Deep Learning using CNNs for Ball-by-Ball Outcome Classification in Sports

Kalpit Dixit Stanford University

kalpit@stanford.edu

Anusha Balakrishnan Stanford University

anusha@cs.stanford.edu

Abstract

In this paper, we compare the performance of three different Convolutional Neural Network architectures, inspired by literature on activity recognition in videos, on the novel task of classifying ball-by-ball outcomes for sports videos, specifically for cricket. In the sport of cricket, each ball can have one of four different outcomes, and we define our classification task as the prediction of this outcome, given a sequence of frames representing the video for that ball. We report the performance of each of these architectures, and explore their advantages and disadvantages for this domain. Our best model is able to achieve an accuracy of almost 80% on our validation set.

1. Introduction

For nearly a hundred years now, we have had access to live video streaming of Sports events, first through television networks and now over the Internet. In this domain, verbal and written commentary have remained important, since they provide a dense expert interpretation of what is happening in the game, both to viewers and to individuals who might be following the game without access to a video stream. A natural outcome of this is that there are good and bad commentators, and people have individual tastes on what kind of commentary they want: light-hearted vs. technical, and descriptive vs. predictive. This is reflected in individual opinions, like "I really like that analysis", or "That commentator ruins the game experience for me". Thus, providing each user with commentary of their liking is an important and open problem.

At present, commentary is often delivered live at the event, and then transcribed almost in real-time and uploaded to websites like ESPN.com. As a result, the problem of automatically generating dense expert interpretations is largely unexplored. A preliminary step in this direction is the classification of the outcome of a particular play. For example, in the game of cricket, this involves classifying the ball's outcome as a "boundary" or a "wicket" without

getting into the details of where the bowler bowled the ball, what shot the batsman attempted, which part of field the ball went to, etc. Another example is tennis, where we would first like to learn how to simply classify who won a particular point without getting into the details of how good the service was, or what shots were played in the rally.

It is not immediately clear how easy (or not) outcome classification of single plays in sports is. In this project, we address this problem, focusing specifically on cricket. The input to our algorithm is a video clip of a single ball being played out in Cricket. We explore three different deep learning architectures to output a classification of the input ball video into one of four classes: {"no_run", "runs", "boundary", "wicket"}.

2. Related Work

Our work is on understanding what happens in a video, or in other words, understanding the meaning of a sequence of images (the frames of the video). We did not find any work which attempts to classify sports outcomes. In light of that, we look at the related areas of Dense Image Captioning, Sports Recognition and Action Recognition.

In [1], Karpathy and Li, Fei-Fei train a CNN followed by an RNN for dense captioning of images (sentences describing several details of an image). Donahue *et al.* in [2] used a Long-Term Convolutional Recurrent Network to train networks for activity recognition, image description and video description.

Interestingly, most of the works on action recognition in videos feature both a novel architecture and a novel data augmentation method. Highlighting the shortage of labelled videos for Deep Learning. Ji *et al.* introduced the 3D ConvNet for Human Action Recognition in videos in [3]. Simonyan and Zisserman in [4] use multi-task learning which gives them access to multiple datasets; they train a ConvNet on dense multi frame optical-flow for recognizing human sports movements. Wang *et al.* in [5] used a Depth Map [6] representation to train a 3D ConvNet for Human Action Recognition; they rotated the original depth data to mimic camera rotation and hence use the rotated depth maps for data augmentation. Geng and Song in [7] used

an auto-encoder to pre-train their CNN model and then used a SVM classifier to recognise Human Sports Actions. Karpathy *et al.* in [8] combine CNNs to develop Fusion architectures for classifying videos into kinds of sports.

Past work with an objective closest to ours is by Karpathy *et al.* in [8] and indeed amongst the past works, the architectures in [8] are a large source of motivation for this work. We use Late Fusion (as defined in [8]) as a baseline and compare its performane on this task against two other architectures. The first is a single-frame classification architecture, inspired by the single-frame analysis in [8], and the second is a Long-Term Recurrent Convolutional Network (as described in [2]). All three architectures are described in more detail below in Sections 3.4, 3.5, and 3.6.

3. Methods

The input to our model is a video of F frames where each frame is of size 224x224x3. Thus our input dimensionality is 224x224x3xF, which we map to just four output dimensions, one score each for the four possible outcome classes. However, in order to use backpropagation to train the weights, we need a single number to represent the *loss value*. i.e. we need a loss function. For this, we use the Softmax function to convert the individual class scores into class probabilities and then use the Cross Entropy Loss function to combine the class probabilities and the correct class label into a single loss value.

3.1. Softmax Function

Consider a classification problem with C classes. Given an input example and an algorithm, the algorithm produces raw scores for each class, $s_1, s_2...s_C$. The only interpretation of these scores is that a higher score s_i indicates that the algorithm thinks that the input example's likelihood of belonging to class i is higher. The Softmax function translates these raw scores into $softmax\ scores$ which are all in [0,1) and sum up to 1. This has two uses: First, it allows the softmax scores to be interpreted as class probabilities. Second, it allows us to use the Cross Entropy Loss function which requires inputs that define a probability distribution:

Softmax Score:
$$ss_i = \frac{e^{s_i}}{\sum_i e^{s_j}} \forall i \text{ in } \{1, 2...C\}$$
 (1)

3.2. Cross Entropy Loss

The Cross Entropy Loss function comes from Information Theory [9], where it is used to estimate how well one probability distribution describes another probability distribution. Given two two probability distributions over C classes, $\{p_1, p_2...p_C\}$ and

 $\{q_1, q_2...q_C\}$:

Cross Entropy Loss (p,q) =
$$-\sum_{j=1}^{j=C} p_j \ln(q_j)$$
 (2)

In our case, the distribution p for each example is the ground truth for that example. The ground truth distribution for an example i with true class c_i , is a vector of length C with all entries equal to 0 except for entry number c_i which is set to 1. Our softmax output will be the distribution q. It can be shown that given a fixed distribution p, the distribution p that minimizes the Cross Entropy Loss must be equal to p.

3.3. Transfer Learning - Fine-tuning and learning layers from scratch

As has been seen over the past four years in ILSVRC Challenges, Deep CNN architectures have outperformed other methods on tasks related to Images [10]. Karpathy et al. in [8] and Ng et al. in [11] showed that Deep CNN/RNN architectures outperform other methods in classifying sports videos into different sports. While there is no work on classifying sports videos into outcomes, there is overwhelming evidence to suggest that Deep CNN/RNN architectures would outperform other methods on the task, due to its similarity to the task of activity recognition. Since most Deep architectures contain tens of millions of parameters, they require a lot of training data (hundreds of thousands of examples) and a lot of compute power (weeks of GPU time) to train well. We have a major shortage on data, with less than a thousand training examples, and on compute power, with not enough time to complete multiple weeks of training on a GPU. Thus, we turn to Transfer Learning.

Transfer Learning refers to using a model trained on one dataset on another, somewhat similar, dataset. It entails copying not only the architecture of the Network but also the pre-trained values of the weights. Intuitively since images in nature follow some general statistics, training a network on one dataset of natural images should make the network work decently well with other datasets of natural images. Concretely, training from scratch, without any Transfer Learning, entails initializing all the weights close to zero and then training all the weights with a relatively large learning rate. On the other hand, training with Transfer Learning entails initializing all the pre-final layer weights to the pre-trained values and then training just the last few layers with a relatively low learning rate. [12] explains how to decide the number of layers to fine-tune.

We use the pre-trained VGG16Net [13] for our experiments by initializing all layers of the VGG16Net with pretrained weights obtained from the Caffe Model Zoo. When training models, we leave the weights of the

convolutional layers unchanged, and report results from finetuning (or training from scratch) just the last fully connected layer, the last two fully connected layers and all three fully connected layers.

3.4. Single Frame Classification

Single Frame classification is possibly the simplest Deep Learning method for working with videos. It treats each video as a set of images (frames) and runs models meant for image classification on the individual frames. Thus the problem of classifying videos into classes devolves into classifying images into classes. This leads to a big decrease in input dimension and allows us to use the much more extensive work done on Image Classification (compared to Video Classification). Additionally, methods other than Single Frame Classification typically involve more parameters, which become difficult to train well with a limited dataset (as in our case). Single Frame Classification has been shown to be a very strong baseline to other more complex architectures [8].

As in Figure 1, in this work we go a step further by performing Single Frame Averaging. In this method, we first perform end-to-end Single Frame Classification on several frames, including transforming the raw scores for each image into softmax probabilities. We then average the class probabilities over all processed frames for a single clip to get the final class probabilities for that clip. Note that since the individual frames generate a probability over the classes, the average over all frames will also be a probability measure over the classes.

We explore three different configurations of the single frame averaging architecture. In all three configurations, we train the output layer (fc_8) from scratch, but we vary the number of fully connected layers of the VGG16 net that are finetuned. We report results from training only the output layer, finetuning fc_7 , and finetuning fc_7 and fc_6 .

3.5. Late Fusion

This is a method explored by Karpathy *et al.* in [8]. We don't consider their other models such as Early Fusion and Slow Fusion since that would require training the entire network from scratch and not allow us to use transfer learning from the VGG16Net.

Late Fusion, as in Figure 2, runs single-frame networks separately on the first and last frame of the video and then combines the outputs of the last convolutional layer along the 'channel' dimension. Hence, the first fully-connected layer now receives twice as many input activations as the first fully-connected layer in the original VGG16Net. Note that other number of activations in the inputs/outputs of the fully-connected layers remain unchanged. The intuition behind this architecture is that while neither of the two convolutional layers can individually capture motion,

their combination in the first fully-connected layer would allow the fully-connected layers to learn global features of motion.

3.6. LSTMs

In [2], Donahue et al. propose the use of a Long-Term Recurrent Convolutional Network (LRCN) for tasks that require the synthesis of temporal information, like activity recognition and video captioning. Since our classification task requires an "understanding" of motion and visual change over time, we explore the use of this model for our purposes. In our adaptation of the LRCN, the input at each timestep (each frame of a single ball) is passed through the convolutional layers of the VGG16Net and through its fully connected layers to obtain a vector representation ϕ_t of the frame. This representation is then passed as input to an LSTM (Long Short Term Memory), a type of recurrent neural network that is capable of capturing long-term dependencies in sequential data. The LSTM has a hidden state and a cell state, which it updates at each timestep using the previous input, hidden state, and timestep. Although LSTMs can be used to produce an output at each timestep (thus producing t outputs for t timesteps or frames), we discard outputs from all timesteps except the very last one. The output from this timestep is passed through a final fully connected layer with a softmax nonlinearity that produces a probability distribution over the four possible classes. This architecture is shown in Figure

At training time, gradients are first backpropagated through the LSTM and then through the fully connected layers of the VGG16Net. As with the other architectures described in this paper, we only experiment with finetuning the fully connected layers of the VGG16Net. Thus, the parameters for the LSTM are initialized randomly and learnt from scratch, while the parameters for the fully connected layers fc_6 and fc_7 are initialized using the pretrained VGG16Net weights and then finetuned at training time.

4. Dataset and Features

4.1. Videos

Our dataset comprises of video clips of balls in cricket, where each clip corresponds to a single ball and is associated with one of four labels: {"no_run", "runs", "boundary", "wicket"}. Each label represents the outcome of that ball. In order to obtain the clips, we downloaded the full-length videos of 2 games of cricket from the Pepsi Indian Premier League (IPL) tournament in 2014 (a professional cricketing tournament that takes place every year). The games were both of the "T20" format, meaning that each game comprises ≈ 240 balls. We then manually

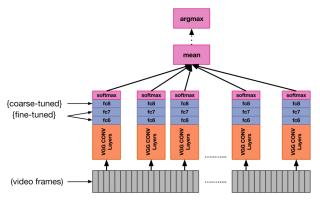


Figure 1. Single Frame Averaging

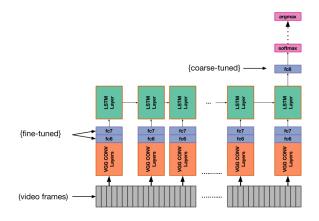


Figure 3. Long-Term Recurrent Convolutional Network

segmented these videos into balls, taking care to remove any visual elements that could inadvertently be used by the learning algorithm, like audience and player reactions after each ball.

4.2. Labels

The labeled outcomes for each ball were obtained by scraping the sports website ESPNcricinfo.com [14], which contains commentary archives of games from all past IPL tournaments. The commentary for each game contains the ball-by-ball outcomes as well as text commentary, and we used the latter as labels for our data. The number of different possible outcomes in cricket is high, and so we made some simplifying assumptions in order to gain as many examples as possible for each class. First, we ignored "wides" and "no balls" (there are 9 in our dataset), which are essentially a penalty for bowling a ball illegally. Although these balls almost always grant one or more "runs" to the batting team, they are visually extremely different from runs won by the team on a legal ball. As a result, we refrain from training our model on them.

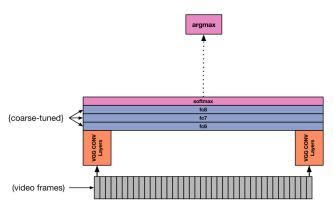


Figure 2. Late Fusion

Second, our "runs" class contains any (legal) balls where the batting team scores one or more runs (but not a boundary). Although it is semantically possible to break this class up into two more classes, "1 run" and "2 runs" (a third class of "3 runs" exists but is very rare in general), the task of distinguishing between "1 run" and "2 runs" from just the video of the ball is a difficult one even for humans, since it requires one to pay attention to the pitch of the cricket field for the duration of the ball, and this is often impossible because the camera pans away from the pitch to follow the trajectory of the ball. For this reason, we group all such balls into a generic "run" class.

4.3. Final Dataset

Our final dataset contains 480 balls of gameplay. Of these, we use 100 balls for validation and 100 balls for test. Using the jittered sampling method described below for data augmentation, we augment the remaining 280 balls to 600 balls total, and train on that set. The original class distributions for each class in the original dataset are given in Table 1. As shown in the table, the natural distribution of the "wicket" class is particularly low - only 5% of the examples in our dataset have this label - and the distribution of "boundaries" is relatively low, too. However, we were most interested in learning to predict these classes, since they are the two visually and semantically most interesting classes in cricket. For this reason, we artificially boosted our training set to create a more even distribution (shown in Table ??). A sample video (subsampled to 5 frames) is shown in Figure 4.

Class	% of Examples
Run	50%
No Run	30%
Boundary	15%
Wicket	5%

Table 1. Natural distribution of classes in our dataset











Figure 4. A sample video in our dataset

Class	% of Examples
Run	35%
No Run	25%
Boundary	20%
Wicket	20%

Table 2. Distribution of classes after boosting

4.4. Jittered Sampling

Videos typically have 30 frames per second, and the video of a single ball in cricket can be approximately 5 seconds long. This means that each video will have 150 frames each of size 224x224x3 i.e an input dimension of 224x224x3x150 22.5 million values! Due to the high dimensionality of the original data, it becomes necessary to heavily downsample the videos. We do this by simply using only a subset of the frames of each clip. Since we have no prior information about which frames are useful, our subsampling methods must give each frame an equal probability of being selected. At the same time, frames which are temporally close to each other convey similar information, and so we would like to avoid subsampling all the frames from the same temporal region. To simultaneously address both needs, we devised a sampling method called Jittered Sampling.

Given the problem of selecting f frames out of f_{total} frames, Jittered Sampling works as follows. We first divide the set of frames into f contiguous windows, where window i consists of frames $floor(\frac{f_{total}}{f})*(i-1)$ through $floor(\frac{f_{total}}{f})*(i)$. We then select one frame uniformly at random from each window.

One very important side-effect of jittered sampling is data augmentation. Given a video having f_{total} frames where we want only $f < f_{total}$ frames, there are $f^{\frac{f_{total}}{f}}$ possible "jittered" versions. This means that the same training example can be sampled multiple times to produce different sequences of frames.

4.5. Data Preprocessing

The original videos that we obtained had a resolution of 1280x720, and thus each frame has 1280x720x3 pixels. Since the VGG16Net was trained on images of size 224x224, however, we preprocess all our videos by resizing

them to size 224x274 and then cropping the frames by removing the 50 pixels at the bottom of each frame. We perform this cropping in order to remove the information bar at the bottom of each frame that provides the score at that instant; this is done to ensure that our models don't inadvertently learn to classify outcomes using features of the scoreboard. We also subtract the mean BGR values, that the VGG16Net was originally trained with, from every frame, to maintain consistency with the VGG16Net input representation.

5. Experiments/Results/Discussion

(2-3 pages) Table 3 compares the video classification performance of each of the different architectures on the training and validation set. We also include random baseline accuracies on the validation set for two baselines: a "completely naive" baseline that simply predicts any one of the output labels with equal probability, and a "naive" baseline that predicts any one of the output labels based on an a prior understanding of the probability distribution over the classes.

We see from these results that the LRCN strongly outperforms both the late fusion model and single frame averaging model. At the same time, the success of the single frame averaging method is much higher than would be expected. In each of the subsections below, we discuss in detail the training hyperparameters and optimization method used for each model type, and show a detailed evaluation of the best performing model (LRCN). We also discuss possible reasons for the success or failure of each model.

5.1. Late Fusion

When training the late fusion model, we used an initial learning rate of 7e–5, that was decayed by $\gamma=0.995$ after every epoch, and a regularization factor of 1e–3 that was applied to the parameters of all the fully connected layers (including the output layer). We used the Adam update method to update weights at every iteration. When training this model, we subsampled 5 frames per clip (i.e. per ball), and thus the input to the first fully connected layer, fc_6 , is an input volume that is 5x as large as the output volume of the last convolutional layer (since all the output volumes for a given clip are stacked together to form the input to

Model	Training accuracy	Validation accuracy	
Completely Naive Baseline	-	25%	
Naive baseline	-	42.4%	
Late Fusion	82%	65%	
Single Frame Averaging (only train fc_8)	85.33%	71.5%	
Single Frame Averaging (finetune fc_7)	86.67%	76.5%	
Single Frame Averaging (finetune fc_6 and fc_7)	88%	77%	
LRCN	92%	78%	

Table 3. Training and validation accuracies of different models, with the best hyperparameters for that model.

 fc_6). We see that the Late Fusion model performs worse even than Single Frame Averaging, even though the primary motivation behind this method is to synthesize information across time before producing an output. We posit that this is because of the length of our videos; while the videos in [8] were extremely short, the frames in our dataset are subsampled from much longer videos (5-6 seconds at least). As a result, simply combining information across the first and last frames is not enough, since there is a great deal of motion between those frames that is not captured by late fusion.

5.2. Single Frame Averaging

hyperparameters that we used best-performing single frame averaging model were an initial learning rate of $5\mathrm{e}{-4}$ for the output layer, and a lower rate of 8e-5 for the finetuned layers fc_6 and fc_7 , since we intended to only finetune parameters for these layers rather than fully update them. Both learning rates were decayed at a rate of $\gamma = 0.995$ after each epoch. We used the Adam update method to update weights at every iteration. When training this model, we again subsampled 5 frames per clip, and the minibatch size that we used was 15. One of the interesting features of this model that we observe is that the model seems to to identify salient "shots" that indicate a high probability of a certain class. For example, as we see in Figure 5, it appears as though the model has learned that since shots of just the pitch (the first frame in the figure) appear at the start of almost every clip, they are not strong indicators of any of the classes in particular. Here, the prediction for that frame is simply a prior over the class distributions. However, as the model "sees" more frames of the image, we see that the probability of the right class, "boundary", peaks at the 4th and 5th frames, where the model has a clear view of the bounary and the audience. Boundary and audience shots are rarely seen in other outcomes, where the ball might not reach as far across the field. Therefore, we hypothesize that the model is simply learning to associate different shot types with different outcomes.



Figure 5. Class probabilities per frame from the single-frame averaging model. The correct class here is "boundary", which is maximized at the last two frames, which actually visual contain the boundary of the field.

5.3. LRCN

The LRCN-based approach outperforms all the other approaches explored in this paper. When training the LRCN, we used a learning rate of 2.5e-4 and a regularization factor of 2.5e-1. Again, we used the Adam update method to update our parameters after every iteration. Since the LRCN has temporal dependencies, we experimented with different rates of subsampling, and our best performing model uses videos subsampled at 7 frames per clip. We used a minibatch size of 10 to train this model. After looking at the cases where the LRCN correctly classifies the outcome (but the single-frame averaging model fails), it is clear that temporal dependencies play an important role in correct outcome classification in cricket. For example, in the example in Figure 4, the correct outcome is "no run" because the ball is picked up by a fielder and returned to the pitch before it reaches the boundary. The crucial shot here is the last frame; in a "boundary" clip, the last frame would usually be one of the boundary itself, since the ball is usually not immediately returned to the pitch. However, due to the sequence of frames that show the boundary and the audience, the single frame averaging model predictably classifies this input as a boundary. On the other hand, the LRCN is able to correctly classify this as a "run", presumably because it is able to recognize that this sequence of frames (where there is a shot of the boundary followed by a shot of the pitch) is characteristic of the "run" class rather than the

"boundary" class. The single frame model, which has no notion of temporal dependencies, is unable to recognize this and hence misclassifies this input.

A detailed confusion matrix for the LRCN is shown in Table 4. As seen in the table, one shortcoming of our dataset is that the lack of examples for the "wicket" class results in the presence very few examples of that class in our validation set. Further, all the "wicket" examples are misclassified, which implies that our boosting of the training set may be insufficient to accurately classify these outcomes. We also hypothesize that this is because of the great diversity in types of wickets - there are at least four different types of wickets, each completely different from the other. As a result, the correct detection of this class might require a much larger and much more diverse training set than the one that we have at our disposal.

5.4. Evaluation

One important point to iterate is that both the best performing model architectures seem to rely more on general patterns of each frame (and the sequence of frames, in the case of LRCNs), rather than individual details, like the position of the ball or of individual players. This provides an important insight into many of the mistakes made by these models. Often in cricket, the detection of an outcome lies not only in understanding global features of the video for that ball, but also in seeing where the ball went, whether it was caught or dropped, etc. This is a very tough and open problem, because the ball is often very small (at most a few pixels in height and width), and can very easily be lost in the frames of a video. Thus, completely accurate outcome classification might require the localization of the ball and of individual players. However, our algorithms are able to correctly classify almost 80% of the examples, which is significantly better than a random baseline!

6. Conclusion/Future Work

In this paper, we show that training a Long-Term Recurrent Convolutional Network using a pretrained VGG16Net can show great performance on the task of classifying outcomes from cricket videos. We also compare the LRCN architecture to a single frame based architecture, and show that even the latter is a reasonable model for this problem. Our models are able to achieve approximately 80% accuracy on the validation set for this problem. If we had more computational resources at our disposal, we would definitely want to explore the potential of temporal convolution methods like slow fusion and early fusion (as described in [8]). It would also be extremely interesting to try to extend this problem to the more general problem of commentary generation, which is requires a far more fine-grained understanding of a video than simply outcome classification.

References

- [1] Li Fei-Fei Andrej Karpathy. Deep visual-semantic alignments for generating image descriptions.
- [2] Sergio Guadarrama Marcus Rohrbach Subhashini Venugopalan Kate Saenko Trevor Darrell Jeff Donahue, Lisa Anne Hendricks. Long-term recurrent convolutional networks for visual recognition and description.
- [3] Yang M et al Ji S, Xu W. 3d convolutional neural networks for human action recognition. *IEEE transactions on Pattern Analysis and Machine Intelligence*, pages 221–231.
- [4] Andrew Simonyan, Karen; Zisserman. Two-stream convolutional networks for action recognition in videos.
- [5] Zhimin Gao Jing Zhang Chang Tang Pichao Wang, Wanqing Li and Philip Ogunbona. Deep convolutional neural networks for action recognition using depth map sequences.
- [6] Wikipedia. Depth map Wikipedia, the free encyclopedia, 2014. [Online; accessed 12-March-2016].
- [7] JianXin Song Chi Geng. Human action recognition based on convolutional neural networks with a convolutional auto-encoder. 5th International Conference on Computer Sciences and Automation Engineering (ICCSAE 2015).
- [8] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In CVPR, 2014.
- [9] Wikipedia. Cross entropy Wikipedia, the free encyclopedia, 2016. [Online; accessed 12-March-2016].
- [10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [11] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

	Run	No Run	Boundary	Wicket	Precision
Run	30	10	3	1	0.6818
No Run	12	18	1	2	0.5454
Boundary	3	0	16	4	0.6956
Wicket	0	0	0	0	N/A
Recall	0.6667	0.6428	0.8	0	

Table 4. Confusion matrix for LRCN. The last column and row contain the precision and recall values for each class.

- [12] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Cs231n-stanford-winter1516-lecture11-slide34, 2016. [Online; accessed 12-March-2016].
- [13] Andrew Simonyan, Karen; Zisserman. Very deep convolutional networks for large-scale image recognition.
- [14] ESPN Inc. Pepsi indian premier league 2014. http://www.espncricinfo.com/indian-premier-league-2014/content/series/695871.html. Accessed: 2016-02-15.