Luis Angel Rivas Olvera, Stefanny Arboleda Ceferino

Visión Computacional I

11 de junio de 2025

Proyecto: ROLO en PyTorch

I. Introducción

En este proyecto se implementó el algoritmo ROLO de el artículo "Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking (ROLO)[2]" en PyTorch dado que originalmente fue implementado en Tensorflow. Se utiliza una arquitectura de detección (YOLO v2), más una red de aprendizaje temporal (LSTM) para rastrear objetos únicos a lo largo de un video. En nuestro caso, además de portar el método a PyTorch, se tenía como objetivo realizar una mejora respecto al detector (YOLOv2 small) usado en el artículo original para así comparar los resultados que se podían obtener actualmente con un proceso más flexible del original en cuanto a detecciones y siguiendo la misma lógica del artículo en cuanto a los experimentos y la separación de los conjuntos de entrenamiento y prueba. Se presentan todos los aspectos necesarios para esta actualización, así como los resultados obtenidos y nuestras conclusiones.

II. Aspectos principales.

Para portar el modelo ROLO a PyTorch, se siguieron los principios fundamentales del artículo original pero adaptando cada componente a la sintaxis y buenas prácticas de PyTorch.

- **Extracción de características:** Se reemplazó el YOLO v2 original por YOLO v8 de Ultralytics, aprovechando su facilidad de uso en PyTorch y su capacidad para extraer representaciones semánticas profundas. Se utilizó la variante YOLO v8s pues originalmente también se usa la versión pequeña. Para lograr lo anterior se implementó una clase llamada `YOLOv8FeatureExtractor.py` en la cual se realizan dos tareas principales:
  1. Detectar el objeto con mayor confianza en una imagen.
  2. Extraer un vector de características profundo a partir de las capas internas del modelo YOLO v8s, específicamente de la última capa del backbone de YOLO v8s, cuya salida tiene 512 canales de características.
- **Normalización de cajas:** los ground truths y las predicciones del detector se transformaron al formato normalizado (YOLO: `[x_center, y_center, width, height]`) con respecto a una resolución fija de 640×640 para asegurar la compatibilidad con los datos de entrada a la LSTM pues la entrada de YOLO es esta por defecto.
- **Dataset secuencial:** se creó un dataset personalizado en PyTorch a través de la clase (`ROLOConcatDataset`), disponible en los archivos correspondientes al entrenamiento, la cual genera secuencias temporales de longitud seq_len = 6. Donde, cada muestra contiene seq_len vectores de

características concatenados con los bounding boxes predichos por YOLO v8, y se empareja con el ground truth del último frame de la secuencia.

- **Red ROLO en PyTorch:** se implementó una arquitectura basada en LSTM que recibe como entrada la secuencia de vectores `[features + bbox_pred]` por frame, donde `features` es de dimensión 512 y `bbox_pred` es de 4 coordenadas normalizadas. Esta red fue entrenada para minimizar el error cuadrático medio con respecto al ground truth del último frame en la secuencia.

III. Flujo de trabajo

Antes de explicar el flujo de trabajo implementado, es importante destacar algunas diferencias clave con respecto al artículo original.

Dataset. En dicho artículo, se realiza un preentrenamiento de `YOLOv2` mediante *fine-tuning* con el conjunto de datos `ImageNet`, seguido de un entrenamiento específico sobre el dataset `Pascal VOC`. Esto se debe a que `YOLOv2`, en su configuración por defecto, no está entrenado para detectar los objetos presentes en las secuencias de la base de datos utilizada (`OTB-100`). Por esta razón, se seleccionan únicamente 30 videos —denominados `OTB-30`— de los 100 disponibles en dicha base.

Es importante resaltar que, estos 30 videos fueron seleccionados porque sus clases corresponden a aquellas disponibles en el dataset de entrenamiento `Pascal VOC`, lo que originalmente permite realizar fine-tuning de YOLO v2 sobre el mismo dominio. Sin embargo, recuérdese que en nuestro caso no se está realizando dicho fine-tuning. Por lo anterior, luego de identificar dichos 30 videos se ha creado el archivo `otb30_list.txt` a través del cual se van a cargar los nombres de las carpetas a usarse para entrenamiento, prueba y evaluación, siendo esta última un paso opcional que no está incluido en el artículo original (no explícitamente). Así, los primeros 30 nombres presentes en este archivo se usarán para entrenamiento / prueba y los 3 últimos para probar la capacidad de generalización (evaluación) que tiene el experimento realizado.

Por último, se habla de experimento porque se han realizado tres experimentos diferentes variando la forma de entrenamiento y prueba, igual que en el artículo original, a saber:

- *Experimento 1:* En este experimento, se utilizan 22 videos para entrenamiento y 8 para test, se incluyen todos los frames de cada video y sus respectivos ground truths.
- *Experimento 2:* En este caso, se utilizan solo 1/3 de los frames disponibles para cada uno de los 30 videos con sus respectivos ground truths. Y se utilizan las secuencias completas con sus respectivos ground truths para el test.
- *Experimento 3:* Finalmente, en el que corresponde al mejor de los modelos presentados y sobre el cual se muestran métricas en el artículo original, se utilizan todos los frames pero sólo 1/3 de los ground truths para entrenamiento y se realiza el test igual que en el experimento anterior.

Así pues, habiendo reproducido los mismos experimentos, bajo las mismas condiciones de elección y preprocesamiento del dataset, los resultados presentados en este reporte son comparables con los del artículo original.

Continuando, el flujo de trabajo implementado sigue una estructura de preprocesamiento, entrenamiento y evaluación (capacidad de generalización) de la siguiente manera:.

1. **Preprocesamiento y extracción:** Dependiendo el experimento a realizar ({1,2,3}) se usa el dataloader correspondiente a dicho experimento, a saber:
   I. *Experimento 1:* Se utiliza el archivo "`dataloader.py`".
   II. *Experimento 2:* Se utiliza el archivo "`dataloader_exp2.py`".
   III. *Experimento 3:* Se utiliza el archivo "`dataloader_exp3.py`".

   en el cual, se recorren los videos del dataset OTB-30, uno a uno, creando carpetas separadas entre los que son para entrenamiento, los que son para prueba y finalmente aquellos que van a usarse para evaluación. Para cada video, cada frame se redimensiona a 640x640 y se le aplica YOLO v8s, donde se extrae:
   - El vector de características del backbone de YOLO v8s.
   - La caja predicha (bbox_pred) por YOLO v8s.
   - La caja ground truth del dataset (a través del archivo `groundtruth_rect.txt`), normalizada a formato YOLO.

   Finalmente, los datos se almacenan en carpetas cuyos nombres corresponden al nombre de cada video, en la ruta especificada (en el archivo .py de dataloader correspondiente) y en formatos "`.npy`" para cada video. A saber: `features.npy`, `bbox_yolo.npy`, `gt.npy`.
2. **Generación del dataset secuencial:** Una vez que hemos cargado los datos a través del dataloader, podemos pasar al entrenamiento a través del archivo "`train.py`", para los experimentos 1 y 2; y "`train_exp3.py`" para el experimento 3. En dicho archivo se encuentra la clase `ROLOConcatDataset`, de la cual se ha hablado previamente, y con la cual se crea un conjunto de secuencias con seq_len = 6 frames, cada uno conteniendo `[vector_de_características, bbox_pred]`. Donde, el objetivo de cada secuencia es predecir la ground truth del último frame.
3. **Entrenamiento:** Se entrena la red ROLO con:
   - Entrada: secuencia de vectores `[features + bbox_pred]`, dimensión de 516.
   - Salida: predicción de bounding box, dimensión de 4.
   - Pérdida: se utiliza un Mean Squared Error (MSE) entre la predicción y el ground truth.

   El entrenamiento se realiza con `batch_size = 1`, lo cual permite preservar la información temporal de cada secuencia individual, evitando mezclas entre distintas secuencias que puedan degradar el aprendizaje del modelo LSTM. Además, se realiza el entrenamiento por un total de 10, 30 y 100

épocas, siendo la última la que arrojó mejores resultados para cada experimento (como era de esperarse en una LSTM). Y se guarda el modelo al final del entrenamiento para poder pasar a la parte de evaluación.

4. **Evaluación:** Finalmente, se evalúa el modelo en los videos de test y evaluación (por separado), a través del mismo archivo "`evaluation.py`". En cada paso de tiempo se predice la posición del objeto y se compara con la ground truth, a través de la métrica IoU (Intersection over Union) la cual mide la superposición entre la caja predicha (bounding box) y el ground truth, frame por frame. Cuya fórmula es:

$$\text{IoU} = \frac{\text{Área de intersección}}{\text{Área de unión}} \tag{1}$$

Y de manera global para cada video usamos la métrica AOS (Average Overlap Score), la cual corresponde al promedio del IoU a lo largo de todos los frames del video. De acuerdo al artículo, mide el seguimiento global del objeto y se utiliza comúnmente en benchmarks como OTB. Su fórmula es:

$$\text{AOS} = \frac{1}{N} \sum_{i=1}^{N} \text{IoU}_i \tag{2}$$

Adicionalmente, se considera que un valor de IoU ≥ 0.5 representa una detección/seguimiento aceptable, y un valor de IoU promedio (AOS) alto indica un seguimiento robusto.

¿Qué se espera de ROLO?. Primero, es importante recalcar que ROLO tiene como objetivo detectar y seguir objetos en secuencias de video en tiempo real. Su idea principal es combinar detección y seguimiento en un solo sistema:

- YOLO realiza la detección rápida del objeto y genera una representación semántica (features, bbox) del frame.
- LSTM aprende la evolución temporal del objeto a través de sus features y bounding boxes, permitiendo predecir de forma robusta la ubicación del objeto incluso si la detección falla parcialmente ya sea por oclusiones, desenfoques o movimientos bruscos.

Por lo anterior, se espera que ROLO siga un objeto de forma consistente y suave en el tiempo. Además, interpole o corrija errores de detección usando la memoria de secuencias anteriores. Y finalmente, que sea capaz de trabajar en tiempo real gracias a la eficiencia de YOLO y a una LSTM de bajo costo.

IV. Librerías y ejecutables

En general, cada archivo mencionado anteriormente contiene las librerías necesarias para su correcto funcionamiento con las respectivas versiones. Adicionalmente, se deja una pequeña lista con las librerías más esenciales:

- `torch==2.2.2+cu118`
- `torchaudio==2.2.2+cu118`
- `torchvision==0.17.2+cu118`
- `numpy==1.26.3`
- `tqdm==4.65.2`
- `opencv-python==4.11.0.86`
- `Ultralytics`

Además, cada archivo ejecutable se entrega con su respectiva documentación línea a línea en su interior[1].

V. Resultados

Los resultados a mostrarse fueron obtenidos a partir del dataset OTB-100 disponible en [3]; su ejecución se realizó en una GPU `NVIDIA GeForce RTX 4070 Ti SUPER`, en un sistema operativo Windows y utilizando un entorno de `PyTorch` con soporte para CUDA.

A continuación se presentan los resultados obtenidos para cada experimento:

Experimento 1: Recuérdese que en este caso, solo tenemos 8 videos para prueba. En la Tabla 1 se muestran los resultados obtenidos al variar las épocas de entrenamiento.

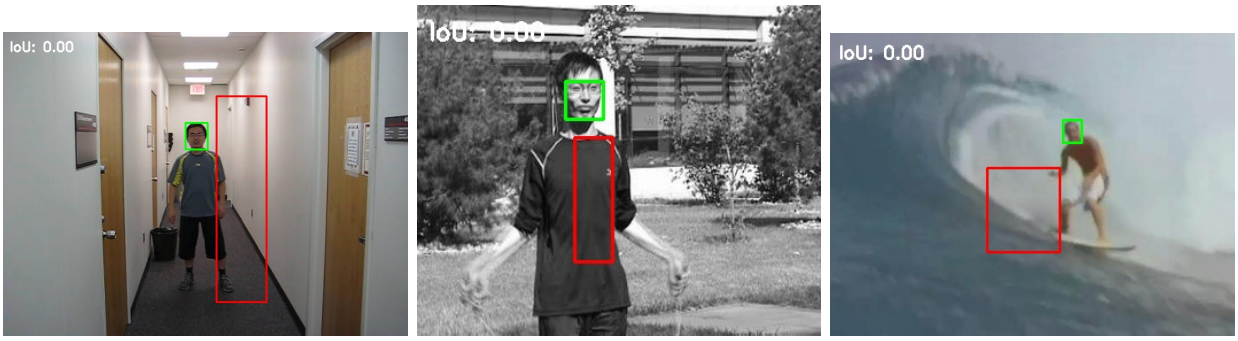
Secuencia	**10 épocas**	**30 épocas**	**100 épocas**
CarScale	0.009	0.020	0.018
Dancer	0.354	0.352	0.254
Dancer2	0.253	0.391	0.199
Gym	0.228	0.244	0.291
Human8	0.049	0.077	0.075
Jump	0.104	0.126	0.067
Skater	0.229	0.154	0.195
Skater2	0.193	0.127	0.220

Tabla 1. Comparación de desempeño (AOS) del experimento 1 para nuestra implementación de ROLO con diferentes épocas de entrenamiento en distintas secuencias.

[1]Código abierto y disponible en https://github.com/StefannyAC/ROLO-PyTorch.git

(A) CarScale (B) Human8 (C) Dancer2

FIGURA 1. Ejemplos de predicciones (en rojo) del experimento 1 en el conjunto de prueba para el modelo de 30 épocas, ordenados de izquierda a derecha de menor a mayor AOS.



(A) Boy (B) Jumping (C) Surfer

FIGURA 2. Ejemplos de predicciones (en rojo) del experimento 1 en el conjunto de evaluación.

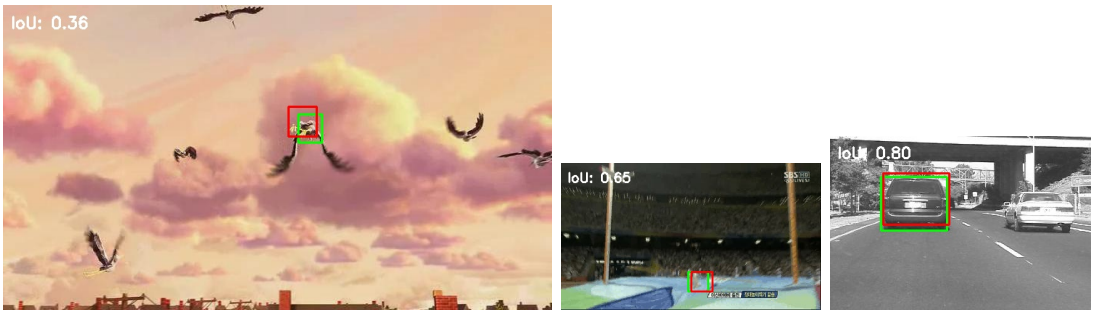
Como se evidenció en los resultados obtenidos y en línea con lo reportado en el artículo original, este experimento muestra una capacidad limitada de generalización (Ver Figura 2). El modelo entrenado con únicamente 22 videos no logra un desempeño robusto al ser evaluado sobre videos no vistos (Ver Figura 1), especialmente cuando se enfrenta a dinámicas no presentes en los datos de entrenamiento. Esta limitación motivó la realización de los siguientes experimentos.

Experimento 2: Recuérdese que, en este caso, tenemos como conjunto de entrenamiento 1/3 de los frames de cada uno de los videos del OTB-30 dataset con sus respectivos ground truths; y como conjunto de prueba todos los frames y ground truths de cada video.

Nuevamente, en la Tabla 2 se presentan los resultados obtenidos por épocas para este experimento.

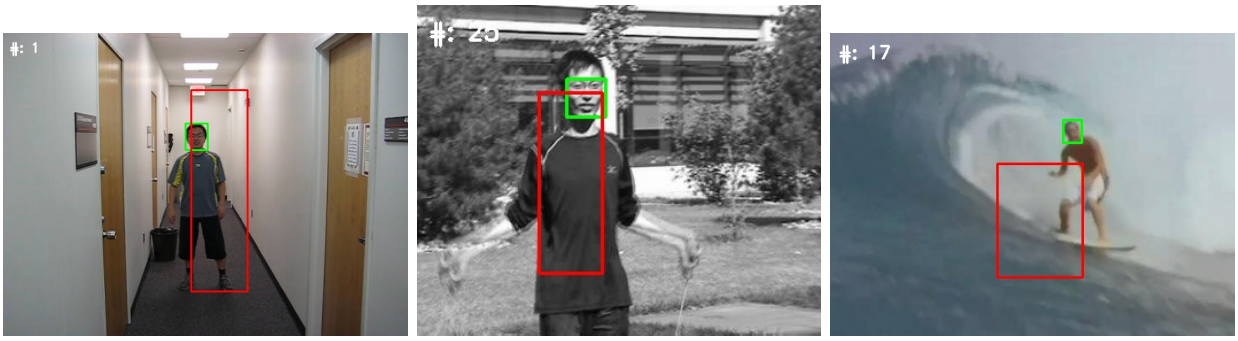
Secuencia	10 épocas	30 épocas	100 épocas
Human2	0.469	0.502	0.699
Human9	0.292	0.412	0.493
Gym	0.503	0.539	0.629
Human8	0.176	0.260	0.579
Skater	0.575	0.618	0.721
SUV	0.504	0.597	0.730
BlurBody	0.487	0.532	0.631
CarScale	0.495	0.646	0.719
Dancer2	0.581	0.641	0.785
BlurCar1	0.401	0.503	0.512
Dog	0.268	0.365	0.409
Jump	0.196	0.234	0.306
Singer2	0.498	0.599	0.657
Woman	0.338	0.393	0.559
David3	0.136	0.246	0.439
Dancer	0.538	0.609	0.712
Human7	0.327	0.422	0.526
Bird1	0.110	0.180	0.293
Car4	0.594	0.733	0.781
CarDark	0.380	0.465	0.589
Couple	0.199	0.267	0.348
Diving	0.435	0.482	0.557
Human3	0.261	0.360	0.529
Skating1	0.393	0.467	0.624
Human6	0.277	0.379	0.485
Singer1	0.541	0.625	0.762
Skater2	0.560	0.589	0.712
Walking2	0.474	0.479	0.666
BlurCar3	0.385	0.507	0.464
Girl2	0.346	0.508	0.669

TABLA 2. Comparación de desempeño (AOS) del experimento 2 para nuestra implementación de ROLO con diferentes épocas de entrenamiento en distintas secuencias.



(A) Bird1 (B) Jump (C) Car4

FIGURA 3. Ejemplos de predicciones (en rojo) del experimento 2 en el conjunto de prueba para el modelo de 100 épocas, ordenados de izquierda a derecha de menor a mayor AOS.



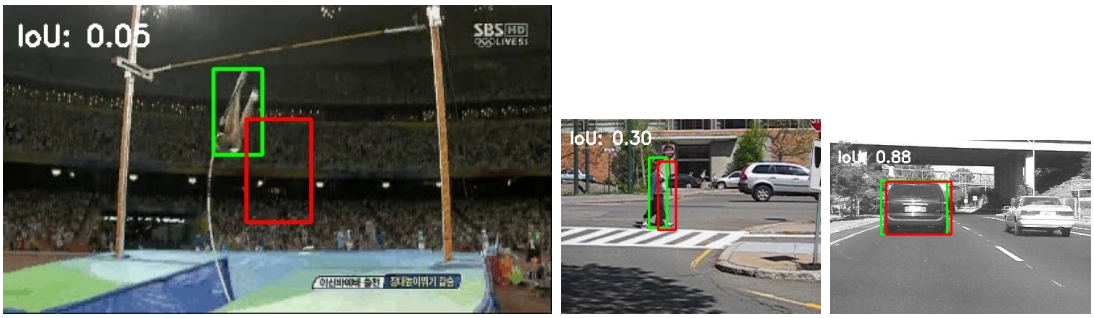
(A) Boy (B) Jumping (C) Surfer

FIGURA 4. Ejemplos de predicciones (en rojo) del experimento 2 en el conjunto de evaluación.

Es importante notar que, aunque en el conjunto de evaluación los ground truths corresponden al rostro del individuo, el modelo ROLO tiende a predecir la ubicación del cuerpo completo (Ver Figura 4). Esto sugiere que el modelo ha aprendido una representación más global del objeto a lo largo del tiempo, priorizando la detección del cuerpo humano en lugar de ajustarse a la región específica del rostro, lo cual sería esperable en un escenario de seguimiento en tiempo real.

Experimento 3: Recuérdese que, en este caso, tenemos como conjunto de entrenamiento todos los frames de cada uno de los videos del OTB-30 dataset con 1/3 de los respectivos ground truths; y como conjunto de prueba todos los frames y ground truths de cada video.

A continuación, se presentan los resultados obtenidos por épocas para este experimento en la Tabla 3.

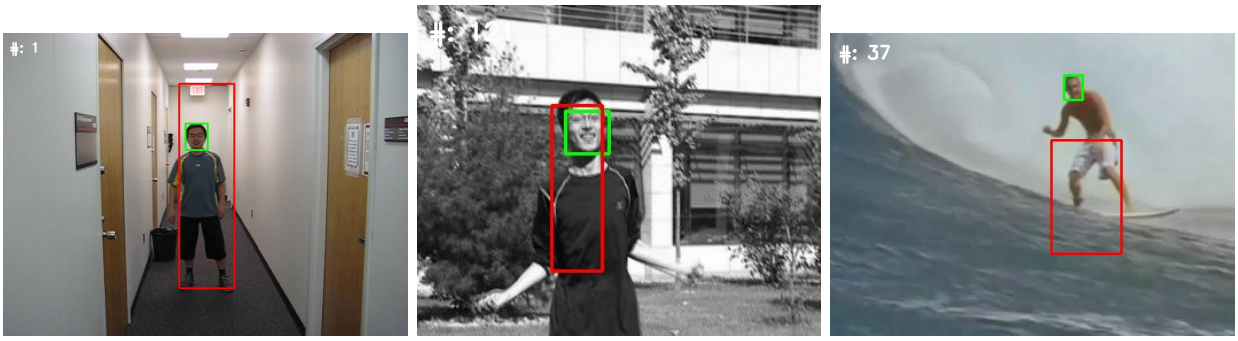


(A) Jump (B) Couple (C) Car4

FIGURA 5. Ejemplos de predicciones (en rojo) del experimento 3 en el conjunto de prueba para el modelo de 100 épocas, ordenados de izquierda a derecha de menor a mayor AOS.

Secuencia	10 épocas	30 épocas	100 épocas
Human2	0.419	0.698	0.812
Human9	0.113	0.519	0.599
Gym	0.429	0.632	0.697
Human8	0.089	0.529	0.652
Skater	0.666	0.716	0.776
SUV	0.622	0.707	0.773
BlurBody	0.526	0.608	0.714
CarScale	0.374	0.757	0.828
Dancer2	0.520	0.791	0.835
BlurCar1	0.492	0.609	0.649
Dog	0.282	0.559	0.539
Jump	0.175	0.298	0.327
Singer2	0.403	0.667	0.756
Woman	0.212	0.576	0.662
David3	0.075	0.432	0.492
Dancer	0.616	0.673	0.768
Human7	0.134	0.567	0.625
Bird1	0.152	0.322	0.460
Car4	0.693	0.811	0.856
CarDark	0.428	0.593	0.701
Couple	0.255	0.393	0.426
Diving	0.414	0.529	0.587
Human3	0.161	0.525	0.653
Skating1	0.303	0.600	0.745
Human6	0.244	0.562	0.648
Singer1	0.335	0.715	0.819
Skater2	0.595	0.727	0.780
Walking2	0.381	0.698	0.737
BlurCar3	0.500	0.598	0.643
Girl2	0.357	0.618	0.784

TABLA 3. Comparación de desempeño (AOS) del experimento 3 para nuestra implementación de ROLO con diferentes épocas de entrenamiento en distintas secuencias.



(A) Boy (B) Jumping (C) Surfer

FIGURA 6. Ejemplos de predicciones (en rojo) del experimento 3 en el conjunto de evaluación.

Comparación con el mejor modelo. De acuerdo a los resultados presentados, se puede concluir que el mejor modelo corresponde al experimento 3 entrenado 100 épocas, por esta razón se cree pertinente presentar la comparación entre dichos resultados y los del artículo original.

A tener en cuenta, se muestra en rojo en la Tabla 4 el mejor resultado.

Secuencia	**ROLO**	**ROLO (nuestro)**
Human2	0.545	0.812
Human9	0.352	0.599
Gym	0.599	0.697
Human8	0.364	0.652
Skater	0.618	0.776
SUV	0.627	0.773
BlurBody	0.519	0.714
CarScale	0.565	0.828
Dancer2	0.627	0.835
BlurCar1	0.537	0.649
Dog	0.429	0.539
Jump	0.547	0.327
Singer2	0.588	0.756
Woman	0.649	0.662
David3	0.622	0.492
Dancer	0.755	0.768
Human7	0.456	0.625
Bird1	0.362	0.460
Car4	0.768	0.856
CarDark	0.674	0.701
Couple	0.464	0.426
Diving	0.659	0.587
Human3	0.568	0.653
Skating1	0.572	0.745
Human6	0.532	0.648
Singer1	0.653	0.819
Skater2	0.652	0.780
Walking2	0.595	0.737
BlurCar3	0.539	0.643
Girl2	0.517	0.784

TABLA 4. Comparación de desempeño (AOS) entre ROLO original y nuestra implementación de 100 épocas en distintas secuencias.

VI. Predicción mediante mapas de calor (heatmaps)

A diferencia de las versiones anteriores de ROLO que predicen directamente las coordenadas del *bounding box* (`ROLO-coordinate`), en este caso se implementa la variante `ROLO-heatmap`, en la cual la salida de la LSTM es un **vector de 1024 dimensiones** que representa un *mapa de calor* (*heatmap*) sobre una grilla de 32×32 celdas.

Ventajas del enfoque por heatmaps:

- Permite representar incertidumbre espacial (por ejemplo, cuando el objeto se mueve rápidamente o está parcialmente ocluido).
- Puede aprender una distribución más suave del movimiento en lugar de predecir un punto exacto.
- Visualmente, ofrece una interpretación clara del enfoque del modelo en cada frame.

A continuación, se muestra en la Figura 7 una secuencia de frames consecutivos con sus respectivos mapas de calor superpuestos, en los que se observa cómo el modelo sigue el objeto a lo largo del tiempo, con el ground truth en verde y podemos observar como predice incluso en situaciones de oclusión:

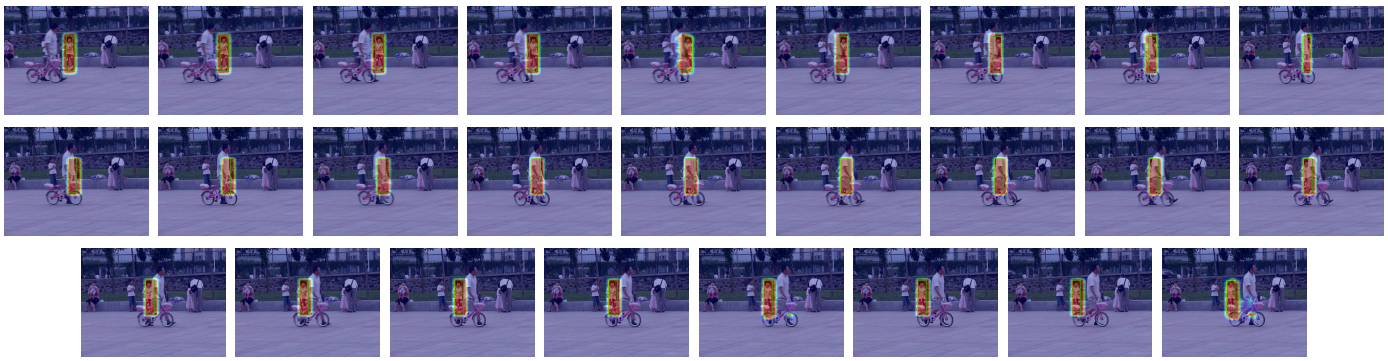


Figura 7. Secuencia de 26 predicciones consecutivas representadas como mapas de calor (Girl2).

VII. Conclusiones

- En el trabajo original de ROLO se utilizó YOLO v2, el cual fue preentrenado en ImageNet y ajustado en Pascal VOC, lo que le permitió generar detecciones más robustas y coherentes cuadro a cuadro, incluso en condiciones desafiantes (oclusiones, desenfoques, escalas cambiantes).
- Dado que se usó YOLOv8s original sin preentrenamiento adicional, se observa que este tiende a fallar en detectar los objetos de interés en varios frames consecutivos, especialmente en secuencias con pocas características distintivas del objeto (por ejemplo, “Jump”, “Couple” que tienen un bajo IoU), esto podría mitigarse realizando un pre-entrenamiento.
- El desempeño general de nuestra implementación es comparable al trabajo original (AOS prom 0.6781 vs 0.565), a pesar de utilizar YOLOv8s como extractor sin ajuste fino (fine-tunning).

- El modelo LSTM fue capaz de aprender secuencias temporales efectivamente, lo cual se refleja en la mejora del desempeño en muchas de las secuencias, incluso cuando las detecciones fallaban parcialmente.
- El hecho de haber alcanzado un mejor desempeño global sin usar técnicas de ajuste fino (fine-tuning) sobre las secuencias de entrenamiento sugiere que ROLO aún es competitivo como base para desarrollos más modernos en seguimiento visual.

Referencias

[1] Guanghan Ning. ROLO: Recurrent YOLO for object tracking. https://github.com/Guanghan/ROLO, 2016.

[2] Guanghan Ning, Zhi Zhang, Chen Huang, Zhihai He, Xiaobo Ren, and Haohong Wang. Spatially supervised recurrent convolutional neural networks for visual object tracking. *arXiv preprint arXiv:1607.05781*, 2016.

[3] ZLY1402875051. OTB2015: Object Tracking Benchmark (100-sequence version). https://www.kaggle.com/datasets/zly1402875051/otb2015/data, 2025. Dataset descargado el 18 de mayo de 2025.