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, on average. 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.

2. Summary

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 signaling 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.

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.

3. 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.

4. Dataset and Methods

We have collected a dataset of 12 complete T20 match videos (a mix of both international and league games) along with the official broadcaster's highlights. Out of these, 8 videos are used to train the CNN and the remaining 4 will be used to evaluate our final model that generates the highlights. The methodology we use for extracting highlights from a given match is summarized in the figure below:

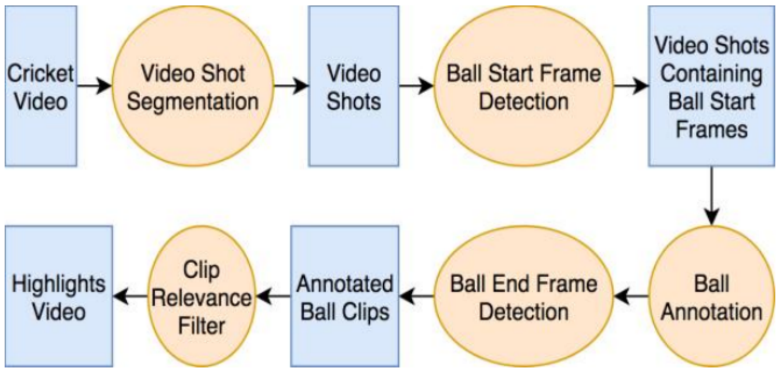


Figure 1: Architecture of the proposed approach

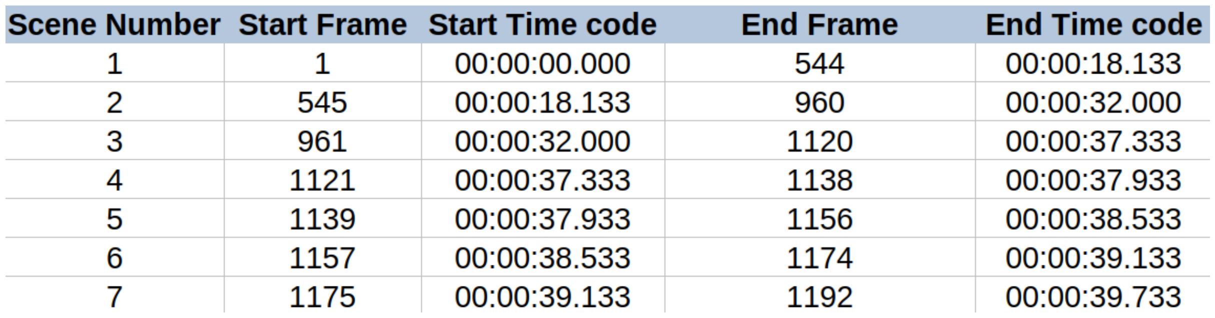
In the first phase of this project, we have worked on the first two steps i.e. video shot segmentation and ball start frame detection are described in detail below.

4.1 Match Video Segmentation

The first step involves breaking down a full match video into separate scenes using an open source python library called PySceneDetect[1]. The *detect-content* module is used to compare the difference in content between adjacent frames against a set threshold, which if exceeded, triggers a scene cut.

[1] http://scenedetect.com/projects/Manual/en/latest/cli/commands.html

Three key frames [2], marking the start, middle and end of the scene, are saved against every distinct scene detected in the match video. This helps us cut down the processing cost as all further steps are only applied on these key frames and not all the frames of the whole match video. We also use the *list-scenes* command of the same package to help us write out the results to a CSV file in the form of a table containing all scenes, their start/end timecodes, and frame numbers.



Scene Number	Start Frame	Start Time code	End Frame	End Time code
1	1	00:00:00.000	544	00:00:18.133
2	545	00:00:18.133	960	00:00:32.000
3	961	00:00:32.000	1120	00:00:37.333
4	1121	00:00:37.333	1138	00:00:37.933
5	1139	00:00:37.933	1156	00:00:38.533
6	1157	00:00:38.533	1174	00:00:39.133
7	1175	00:00:39.133	1192	00:00:39.733

Figure 2: Example csv file containing scene information

4.2 Ball Start Frame Detection

In this step, we train a binary classifier (CNN) that helps us detect all the ball start frames in the match. The challenge here is to generate training samples for the CNN since we need to dig out the ball start frame samples from the match videos ourselves.

4.2.1 Preparation of training dataset

To make our lives a little easier, we make use of image dilations in a semi-supervised approach to generate a pool of potential positive and negative samples for ball-start frames, which we then manually review to prepare our final training set.

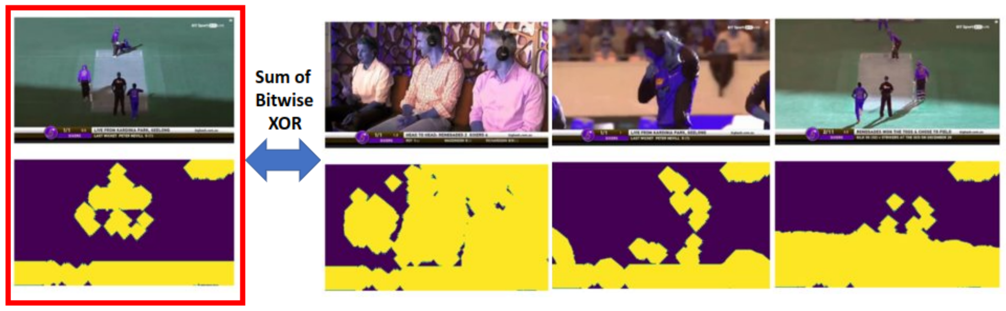


Figure 3: Generating potential positive and negative training samples for ball start frame detection. (Left): Sample ball start frame from a match and its dilated version. (Right): Other key frames in the match along with their dilations.

Dilation[2] is a morphological operator that we apply to an image in order to enhance its primary features and join disparate elements to get a sense of the distribution of foreground objects in the image. Figure 3 (left) shows a typical ball start frame where the batsman, non-striker, bowler, wicket-keeper and umpire are all positioned in a certain way around the pitch. This positioning stays more or less constant across all ball start frames of the match so their dilations should also look similar. Hence, by comparing the dilation of the sample ball start frame with the dilations of other key frames in the match video, we can shortlist a set of frames that are also likely to be ball start frames. The comparison is carried out by calculating frame difference which is defined as the sum of the bitwise XOR of the corresponding pixels of the frames. If this sum is less than a threshold, we collect that image as a potential

[2] https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

positive sample. Rest of the images work as our negative sample set. We then manually review the collected samples and make all the required label corrections before we proceed to training the CNN.

Using this method, we obtained ~1500 ball start frames which we increased to about ~4000 by applying data augmentation techniques such as horizontal flipping and changing brightness/contrast levels. Although the number of negative samples collected is a lot more, we downsample to about ~5000 images in order to avoid class imbalance issues during training.

4.2.2. Model Architecture and Training

The following figure describes in detail the CNN architecture used for ball start frame detection.

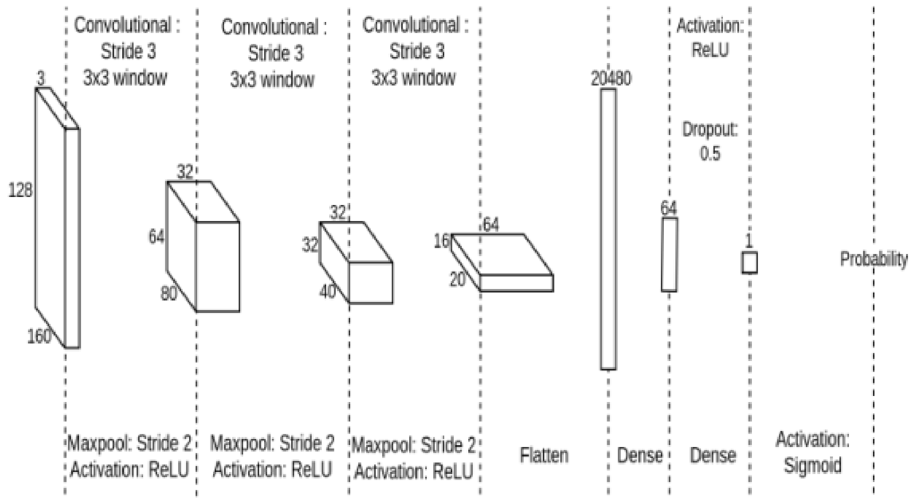

Convolutional : Stride 3 3x3 window
Convolutional : Stride 3 3x3 window
Convolutional : Stride 3 3x3 window
Activation: ReLU
Dropout: 0.5
3
128
160
32
64
80
32
32
40
16
64
20
20480
64
1
Probability
Maxpool: Stride 2 Activation: ReLU
Maxpool: Stride 2 Activation: ReLU
Maxpool: Stride 2 Activation: ReLU
Flatten
Dense
Dense
Activation: Sigmoid

Figure 4. Ball start CNN architecture

The CNN is implemented in PyTorch. As can be seen in the figure above, the final dense layer consists of a single neuron. Application of the sigmoid activation function on this neuron's output gives the probability of the input frame being a start-of-ball frame. The model takes as input individual frames downsized to dimensions 160x128. The frames are kept in RGB and not converted to grayscale. For training (as well as testing), the scorebar in the frames is excluded. A train-validation split of 80:20 is used with stratified sampling to ensure equal class proportions in both sets. The loss function used is Binary Cross Entropy along with a learning rate of 1e-4.

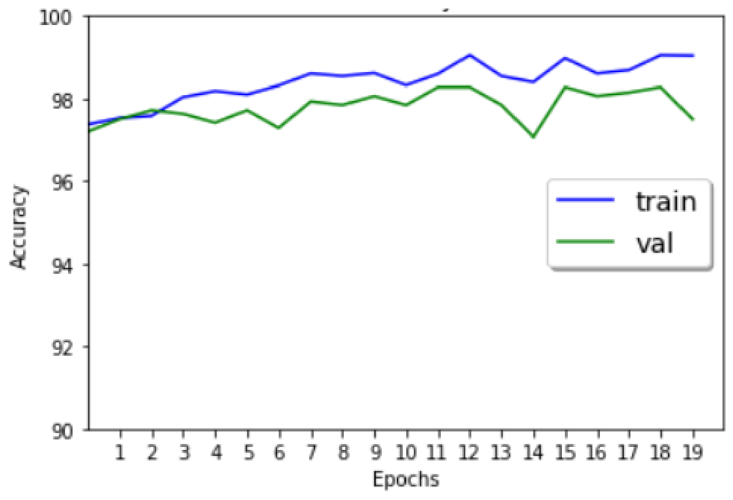


Figure 5. Training and validation set accuracies

The model weights are saved when the best validation accuracy is achieved around epoch 15.

5. Results and Discussion

The trained ball start frame classifier achieves an accuracy of 98.25% on a test set of 458 images. Predictions contributing to the loss of accuracy are visualized in the figure below.

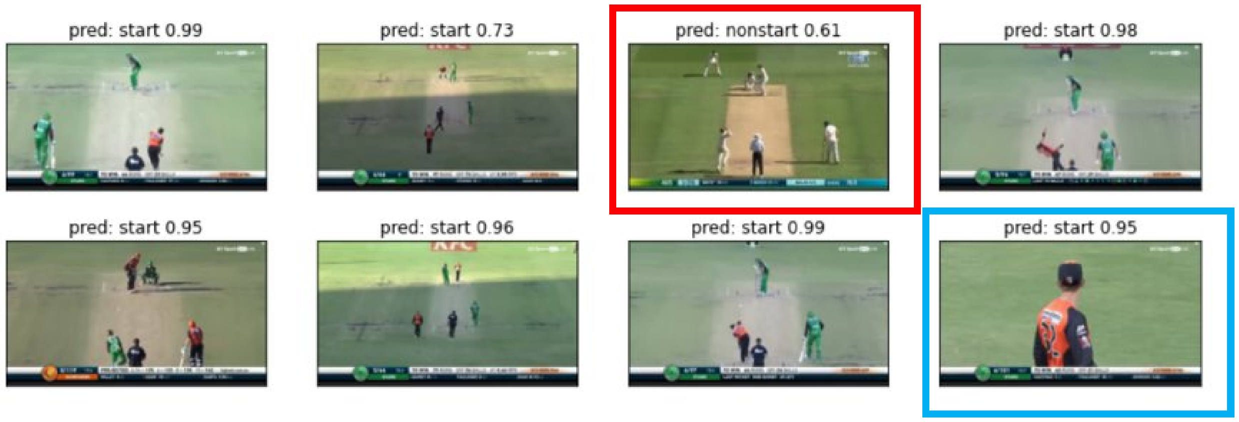


Figure 6. Apparently incorrect predictions on the test set

It turns out that 6 of these 8 predictions (the unboxed ones) are actually correct; these frames are indeed ball-start frames but their true labels are incorrect. The two predictions highlighted are incorrect though. For the blue one, it turns out that some similar kind of frames in the training set were wrongly labeled as ball-start frames, hence the model has probably learned some wrong features for this class. Once these labeling errors are corrected, such mispredictions will most likely vanish.

The more concerning misprediction is the red one (a false negative). The model predicts this frame to be a non-ball start frame with 0.61 confidence although this clearly is a ball start. One plausible explanation for the model getting confused here is that in this frame, all the players are wearing white kits (as in a test match) and this does not happen in a T20 match where both teams usually wear colored kits. It will be a good idea to rerun the training, this time switching color channels as part of data augmentation or training with only gray-scale images. This will help our model become color agnostic in its prediction and might eliminate such incorrect predictions.

Nevertheless, in the broader scheme of the highlights generation problem, even if our classifier misses a few balls, it is most likely not going to affect the quality of our final highlights because not every ball of the match is supposed to be included in the highlights anyway.

6. Future Work

Most of the milestones of the first project phase were met. I just hoped to be able to start the OCR implementation in this phase but it turned out that collecting training data for the ball start frame CNN took a lot more time than I had expected. Still, the pace of the project is going well and I am positive that I will be able to finish the project on time.

The remaining steps of the proposed methodology for highlights generation are

a) Ball End Frame Detection using OCR

b) Ball Clip Annotation

c) Relevant Clips Filtration and Final Highlights Generation using MoviePy[3].

These will be implemented in the next phase of the project and then we will evaluate our complete highlights generation model by conducting user studies as well as comparing our highlights with those provided by the official broadcaster using the Intersection over Union (IoU) metric.

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

7. 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] W. Wolf, “Key frame selection by motion analysis,” in Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on, vol. 2. IEEE, 1996, pp. 1228–1231.

8. Appendix

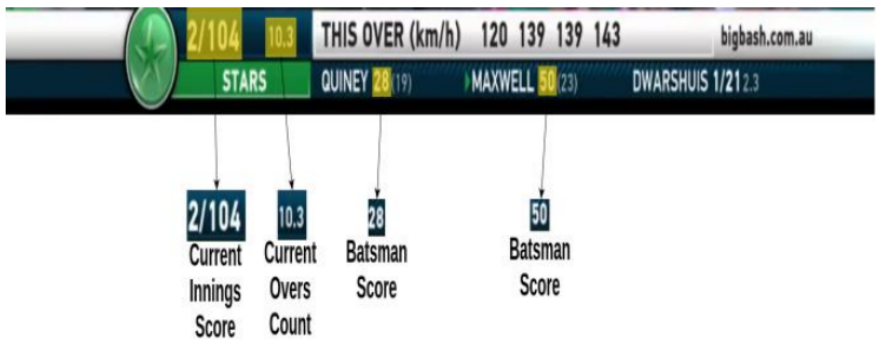


Figure 7. Typical cricket scorebar showing all the regions relevant for highlights generation

Link to Github repository