Baseline Report: Sequence Models for Soccer Action Classification

Omid Naeej Nejad

December 30, 2025

Summary

This report presents early baseline results for a 3-class soccer action classification task using sequence models (RNN, LSTM, BiLSTM). The evaluated classes are: `Red Card`, `scoring`, and `tackling`. The LSTM baseline achieves the best overall performance with accuracy 93.44 % and macro-F1 89.89 % on the evaluation split.

1 Task Definition

Given an input sequence $X = [x_1, x_2, \ldots, x_T]$, where each $x_t \in R^D$ represents the feature vector at time step t, the goal is to predict one of $C = 3$ action classes:

$$\hat{y} = \arg \max_{c \in \{1,\ldots,C\}} p(y = c \mid X).$$

2 Dataset and Splits

The class distribution across train/validation/test splits is imbalanced, with the `scoring` class having the highest frequency.

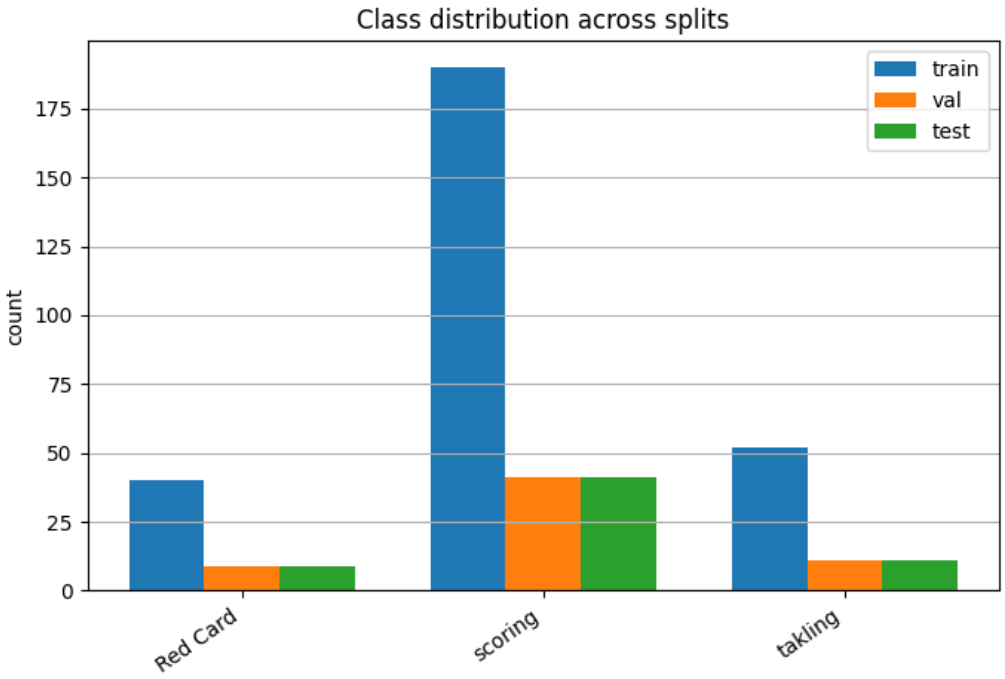


Figure 1: Class distribution across splits (train/val/test).

From the confusion matrices, the evaluation split size is $N = 61$ samples (9 Red Card, 41 scoring, 11 tackling).

3 Models

All models are sequence encoders followed by a classification head. They differ in the recurrent unit used. In our pipeline, the recurrent network does not operate directly on raw RGB frames. Instead, each frame is first encoded into a fixed-length feature vector using a frozen ResNet-18 feature extractor, and the resulting feature sequence is fed to the RNN/LSTM/BiLSTM.

3.1 Frame Feature Encoder (Frozen ResNet-18)

Let I_t denote the t-th RGB frame. We extract a per-frame embedding using a pretrained ResNet-18 backbone with its final classifier removed:

$$x_t = f_{\text{enc}}(I_t), \quad x_t \in R^{512}.$$

The encoder corresponds to ResNet-18 up to (and excluding) the final fully connected layer, producing an output of shape $(B, 512, 1, 1)$ which is flattened to $(B, 512)$. In the implementation, the encoder is set to evaluation mode and all parameters are frozen (no gradient updates), i.e.,

$$\frac{\partial \mathcal{L}}{\partial \theta_{\text{enc}}} = 0.$$

Given a sequence of T frames, this produces the model input:

$$X = [x_1, x_2, \ldots, x_T] \in R^{T \times 512}.$$

3.2 Common Input/Output Interface

Input:

$$X \in R^{T \times D}, \quad \text{with } D = 512 \text{ in our setup.}$$

Encoder produces hidden states h_t (or concatenated states for bidirectional models). A sequence-level representation z is formed (commonly the last state h_T, or pooling over time):

$$z = \text{Pool}(h_1, \ldots, h_T).$$

Classifier:

$$\text{logits} = Wz + b, \quad p(y \mid X) = \text{softmax}(\text{logits}).$$

Note: The exact hyperparameters (hidden size, number of layers, dropout, pooling choice, optimizer, learning rate, batch size, epochs) should be copied from the project notebook/code.

3.3 Vanilla RNN

A standard recurrent update:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b).$$

Strength: simple baseline. Limitation: weaker long-range dependency modeling due to vanishing gradients.

3.4 LSTM

LSTM introduces gates to better preserve long-term dependencies:

$$\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
h_t &= o_t \odot \tanh(c_t).
\end{aligned}$$

3.5 Bidirectional LSTM (BiLSTM)

BiLSTM runs two LSTMs:

$$\overrightarrow{h}_t = \text{LSTM}_{fwd}(x_t), \quad \overleftarrow{h}_t = \text{LSTM}_{bwd}(x_t),$$

then concatenates:

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t].$$

This can improve performance when both past and future context help, but it may overfit or behave worse depending on data size and noise.

4 Evaluation Metrics

We report:

- Accuracy
- Macro-F1 (treats all classes equally, sensitive to minority-class performance)
- Weighted-F1 (weights classes by support)

5 Results

5.1 Overall Performance

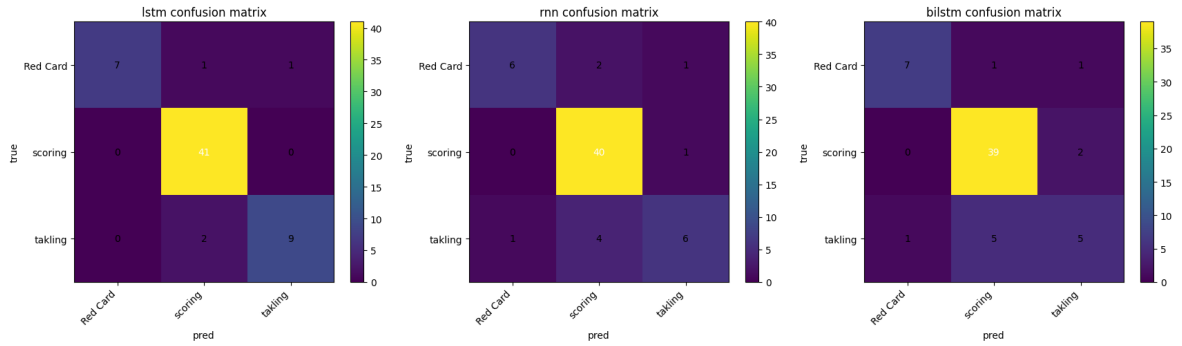
Table 1 summarizes overall metrics and saved best checkpoints.

Model	Accuracy	Macro-F1	Weighted-F1	Best checkpoint
RNN	0.8525	0.7670	0.8426	`/content/runs/soccer_seq/rnn/rnn_best.pt`
LSTM	0.9344	0.8989	0.9321	`/content/runs/soccer_seq/lstm/lstm_best.pt`
BiLSTM	0.8361	0.7523	0.8260	`/content/runs/soccer_seq/bilstm/bilstm_best.pt`

Table 1: Overall performance on the evaluation split.

5.2 Confusion Matrices

Confusion matrices (rows=true, columns=predicted) are shown below. Figure placeholders are included for your saved plots.



(a) LSTM (b) RNN (c) BiLSTM

Figure 2: Confusion matrices for the evaluated models.

For completeness, the numeric matrices are:

LSTM (true \ pred)	Red Card	scoring	tackling
Red Card	7	1	1
scoring	0	41	0
tackling	0	2	9

RNN (true \ pred)	Red Card	scoring	tackling
Red Card	6	2	1
scoring	0	40	1
tackling	1	4	6

BiLSTM (true \ pred)	Red Card	scoring	tackling
Red Card	7	1	1
scoring	0	39	2
tackling	1	5	5

Table 2: Numeric confusion matrices (rows=true, columns=predicted).

5.3 Per-class Precision/Recall/F1 on Evaluation Split

Using the confusion matrices, Table 3 reports per-class metrics.

Model	Class	Precision	Recall	F1
LSTM	Red Card	1.0000	0.7778	0.8750
LSTM	scoring	0.9318	1.0000	0.9647
LSTM	tackling	0.9000	0.8182	0.8573
RNN	Red Card	0.8571	0.6667	0.7500
RNN	scoring	0.8696	0.9756	0.9190
RNN	tackling	0.7500	0.5455	0.6316
BiLSTM	Red Card	0.8750	0.7778	0.8235
BiLSTM	scoring	0.8667	0.9512	0.9066
BiLSTM	tackling	0.6250	0.4545	0.5263

Table 3: Per-class metrics computed from the confusion matrices.

6 Discussion

6.1 Key Observations

- LSTM performs best overall and achieves perfect recall on `scoring` (41/41 correctly classified).
- The main remaining errors for LSTM are confusions between `tackling` and `scoring` (2 tackling samples predicted as scoring), and a small number of `Red Card` samples confused with the other two classes.
- Both RNN and BiLSTM struggle more on `tackling`, which lowers macro-F1 due to class imbalance.

6.2 Impact of Class Imbalance

Macro-F1 is substantially lower than weighted-F1 for RNN and BiLSTM, indicating that minority-class performance (especially `tackling` and `Red Card`) is weaker. This aligns with the split distribution where `scoring` has much higher support.

7 Next Steps (Planned Improvements)

These are suggested directions for the next iteration of the baseline study:

- Report exact model/training hyperparameters from the code (hidden size, layers, dropout, pooling, optimizer, LR schedule).
- Add strategies for class imbalance: class-weighted loss, focal loss, or balanced sampling.
- Add regularization and stability checks: dropout tuning, early stopping, multiple seeds, and mean/std reporting.
- Try stronger sequence encoders: temporal CNN/TCN, Transformer encoder, or attention pooling on top of LSTM.
- Add ablations: sequence length T, feature dimension D, pooling choice (last vs mean vs attention).

A Reproducibility Checklist

Fill these from your notebook/code to make the report fully reproducible:

- Dataset source and preprocessing:
  - feature type (what x_t represents), normalization, sequence padding/truncation strategy
  - split procedure and random seed
- Training:
  - optimizer, learning rate, weight decay, batch size, epochs, LR scheduler
  - loss function and any class weighting
- Model hyperparameters:
  - hidden size, number of layers, dropout, bidirectionality, pooling choice
- Hardware/software:
  - PyTorch, CUDA, GPU: Colab T4