Recurrent CNN for Video Object Tracking

Rowshanak Hosseinzadehattar*, Ghazal Hodaei*, Seyedali Divbandroudbaraki*, Muhannad Albakkar*
*Faculty of Engineering, University of Rostock, Germany
{rowshanak.hosseinzadehattar, ghazal.hodaei, seyedali.divbandroudbaraki, muhannad.albakkar}@uni-rostock.de

***Abstract*—Object tracking in video streams is a crucial task in computer vision, with applications in surveillance, traffic monitoring, and activity analysis. While single-stage detectors such as YOLO enable real-time object detection, they lack the temporal modeling required for robust tracking in dynamic scenes.**

In this paper, we introduce key concepts and neural network architectures relevant to the task of object tracking in videos. We then present a practical implementation of an object tracking system based on an open-source framework that integrates Convolutional Neural Networks (CNNs) with Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) units. Our system demonstrates how spatial features extracted by CNNs can be effectively combined with temporal dependencies modeled by RNNs to achieve accurate and stable tracking across video frames.

Building upon the ROLO (Recurrent YOLO) architecture, we analyze each component of the system and highlight how the fusion of object detection and temporal regression enhances performance in challenging scenarios. The system is evaluated on benchmark datasets, illustrating the advantages of using Recurrent Convolutional Neural Networks (RCNNs) for spatiotemporal prediction. This approach bridges the gap between static object detection and dynamic video understanding by leveraging both visual semantics and motion history to enable real-time, context-aware tracking.

***Index Terms*—CNN,RNN,RCNN,LSTM, Object Tracking, Video Object Detection, Deep Learning, Computer Vision**

I. Introduction

Before one can understand the concept of RCNNs, it is important to have a basic understanding of both CNNs and RNNs. This section briefly introduces these concepts and explains the motivation behind combining them.

A. CNNs, RNNs, and RCNNs – Principles and Motivation

Convolutional Neural Networks (CNNs) are a type of deep neural network particularly effective for analyzing visual data such as images. Originally inspired by the structure of the visual cortex, CNNs apply layers of convolutional filters and pooling operations to extract spatial hierarchies of features directly from raw image data. This allows them to learn increasingly complex patterns, starting from simple edges to full objects. For example, imagine a video of a man running. If we extract a single frame and feed it into a CNN, the network can recognize that there is a man in the image. However, it cannot infer any motion or temporal patterns from a single frame. Over the past decade, CNNs have become the backbone of state-of-the-art systems in image classification, object detection, and segmentation [5].

Recurrent Neural Networks (RNNs), on the other hand, are designed to process sequential data. Unlike traditional

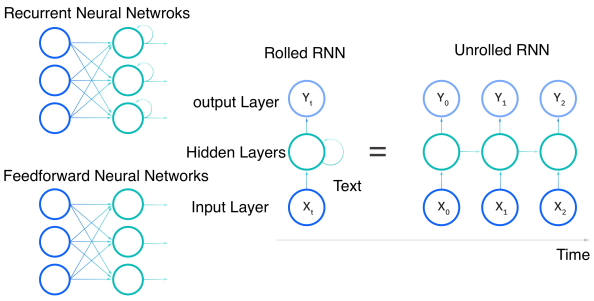

Fig. 1. Feedforward vs recurrent neural networks [3]

feedforward networks, where connections flow only in one direction, RNNs include recurrent connections within layers. This structure allows them to retain a ”memory” of previous inputs through hidden states, making them particularly suitable for tasks involving time dependencies (see Figure 1) such as time series prediction, language modeling, and speech recognition. An RNN could analyze the sequence of frames from the running video and recognize the temporal pattern of running. However, it would not inherently know what is running—just that there is a repeated motion [7]. This is where RCNNs (Recurrent Convolutional Neural Networks) come in. By combining the spatial feature extraction power of CNNs with the temporal modeling capabilities of RNNs, RCNNs can identify an object (e.g., a man) and track its movement over time across multiple frames. In essence, instead of analyzing a single frame in isolation, an RCNN processes a sequence of frames to recognize both what is in the scene and how it changes over time; in our example using RCNN we can track a man running through a sequence of frames. One of the earliest and most biologically inspired versions was proposed by Ming Liang and Xiaolin Hu, where recurrent connections were added within each convolutional layer to mimic lateral interactions in the visual cortex. This design reflects biological findings that neurons in the neocortex receive not only feedforward input but also contextual feedback from neighboring units. Such recurrent synapses are believed to play a critical role in integrating spatial context and enhancing visual perception, especially under ambiguous or occluded conditions [6].

B. Deeper Look at RNNs and LSTM

As it can be seen in Figure 1 Unlike feedforward networks, which map inputs and outputs in a fixed one-to-one fashion,

recurrent neural networks are capable of handling variable-length input and output sequences. This flexibility makes RNNs useful in various domains such as Music generation, Sentiment classification and Machine translation. Several RNN variants have been developed to address different use cases and limitations, including:

- Standard RNNs,
- Bidirectional RNNs (BRNNs),
- Long Short-Term Memory (LSTM),
- Gated Recurrent Units (GRUs),
- Encoder-decoder RNNs

In this paper, we focus on LSTM, as it is also used in the project introduced in a later section. The basic RNN cannot access long-range context due to the back-propagated error either inflating or decaying over time, which is called the vanishing gradient problem [2], which hinders its ability to learn long-term dependencies. This means that if relevant information from earlier time steps is too far in the past, the network struggles to use it effectively in current predictions. LSTM networks, proposed by Hochreiter and Schmidhuber (1997), were designed to solve this problem. They introduce a memory cell along with three gates:

- Input gate (controls how much new information enters),
- Forget gate (determines what information is discarded),
- Output gate (controls what is passed to the next step)

For example, take the sentence: "Alice is allergic to nuts. She can't eat peanut butter." As explained by IBM [3], a model needs to remember that Alice is allergic to nuts—even though it was mentioned earlier—to correctly infer that peanut butter is problematic, however if the pronoun will be repeated multiple times, it can be forgotten. Standard RNNs often fail here, but LSTMs are designed to capture such long-range dependencies. Despite all of this, RNNs—including LSTMs—have declined in popularity in recent years, mainly due to the rise of Transformer-based models like BERT and GPT, which are more effective in capturing long-range dependencies, easier to parallelize, and deliver superior performance in many NLP and time-series tasks.

C. LRCN: Dealing with vanishing gradient

A significant contribution to the state-of-the-art in spatiotemporal modeling is the Long-term Recurrent Convolutional Network (LRCN) architecture, proposed by Donahue et al. This model is designed to handle sequential visual data such as videos and structured language outputs such as captions. Unlike traditional CNN-only approaches, LRCN incorporates temporal modeling through recurrent units—specifically LSTM cells—allowing the network to learn both spatial hierarchies and temporal dependencies in an end-to-end trainable fashion [1].

The core idea behind LRCN is to decouple the spatial and temporal components of the model. A convolutional neural network (e.g., CaffeNet) extracts visual features independently for each frame or image [4]. These features are then passed into one or more LSTM layers, which process the sequence over time. The recurrent layers are responsible for modeling dynamic behavior in video inputs or generating sequences of words for image or video description .

The architecture supports three categories of tasks:

1) Sequential input, static output (e.g., activity recognition),
2) Static input, sequential output (e.g., image captioning),
3) Sequential input and output (e.g., video description).

Each task is instantiated with a specific adaptation of the LRCN pipeline. For activity recognition, sequences of visual frames are encoded and temporally processed to produce a single classification label. For image captioning, the model generates sentences based on static image features. For video description, LRCN is used in an encoder-decoder configuration, mapping video frames to a textual description of arbitrary length [1].

An important strength of LRCN is that both components—CNN and LSTM—are **jointly trained** using back-propagation, allowing the convolutional layers to adapt their representations to the needs of the temporal decoder. This end-to-end optimization contrasts with earlier approaches that either froze CNN weights or used handcrafted features for temporal modeling [1].

LRCN also explores different sequence modeling architectures, such as single-layer vs. stacked LSTMs, and *factored* vs. *unfactored* models. In the factored variant, image features are injected into only the upper layers of the LSTM stack, separating language modeling from visual encoding responsibilities [1].

Experimental results across multiple datasets demonstrate the effectiveness of LRCN:

- On the **UCF101** [12] activity recognition benchmark, LRCN significantly outperforms single-frame baselines, especially when combining RGB and optical flow inputs.
- In **image captioning** on COCO and Flickr30k, LRCN competes with the strongest existing systems and shows improvement from both fine-tuning the CNN and deeper sequence modeling.
- In **video description** on the TACoS multilevel dataset, LRCN with probabilistic CRF inputs achieves state-of-the-art BLEU scores, outperforming statistical machine translation baselines.

Overall, the LRCN model provides a flexible and powerful architecture for a broad range of vision-language tasks and demonstrates how deep learning systems can effectively integrate spatial and temporal reasoning.

D. ROLO: Bridge the Gap

At this stage, we can now take a look at another impactful approach presented by Guanghan Ning et al. in their ROLO framework, which combined CNN-based object detection(YOLO) with an LSTM-based temporal tracking module, creating a spatially and temporally supervised object tracker which will be the core approach to our project [7]. The authors has claimed the key motivation behind this method is that tracking failures can often be effectively recovered by learning

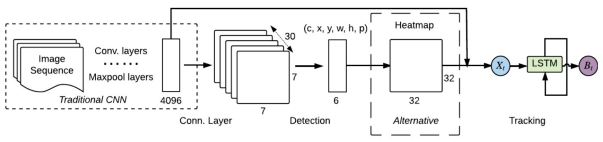

Fig. 2. the proposed Architecture

from historical visual semantics and tracking proposals, when an object due to several reasons,(eg. Occlusion, Motion Blur) it can be recovered using the memory of the last frames.

ROLO addresses two major limitations of traditional tracking systems, first reliance on handcrafted features and the separation of detection and temporal modeling. In contrast, ROLO is designed as an end-to-end trainable ,with gradient-based learning methods, architecture that fuses high-level visual features and location information. The system uses YOLO to extract a rich feature vector of size 4096 from each frame and to generate preliminary bounding box coordinates. These features and spatial data are then concatenated and passed through an LSTM, which performs spatiotemporal regression to predict object positions over time [11]. Generaly the system follows a three-stage end-to-end training process.

1) It uses **YOLO** to extract robust visual features and provide initial location predictions for objects within video frames.
2) These features, along with the bounding box coordinates, are *concatenated* and passed to an **LSTM**, which models the temporal evolution of object positions. This allows the network to correct short-term failures by learning long-term dependencies.
3) Inspired by YOLO's regression-based approach to localization, it explores the regression capabilities of LSTM in a spatio-temporal context.

As illustrated in Figure 2, the full ROLO training pipeline consists of three phases: Pre-training the CNN to learn general visual features, Training the YOLO model for object detection and Training the LSTM to perform tracking based on the fused visual and positional inputs.

Unlike conventional tracking systems that rely on binary classification to verify region proposals, ROLO treats tracking as a regression problem. This allows it to directly predict bounding box coordinates or even a heatmap representation of the object's position. The authors demonstrate that this method is not only more accurate but also more resilient to common challenges in tracking, such as occlusions, fast motion, and motion blur. In several benchmark datasets, ROLO outperformed state-of-the-art tracking models by a significant margin.

Another key advantage of ROLO lies in its modular design, as shown in Figure 2. The YOLO component ensures fast and accurate visual feature extraction and region proposals, while the LSTM leverages historical information from both previous bounding boxes and visual content. This combination enables the network to maintain tracking even when YOLO fails to detect the object due to occlusion or poor lighting conditions. Qualitative examples in the paper show that ROLO continues to predict plausible object locations even when the object disappears temporarily from view.

Additionally, the model supports an alternative design where spatial locations are encoded as heatmaps. This not only improves interpretability but also enables the LSTM to learn confidence-based spatial reasoning. In the case of multiple similar objects in the frame, ROLO learns to focus on the correct target by leveraging both visual features and its internal memory of location history [7].

From an implementation perspective, ROLO achieves real-time performance by decoupling the CNN-based feature extraction from the temporal modeling. YOLO runs at up to 45 fps, while the LSTM module operates at more than 60 fps, making the combined system suitable for applications such as surveillance, autonomous vehicles, or real-time robotics [7].

In summary, ROLO exemplifies a successful fusion of spatial and temporal modeling within a deep learning framework. Its architecture not only advances the capabilities of object tracking systems but also provides a practical blueprint for our own implementation, where real-time, robust performance is essential [7].

E. Project Inrtoduction

By mimicking certain properties of the human brain's neural architecture—such as temporal recurrence, memory, and context-dependent activation—RCNNs allow artificial neural units to evolve their activation over time, even when processing static inputs—enabling better context-aware recognition. Although our implementation focuses on temporal recurrence between frames via LSTM, rather than lateral recurrence within layers, the conceptual foundation of combining spatial and temporal information remains aligned with this biologically inspired vision. Our system similarly benefits from context propagation: CNNs extract local spatial features, while RNNs (LSTMs) model temporal coherence across video frames.

Building upon these concepts, our project implements an object tracking system that integrates CNNs and RNNs, based on an open-source project `Guanghan/ROLO`. The system is designed to track moving objects across video frames by learning both visual representations and temporal motion patterns. It leverages:

- A CNN (e.g., pre-trained YOLO or custom feature extractor) for spatial feature encoding,
- An RNN (typically LSTM) to model temporal dependencies across frames,
- Python and TensorFlow for implementation,
- Training on benchmark video datasets such as OTB and COCO-based sequences.

The pipeline includes preprocessing video data into frame sequences, feeding them through a CNN to obtain high-level feature vectors, and passing these into an RNN to predict the next location of the object. The model is trained using Mean Squared Error (MSE) loss on bounding box coordinates, and

can optionally use heatmap regression for more interpretable predictions.

This integration of CNN and RNN techniques offers a powerful approach for handling real-world tracking challenges such as occlusion, abrupt motion, and appearance changes, and provides a valuable baseline for future research in video object tracking and spatiotemporal learning.

In summary, this project contributes a fully integrated and analyzed object tracking pipeline that combines spatial and temporal learning components, grounded in both state-of-the-art engineering practices and biologically inspired neural network design.

II. Methodology

A. Project Overview

The core goal of this project is to implement a system capable of tracking objects in video streams by combining spatial feature extraction using Convolutional Neural Networks (CNNs) with temporal sequence modeling using Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) units. The project integrates YOLO for object detection and LSTM for predicting consistent object positions across frames.

B. System Pipeline and Demonstration

The updated system is built on a modular structure using a modern Python environment and a user-friendly Streamlit interface. The process begins with uploading a video through the main application `app.py`. The video is split into individual frames, each of which is analyzed by a YOLOv3 object detection model.This model is integrated using OpenCV's DNN module and relies on the configuration files `yolov3.cfg` [9], `yolov3.weights` [10], and `coco.names` [8], all stored within the `yolo/` directory.

For each frame, the detector returns object class predictions and corresponding bounding boxes. These outputs are then passed to a simulated ROLO module, implemented in `rolo/rolo_sim.py`. Instead of a full LSTM-based architecture, this module employs simplified logic that mimics tracking behavior.

The results are visualized by overlaying bounding boxes on the video frames and displaying them in real-time within the Streamlit interface. This interactive design allows for an intuitive demonstration of the full tracking pipeline.

Figure 3 presents an overview of the system architecture and data flow between its components.

In the sample output shown in Figure 4, red indicates the ground truth bounding box, green shows the YOLOv3 detection, and blue marks the predicted location from the ROLO simulation.

C. Modifications and Adaptations

In addition to the simulated demo shown earlier, Figure 5 presents a real output frame from our re-implemented pipeline. The system successfully detects a vehicle on the highway,

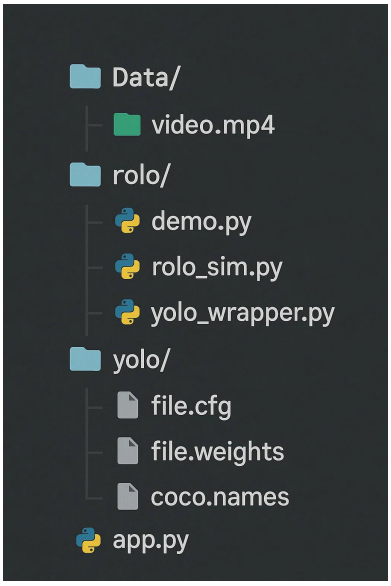

Fig. 3. Updated system architecture combining YOLOv3, ROLO simulation, and Streamlit frontend

Fig. 4. Tracking output on a sample video frame from original projekt

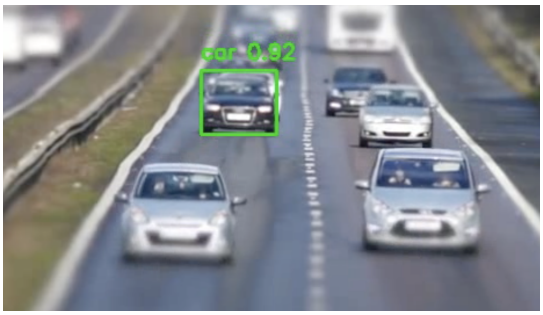

Fig. 5. Real detection output from our re-implemented system using YOLOv3

marks it with a bounding box, and displays the object class ("car") with a confidence score of 0.92. This result was generated using the current YOLOv3-based detection integrated in our Streamlit application.

Due to the deprecated nature of the original ROLO implementation, several model files and dataset links (such as `.mat` and `.ckpt` files) were no longer accessible. To address this, the system was modernized while preserving the original concept and pipeline logic.

Specifically, a compatible pre-trained YOLOv3 model was downloaded from Kaggle and integrated into the pipeline using OpenCV's DNN interface. A custom YOLO configuration file was adapted to fit the TensorFlow-based architecture, and the `coco.names` file was reduced to include only object classes relevant to our use case (e.g., `person`, `bicycle`, `car`). A sample video (`test_video.mp4`) was used to demonstrate the system in action.

Missing components from the original ROLO codebase were replaced with self-developed scripts and logic that simulate the behavior of the original modules. This allowed us to build a functioning prototype that faithfully represents the ROLO pipeline, while leveraging current libraries and maintaining compatibility with modern systems.

III. Conclusion

In this paper, we presented an object tracking system that integrates Convolutional Neural Networks (CNNs) with Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) units, to jointly model spatial and temporal information in video data. This hybrid architecture addresses key limitations of traditional trackers by enabling context-aware, memory-driven predictions over time.

We reviewed foundational models such as LRCN and ROLO, highlighting their contributions to video understanding, and implemented a modernized, modular version of the ROLO pipeline using YOLOv3 and simulated LSTM logic. Our system demonstrates how temporal modeling enhances detection continuity and robustness in dynamic environments.

Overall, recurrent convolutional models like LRCN represent an important step toward more intelligent, context-aware video understanding. They open the door for future developments in areas such as real-time tracking, captioning, and autonomous systems.

IV. Discussion and Future Work

Our implementation shows that combining CNN-based detection with RNN-based temporal modeling improves tracking performance in dynamic video scenes. However, due to limitations in model availability, we simulated the LSTM logic rather than training a full recurrent network.

Future work will focus on integrating a trainable LSTM module for end-to-end learning and extending the system to support multi-object tracking. Additionally, replacing YOLOv3 with newer architectures and exploring transformer-based models could further enhance accuracy and real-time performance.

Improving the user interface and adding features such as live webcam tracking and confidence visualization are also planned to support more practical use cases.

Author Contributions

Rowshanak Hosseinzadehattar: Writing – Original Draft, Conceptualization.
Ghazal Hodaei: Writing – Review & Editing, Visualization.
Seyedali Divbandroudbaraki: Software, Implementation, Methodology.
Muhannad Albakkar: Investigation, Literature Review.

AI Usage Statement

Generative AI tools, including ChatGPT by OpenAI, were used to support writing, editing, and improving the clarity and structure of this paper. All content has been reviewed and validated by the authors to ensure accuracy, originality, and compliance with academic standards. The authors take full responsibility for the integrity and correctness of the published work.

References

[1] Jeff Donahue et al. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.4 (2017), pp. 677–691. DOI: 10.1109/TPAMI.2016.2599174. URL: https://doi.org/10.1109/TPAMI.2016.2599174.

[2] Sepp Hochreiter et al. "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies". In: *A Field Guide to Dynamical Recurrent Neural Networks*. Ed. by John F. Kolen and Stefan C. Kremer. IEEE Press, 2001, pp. 237–243.

[3] IBM. *Recurrent Neural Networks*. Accessed: 2025-06-04. URL: https://www.ibm.com/think/topics/recurrent-neural-networks.

[4] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM International Conference on Multimedia (ACM MM)*. ACM, 2014, pp. 675–678. DOI: 10.1145/2647868.2654889. URL: https://doi.org/10.1145/2647868.2654889.

[5] Teja Kattenborn et al. "Review on Convolutional Neural Networks (CNN) in vegetation remote sensing". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 173 (2021), pp. 24–49. DOI: 10.1016/j.isprsjprs.2020.12.010.

[6] Ming Liang and Xiaolin Hu. "Recurrent Convolutional Neural Network for Object Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3367–3375. DOI: 10.1109/CVPR.2015.7298958.

[7] Guanghan Ning et al. "Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking". In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 1394–1402.

[8] Joseph Redmon. *COCO Class Names File (coco.names)*. Accessed: 2025-06-04. 2018. URL: https://github.com/pjreddie/darknet/blob/master/data/coco.names.

[9] Joseph Redmon. *YOLOv3 Configuration File (yolov3.cfg)*. Accessed: 2025-06-04. 2018. URL: https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg.

[10] Joseph Redmon. *YOLOv3 Pre-trained Weights (yolov3.weights)*. Accessed: 2025-06-04. 2018. URL: https://huggingface.co/spaces/Epitech/Scarecrow/blob/main/yolov3.weights.

[11] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. URL: http://pjreddie.com/yolo/.

[12] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes from Videos in the Wild*. Technical Report CRCV-TR-12-01. Center for Research in Computer Vision, University of Central Florida, 2012. URL: https://www.crcv.ucf.edu/data/UCF101.php.