Automatic Highlights Generation for Cricket Matches

Alina Sarwar

1. Introduction

Cricket is the second most watched sports game in the world. It is no surprise that many people, despite being cricket lovers, are unable to keep up with the live matches because of their busy routines and prefer to watch the highlights instead. However, even the shortest format of cricket i.e. T20, tends to have ~3-4 hours long matches. This means that manually producing highlights for such games is a time-consuming and cumbersome task. An efficient method for generating these highlights automatically would not only save manpower but also make sure that the highlights become available for the viewers to watch quickly after the match ends.

In this project, we aim to implement an approach introduced in a recent paper by Agarwal et al [1] for automatically generating cricket highlights. The proposed method reformulates the problem of generating cricket highlights as simply annotating each and every ball of the game in terms of whether an important event (eg. a four/six, a wicket fall etc) occurred during that ball. It works by segmenting a full match video into individual ball clips using a self-trained Convolutional Neural Network (CNN) to detect the start of the ball and Optical Character Recognition (OCR) for detecting the changes in the scorecard and signalling the end of that ball. Once every ball has been tagged as “interesting” enough or not, the process of generating highlights is then as simple as stitching all such relevant ball clips together into a highlights package.

We observe that the proposed framework produces promising results for four full-length T20 match innings, both in terms of precision and recall. The produced highlights are able to capture the majority of the high-scoring balls of the innings as well as the fall of wickets. It is important to note here that this methodology is more suited to shorter cricket formats like T20 because we are looking to process every ball of the game and include all fours/sixes in the highlights. Therefore, for

longer matches like those of one-day or test cricket, the processing cost will be very high and highlights produced using this method might end up being very long.

2. Structure of a T20 Cricket Match

T20 is just a truncated cricket format in which each team's innings are limited to 20 overs (an over comprises six balls delivered by a bowler to the striking batsman who defends his wicket). During an innings, all members of the fielding team but only two members of the batting team are on the field at any given time. On each delivery, the two batsmen may run across the pitch and switch places to score singles. Additionally, four or six runs are scored if a ball hits beyond the boundary around the cricket field. A team's innings ends if 10 of the 11 team members are dismissed and the team who scores more runs wins the match.

3. Dataset and Methods

We have collected a dataset of 10 complete T20 match videos (a mix of both international and league games). Out of these, 8 videos are used to train the CNN and the remaining 2 are used to evaluate our final model that generates the highlights. The methodology we use for extracting highlights from a given match is summarised in the figure below:

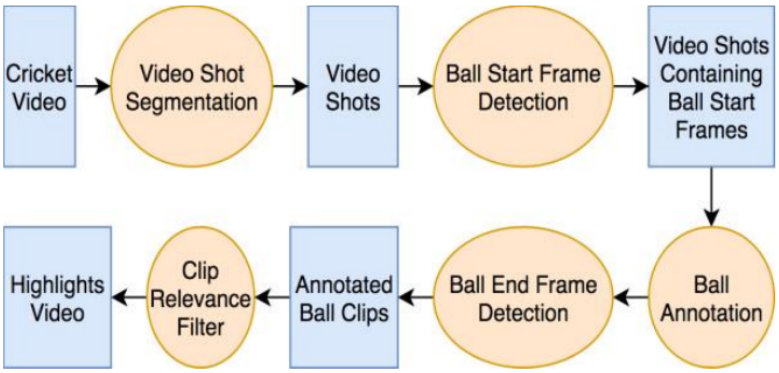

Figure 1: Architecture of the proposed approach

In the first phase of this project, we worked on the first two steps of the proposed methodology for highlights generation: **1) video shot segmentation,** in which full

match videos were divided into separate scenes, each represented by three key frames (marking the start, middle and end of the scene) along with their timestamps, and **2) ball start frame detection**, where we trained a binary classifier for detecting all the ball start frames amongst the key frames in a match video. The remaining steps of the methodology were implemented in the second phase of the project and are described in detail below.

3.1 Ball End Frame Detection using OCR

To fetch the complete clip of any given ball, we need to know its start and end time. Start time information can be extracted easily for each key frame detected as a 'ball start frame' by the CNN classifier trained in Phase 1. In order to determine the ball end time, we use the Optical Character Recognition (OCR) technique to read the score box region of the subsequent match frames. Initially we implemented OCR using the Tesseract [2] library in Python, as done by the authors of the referenced paper [1]. However, later on, we shifted to the more state-of-the-art PaddlePaddle[1] OCR as we found its recognition results to be more accurate.

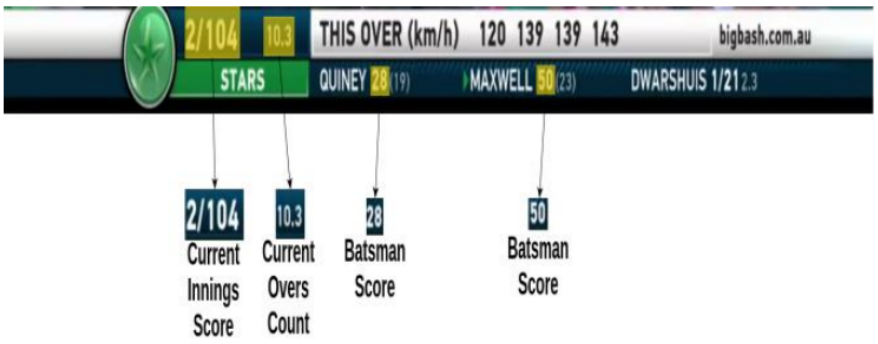

Figure 2. Typical cricket score bar showing all the regions relevant for highlights generation

The score box is typically displayed at the bottom of our screens as an overlay. Since different match broadcasters use different score box designs, we need to demarcate all regions of interest manually before any

[1] https://github.com/PaddlePaddle/PaddleOCR

video processing begins. Once marked, these positions stay constant throughout the match. The overarching aim is to be able to read all the relevant statistics from the score box into a machine readable text format so that we can track how exactly the game is progressing. More specifically, to detect a ball end frame, we are looking to detect a change in the over count region of the score box. As soon as it changes, we conclude that the ball has finished and save the timestamp of the corresponding key frame. In this manner, the complete match video is segmented into multiple ball clips. Each ball clip has its own metadata including the innings score and the over count value at the end of that ball.

Each of the four relevant regions of the score box are first cropped out from the full frame. It is then enlarged and the sharpness level is increased for better readability. Finally, the crops are converted into black and white images before feeding them to the OCR.

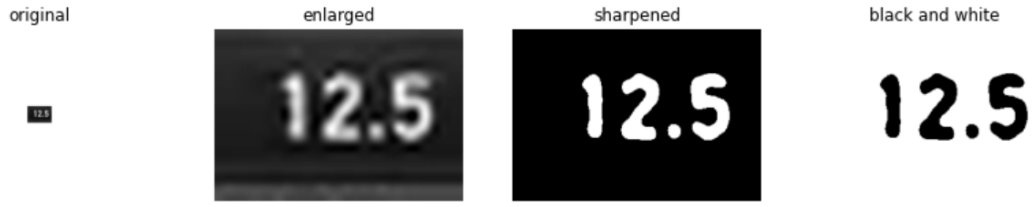

Figure 3. Preparation of input for OCR by applying a series of transformations

Figure 4 below shows the results of running OCR on a sample image. Each of the four relevant regions of the scorebox are preprocessed and then successfully read into a simple text format.

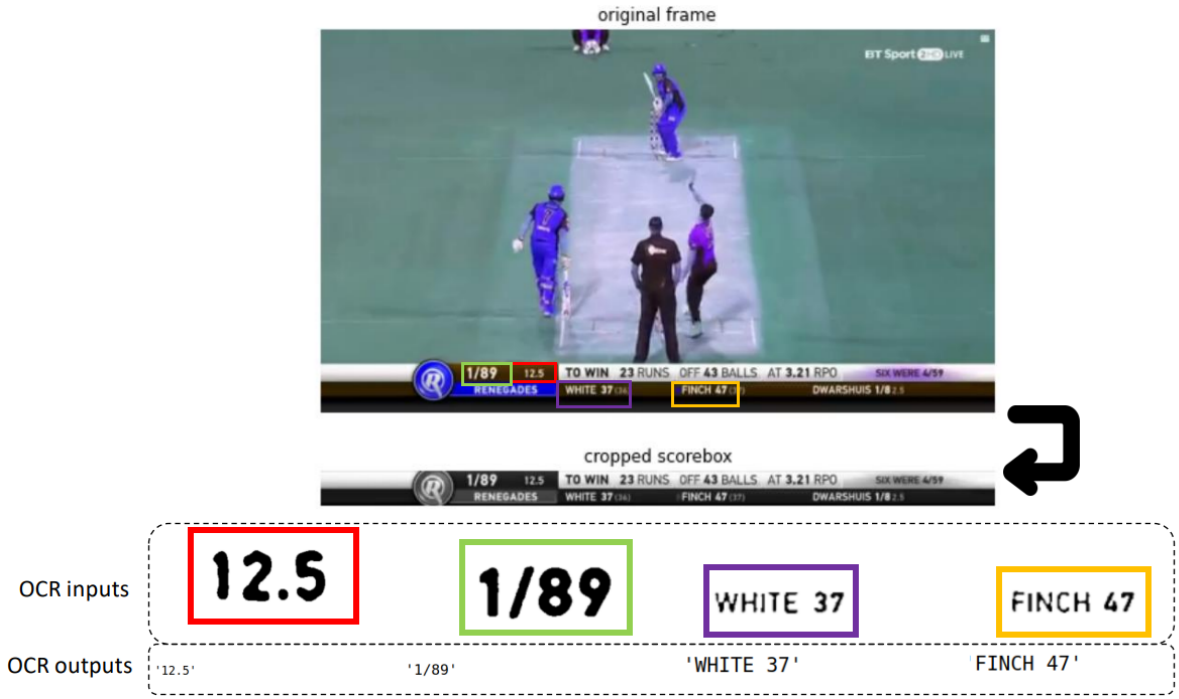

Figure 4. Sample OCR results

3.2 Ball Clip Annotation

Once we have all the ball clips, we annotate them by comparing their metadata. The current innings score consists of the number of wickets lost by the batting team and current score. The number of runs scored on a given ball is calculated by subtracting the score count at the end of the previous ball from the current score count. Similarly, a wicket fall is detected by calculating the difference between the number of wickets on the previous ball and the number of wickets on the current ball.

3.3 Relevant Clips Filtration and Final Highlights Generation

Once the whole match video has been annotated, we generate a highlights clip of the cricket match using the following criteria for a ball to be included in the package:

a) Four or more runs were scored on that ball

b) The wicket fell on that ball

c) The ball is the last of the innings

All such ball clips are then stitched together to generate the final highlights clip using the Python library MoviePy[2].

4. Results and Discussion

We evaluate our final model on two T20 matches, each comprising two innings. Just like there was a lot of manual effort involved in generating the required dataset for training our model, performing its evaluation was an equally tiring task. During the whole debugging and results evaluation phase, we actually had to watch the generated highlights each time to see which clips were included and whether or not they satisfy the criteria given in section 3.3.

Since the output of our final model i.e. the generated highlights package, has more of a qualitative nature whose usefulness may vary from person to person, choosing the right method of evaluation was a bit tricky. Some people may prefer to watch each and every boundary of the game as part of the highlights while others might just want to see the most defining moments of the match and wicket falls included in the highlights. We did conduct a user study and found out that our generated highlights were generally approved by the viewers. However, the scope of this study was limited given the time constraints involved in this project. Furthermore, this kind of analysis is bound to be subjective and tends to undermine the quality of our evaluation by making it sound vague or less definitive.

Hence, we try to quantitatively determine the goodness of our model by reporting the precision and recall of our highlights. It is important to note here that the authors of the original paper [1] instead used the Intersection over Union (IoU) metric to compare their highlights with the ones provided by the official broadcaster. However, we did not have sufficient time to watch the official highlights and extract its statistics for comparison. Also, as of now, we were more interested in determining the accuracy of our implementation (given the aforementioned criteria) rather than gauging how similar our highlights are to a

[2] https://zulko.github.io/moviepy/

standard highlights package. Therefore, it makes more sense to compare our highlights' statistics with the actual innings rather than the official broadcaster's highlights.

	ACTUAL INNINGS					OUR HIGHLIGHTS					
Innings	Length (mins)	Sixes	Fours	Wickets	Last ball	Length (mins)	Sixes	Fours	Wickets	Last ball	False Positives
1	104	1	8	8	1	7	1	7	8	1	1
2	69	2	12	2	1	6	2	10	1	1	4
3	87	3	16	6	1	8	3	15	6	1	3
4	89	1	7	8	1	6	1	5	7	1	3

Figure 5. Comparison of actual innings statistics Vs generated highlights.

Innings	Precision	Recall	Processing time (mins)
1	0.89	0.94	13
2	0.78	0.82	9
3	0.89	0.96	10
4	0.82	0.82	10
AVERAGE	0.85	0.89	10.5

Figure 6. Evaluation metrics for the final highlighters generator. The precision of the model depends on the number of false positives in the highlights package whereas the recall is determined by the number of relevant events in the actual innings that were missing in our highlights.

The videos are processed at the original 720p display resolution in order for the OCR to give accurate results. Using a lower quality video reduces the processing time but increases the error rate of the OCR. The mean precision score for the 4 innings is found to be 0.85 while the mean recall score is 0.89. Our model is correctly able to extract most ball clips that involve a score change of ≥ 4 or a wicket fall. However, we observe three major kinds of glitches in our generated highlights:

1. During the course of a ball, the score box overlay suddenly disappears so the OCR is not able to capture the end of the ball. It keeps waiting for the

scorebox to become visible again so that a change in the over count is detected and the end of the ball is marked. Sometimes, between the scorebox's disappearance and reappearance, a ball replay or an advertisement is being shown so all of that also becomes part of the ball clip and hence the highlights.

2. When a delivery is declared as a wide or no ball, the over count does not change. As a penalty for the bowler, the score count of the batting team is increased by 1. In this case, the OCR is not able to detect the end of the ball (because the over count never changed). Only the end of the next (legitimate) ball is detected so in effect, two distinct balls coalesce into one. This problem can be solved if detection of the ball end frame does not solely depend on a change in the over count. Rather, the OCR should also monitor the innings score and mark the end of the ball if that changes. This way, even when the over count will not change in case of a wide or no ball, the change in the innings score will help OCR successfully detect the end of the ball. However, care needs to be taken when using score count to detect ball end frame since in some cases, it might lead to incomplete ball clips. For example, the score count might change in increments when the batsmen are running between the wickets scoring singles but the eventual result of the ball is a four or six. In this case, if a change in the score count is used to mark the end of the ball, then the model will fail to capture the complete ball and the perceived score for the ball will be 1 instead (i.e. the first time the score count changed).

3. An important ball's start frame is not correctly identified by the CNN so that ball has no chance of being included in the highlights. This happens rarely though because our CNN classifier has very high accuracy.

5. Future Work

Given that this project was done single-handedly, significant overall progress was made towards the achievement of objectives originally planned for this project. In

future, more work needs to be done to make our model robust to match video inconsistencies such as scorebox disappearance. Furthermore, we would like to expand the capacity of our model to be able to detect other important milestones in the match such as a batsman scoring a 50 or a century (something we were not able to implement because of limited time). Finally, we would want to evaluate our model more extensively on a higher number of matches so that we can report our result metrics with higher confidence and identify additional loopholes in the proposed framework that need to be overcome.

6. References

1. Agarwal et al, "Automatic Annotation of Events and Highlights Generation of Cricket Match Videos", in *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 2018.
2. R. Smith, “An overview of the tesseract OCR engine,” in *International Conference on Document Analysis and Recognition (ICDAR),* Vol. 2, pp. 629–633, 2007.

Link to Github repository