and meaningful captions.

5.2 FUTURE WORK

While our proposed system shows promising results, there are several areas for future research.

One challenging aspect with the approach was the processing time it took for training and testing each sample. Future approaches can be focused on improving the response time of the model. The state-of-art transformer models can also be experimented in the place of captioning model. Fine-tuning approaches for improving the grammatical correctness of the generated sentences, possibly through Large-Language Models (LLM) can be incorporated.

Overall, we believe that our approach represents a significant step forward in the development of automatic captioning systems for sports matches, and we look forward to exploring these avenues for future research.

REFERENCES

- [1] Amaresh, M., & Chitrakala, S. (2019, April). Video captioning using deep learning: An overview of methods, datasets and metrics. In *2019 International Conference on Communication and Signal Processing (ICCSP)* (pp. 0656-0661). IEEE.
- [2] Banerjee, S., & Lavie, A. (2005, June). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization* (pp. 65-72).
- [3] Cartas, A., Ballester, C., & Haro, G. (2022, October). A Graph-Based Method for Soccer Action Spotting Using Unsupervised Player Classification. In *Proceedings of the 5th International ACM Workshop on Multimedia Content Analysis in Sports* (pp. 93-102).
- [4] Deliege, A., Cioppa, A., Giancola, S., Seikavandi, M. J., Dueholm, J. V., Nasrollahi, K., ... & Van Droogenbroeck, M. (2021). Soccernet-v2: A dataset and benchmarks for holistic understanding of broadcast soccer videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4508-4519).
- [5] Gao, L., Wang, X., Song, J., & Liu, Y. (2020). Fused GRU with semantic-temporal attention for video captioning. *Neurocomputing*, 395, 222-228.
- [6] Giancola, S., Amine, M., Dghaily, T., & Ghanem, B. (2018). Soccernet: A scalable dataset for action spotting in soccer videos. In *Proceedings of the IEEE*

conference on computer vision and pattern recognition workshops (pp. 1711-1721).

[7] Giancola, S., & Ghanem, B. (2021). Temporally-aware feature pooling for action spotting in soccer broadcasts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4490-4499).

[8] Hammoudeh, A., Vanderplaetse, B., & Dupont, S. (2022). Soccer captioning: dataset, transformer-based model, and triple-level evaluation. *Procedia Computer Science*, 210, 104-111.

[9] Hua, X., Wang, X., Rui, T., Shao, F., & Wang, D. (2022). Adversarial Reinforcement Learning With Object-Scene Relational Graph for Video Captioning. *IEEE Transactions on Image Processing*, 31, 2004-2016.

[10] Lin, C. Y. (2004, July). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out* (pp. 74-81).

[11] Mkhallati, H., Cioppa, A., Giancola, S., Ghanem, B., & Van Droogenbroeck, M. (2023). SoccerNet-Caption: Dense Video Captioning for Soccer Broadcasts Commentaries. *arXiv preprint arXiv:2304.04565*.

[12] Nabati, M., & Behrad, A. (2020). Multi-sentence video captioning using content-oriented beam searching and multi-stage refining algorithm. *Information Processing & Management*, 57(6), 102302.

[13] Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (pp. 311-

318).

[14] Qi, M., Wang, Y., Li, A., & Luo, J. (2019). Sports video captioning via attentive motion representation and group relationship modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(8), 2617-2633.

[15] Tu, Y., Zhou, C., Guo, J., Li, H., Gao, S., & Yu, Z. (2023). Relation-aware attention for video captioning via graph learning. *Pattern Recognition*, 136, 109204.

[16] Wang, B., Ma, L., Zhang, W., & Liu, W. (2018). Reconstruction network for video captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7622-7631).

[17] Wu, A., Han, Y., Yang, Y., Hu, Q., & Wu, F. (2019). Convolutional reconstruction-to-sequence for video captioning. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(11), 4299-4308.

[18] Yan, Y., Zhuang, N., Zhang, J., Xu, M., Zhang, Q., ZHENG, Z., ... & Zhang, W. (2019). Fine-grained video captioning via graph-based multi-granularity interaction learning. *IEEE transactions on pattern analysis and machine intelligence*.

[19] Zhou, X., Kang, L., Cheng, Z., He, B., & Xin, J. (2021). Feature Combination Meets Attention: Baidu Soccer Embeddings and Transformer based Temporal Detection. *arXiv preprint arXiv:2106.14447*.