Tecnológico de Monterrey

Implementación de un modelo de deep learning

IA.

Raymundo Ivan Diaz Alejandre A01735644

1 de Diciembre del 2025

Inteligencia artificial avanzada para la ciencia de datos II

Introducción.

Implementar eficaz de un modelo de Deep Learning capaz de reconocer acciones humanas en videos del dataset UCF101.

En lugar de procesar directamente los píxeles del video, cosa que es muy costosa, se utilizan secuencias de puntos clave del cuerpo (*keypoints*) por frame. Cada video se representa como una secuencia de coordenadas (x,y) para J articulaciones.

Este enfoque reduce los recursos necesarios para obtener los resultados, haciendo que el entrenamiento sea manejable con recursos limitados sin sacrificar la capacidad de capturar patrones de movimiento.

Datos y preparación.

Dataset UCF101 Skeletons

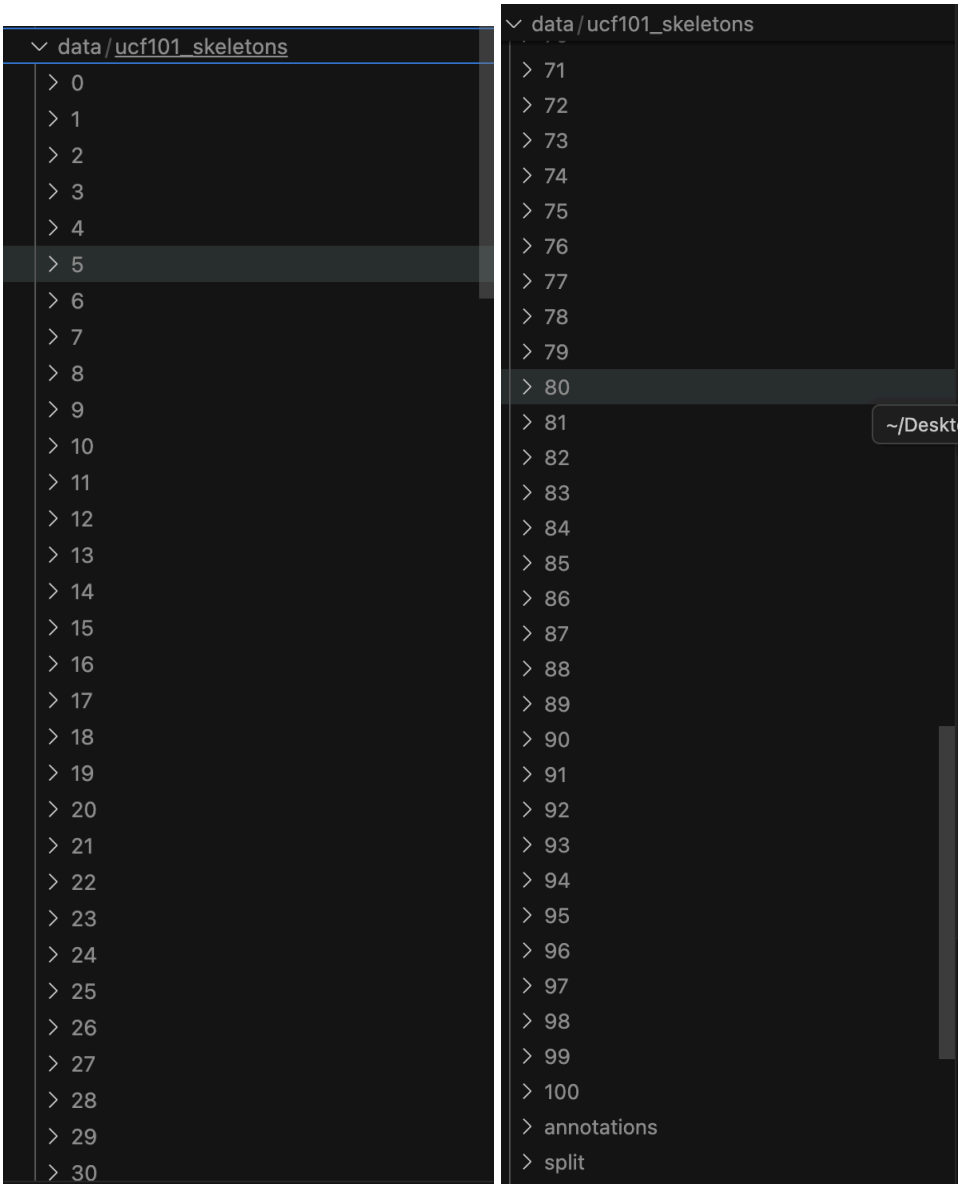
Se utilizaron las anotaciones de esqueletos 2D de UCF101 contenidas en el archivo ucf101_2d.pkl.

El git no los pudo contener por el tamaño (1gb+), pero básicamente se uso el de la liga de la actividad pero se dividió por divisiones realizados con el siguiente comando de terminal:

(La forma de correr el programa es más clara, concisa y directa en el README)

```
python prepare_data.py
```

Para optimizar la carga de datos en PyTorch, se procesó esta carpeta la cual divide en otras carpetas generando una estructura de directorios organizada por clase:



Cada archivo .pkl individual contiene un diccionario simplificado: {"data": array_de_keypoints} con forma aproximada [T, J, 2]. Esta estructura permite restringir o ampliar el subconjunto de clases fácilmente desde la configuración (configs/default.yaml).

Preprocesamiento de Esqueletos

Antes de ingresar al modelo, los datos pasan por transformaciones implementadas en src/transforms.py:

Ajuste de longitud temporal:

-Garantiza que todas las secuencias tengan una longitud fija (la cual es necesaria para el procesamiento). Si el video es más largo, se acorta, si es más corto, se repite cíclicamente.

Centrado y Normalización de Escala:

Después, se normaliza la postura para que el modelo se enfoque en cómo se mueve la persona.

Para lograr esto se restó la coordenada de la pelvis a todas las articulaciones, centrando el movimiento en (0,0) y despues se normalizo el tamaño del esqueleto la cual se basaba en la distancia media de articulaciones clave, como por ejemplo vendrían siendo hombro-cadera, etc, para que el modelo pueda aprender patrones de movimiento y el tamaño dependiendo los distintos ángulos no afecten tanto el resultado.

Modelos Implementados

Se usaron 2 modelos, los cuales se encuentran en la carpeta src/models. Estos BaselineSTM y ImprovedLSTM. La idea de esto fue empezar con una arquitectura sencilla para posteriormente construir una versión mejor que redujera el sobreajuste

BaselineLSTM.

Es una red LSTM clásica que toma como entrada la secuencia completa de articulaciones ya preprocesadas. La red procesa la secuencia frame por frame y, al final se obtiene una representación del movimiento. Esa representación se pasa por una capa conectada que da la predicción final.

Este es algo simple pero estable y muy útil para verificar si el dataset y el cómo se interpretan los datos de preprocesamiento son correctos.

ImprovedLSTM.

Mantiene la misma estructura general pero incorpora mejoras importantes. El objetivo es darle al modelo un poco más de capacidad de generalización y reducir la memorización secuencias específicas del entrenamiento.

Ajustes realizados: Modelo ImprovedLSTM

A diferencia del Baseline (que sufría de sobreajuste rápido), este modelo incorpora técnicas de regularización para mejorar la generalización:

1. **Dropout (0.4):** Se añadió una capa de *dropout* con probabilidad p=0.4 después de la LSTM y antes de la capa lineal. Esto ayuda a apagar de forma aleatoria neuronas durante el entrenamiento, para que la red aprenda de forma más distribuida, evitando que dependa excesivamente de ciertos nodos.
2. **Batch Normalization (1D):** Funciona como un estabilizador para la salida de la LSTM. Esta se encarga de limpiar y equilibrar los datos antes de que pasen a la siguiente etapa, de esta forma el entrenamiento es más fluido, el modelo aprende a mayor velocidad y es menos propenso a fallar por una configuración inicial inestable.

Disponibilidad de Checkpoints

Para garantizar la reproducibilidad y permitir la verificación de estos resultados sin necesidad de re-entrenar, se han incluido los pesos pre-entrenados de ambos modelos en el repositorio entregado:

-Baseline: runs/baseline/best.pt

-Mejorado: runs/improved/best.pt

En (src/infer.py) se configurado para cargar cualquiera de estos checkpoints y generar predicciones en consola sobre nuevos datos.

Entrenamiento.

Comando de ejecución:

python3 -m src.train --config configs/default.yaml --model baseline

```
python3 -m src.train --config configs/default.yaml --model baseline
```

```
raymundodiaz@192 skeleton_acts % python3 -m src.train --config configs/default.yaml --model baseline

[Ep 001] train: loss=1.5643 acc=0.315 f1=0.210 | val: loss=1.4310 acc=0.567 f1=0.476
  -> saved best to runs/baseline/best.pt (f1=0.476)
[Ep 002] train: loss=1.3870 acc=0.493 f1=0.415 | val: loss=1.2008 acc=0.591 f1=0.527
  -> saved best to runs/baseline/best.pt (f1=0.527)
[Ep 003] train: loss=1.1247 acc=0.547 f1=0.451 | val: loss=1.0235 acc=0.591 f1=0.523
[Ep 004] train: loss=0.9147 acc=0.634 f1=0.561 | val: loss=0.8811 acc=0.638 f1=0.573
  -> saved best to runs/baseline/best.pt (f1=0.573)
[Ep 005] train: loss=0.7605 acc=0.662 f1=0.612 | val: loss=1.0690 acc=0.567 f1=0.492
[Ep 006] train: loss=0.9666 acc=0.607 f1=0.561 | val: loss=0.9085 acc=0.591 f1=0.580
  -> saved best to runs/baseline/best.pt (f1=0.580)
[Ep 007] train: loss=0.9780 acc=0.542 f1=0.491 | val: loss=0.8990 acc=0.606 f1=0.581
  -> saved best to runs/baseline/best.pt (f1=0.581)
[Ep 008] train: loss=0.8290 acc=0.673 f1=0.657 | val: loss=0.8210 acc=0.654 f1=0.598
  -> saved best to runs/baseline/best.pt (f1=0.598)
[Ep 009] train: loss=0.7533 acc=0.636 f1=0.574 | val: loss=0.8366 acc=0.661 f1=0.621
  -> saved best to runs/baseline/best.pt (f1=0.621)
[Ep 010] train: loss=0.7621 acc=0.638 f1=0.578 | val: loss=0.8543 acc=0.685 f1=0.671
  -> saved best to runs/baseline/best.pt (f1=0.671)
[Ep 011] train: loss=0.7920 acc=0.694 f1=0.674 | val: loss=0.7884 acc=0.693 f1=0.672
  -> saved best to runs/baseline/best.pt (f1=0.672)
[Ep 012] train: loss=0.7741 acc=0.704 f1=0.674 | val: loss=0.7743 acc=0.685 f1=0.671
[Ep 013] train: loss=0.6417 acc=0.756 f1=0.757 | val: loss=0.7813 acc=0.646 f1=0.638
[Ep 014] train: loss=0.6785 acc=0.741 f1=0.739 | val: loss=0.6886 acc=0.772 f1=0.766
  -> saved best to runs/baseline/best.pt (f1=0.766)
[Ep 015] train: loss=0.5759 acc=0.772 f1=0.755 | val: loss=0.7824 acc=0.677 f1=0.646
raymundodiaz@192 skeleton_acts %
Ln 140, Col 1  Spaces: 4  UTF-8  LF  {} Python  Finish Setup  Python 3.13.2 (homebrew)
```

Entrenamiento con Modelo ImprovedLSTM.

Comando: python3 -m src.train --config configs/default.yaml --model improved --dropout 0.4

```
raymundodiaz@192 skeleton_acts % python3 -m src.train --config configs/default.yaml --model improved --dropout 0.4
[Ep 001] train → loss=1.6005 acc=0.348 f1=0.321 | val → loss=1.4373 acc=0.559 f1=0.490
Nuevo mejor modelo (F1=0.490) → runs/improved/best.pt
[Ep 002] train → loss=1.0227 acc=0.578 f1=0.542 | val → loss=1.2319 acc=0.606 f1=0.579
Nuevo mejor modelo (F1=0.579) → runs/improved/best.pt
[Ep 003] train → loss=0.8480 acc=0.648 f1=0.627 | val → loss=1.0172 acc=0.669 f1=0.660
Nuevo mejor modelo (F1=0.660) → runs/improved/best.pt
[Ep 004] train → loss=0.7880 acc=0.702 f1=0.687 | val → loss=0.8660 acc=0.654 f1=0.639
[Ep 005] train → loss=0.7456 acc=0.706 f1=0.691 | val → loss=0.8246 acc=0.654 f1=0.639
[Ep 006] train → loss=0.6734 acc=0.729 f1=0.718 | val → loss=0.7171 acc=0.717 f1=0.699
Nuevo mejor modelo (F1=0.699) → runs/improved/best.pt
[Ep 007] train → loss=0.6572 acc=0.764 f1=0.754 | val → loss=0.6662 acc=0.732 f1=0.723
Nuevo mejor modelo (F1=0.723) → runs/improved/best.pt
[Ep 008] train → loss=0.6431 acc=0.793 f1=0.783 | val → loss=0.6735 acc=0.748 f1=0.737
Nuevo mejor modelo (F1=0.737) → runs/improved/best.pt
[Ep 009] train → loss=0.4999 acc=0.803 f1=0.796 | val → loss=0.7202 acc=0.756 f1=0.753
Nuevo mejor modelo (F1=0.753) → runs/improved/best.pt
[Ep 010] train → loss=0.4697 acc=0.822 f1=0.816 | val → loss=0.9432 acc=0.701 f1=0.687
[Ep 011] train → loss=0.4722 acc=0.785 f1=0.781 | val → loss=0.7153 acc=0.669 f1=0.666
[Ep 012] train → loss=0.4700 acc=0.822 f1=0.821 | val → loss=0.6452 acc=0.732 f1=0.722
[Ep 013] train → loss=0.3606 acc=0.839 f1=0.832 | val → loss=0.5194 acc=0.811 f1=0.808
Nuevo mejor modelo (F1=0.808) → runs/improved/best.pt
[Ep 014] train → loss=0.3132 acc=0.872 f1=0.869 | val → loss=0.6455 acc=0.787 f1=0.783
[Ep 015] train → loss=0.2855 acc=0.884 f1=0.881 | val → loss=0.6239 acc=0.764 f1=0.752
```

Resultados.

La evaluación formal se realiza ejecutando:

python3 -m src.eval --config configs/default.yaml --ckpt runs/baseline/best.pt

```
python3 -m src.eval --config configs/default.yaml --ckpt runs/baseline/best.pt
```

Resultados (Baseline)
En una ejecución representativa, el modelo base alcanzó:

-Validation Accuracy: 77%

-Macro-F1: 76%

```
raymundodiaz@192 skeleton_acts % python3 -m src.eval --config configs/default.yaml --ckpt runs/baseline/best.pt

Val Accuracy: 0.7717 | Macro-F1: 0.7661
Saved confusion matrix to: runs/baseline/confusion_matrix.png
```

En las pruebas, el modelo BaselineLSTM obtuvo alrededor de 77% de accuracy y un Macro-F1 cercano de 76%.

Esto significa que el modelo identifica la acción correcta en la mayoría de los casos, y también a la vez mantiene un desempeño equilibrado entre todas las clases, incluso aquellas menos frecuentes.

Resultados (ImprovedLSTM)

Comando: python3 -m src.eval --config configs/default.yaml --ckpt runs/improved/best.pt

Y para el modelo mejorado (con Dropout y Batch Normalization) bajo las mismas condiciones de validación

-Validation Accuracy: 81.10%

-Macro-F1: 80.83%

```
raymundodiaz@192 skeleton_acts % python3 -m src.eval --config configs/default.yaml --ckpt runs/improved/best.pt
Val Accuracy: 0.8110 | Macro-F1: 0.8083
Saved confusion matrix to: runs/improved/confusion_matrix.png
```

Comparativa de Desempeño:

Modelo	**Accuracy**	**Macro-F1**	**Diferencia**
BaselineLSTM	77.17%	76.61%	—
ImprovedLSTM	**81.10%**	**80.83%**	**+3.93%**

Los ajustes sin duda fueron efectivos, el uso de dropout redujo el sobreajuste, lo cual permitió que el modelo generalizara mejor los datos no vistos.

Predict.

Con el siguiente comando (Baseline):

```
python3 -m src.infer \
  --config configs/default.yaml \
  --ckpt runs/baseline/best.pt \
  --input data/ucf101_skeletons/0/0_v_ApplyEyeMakeup_g01_c01.pkl
```

Improved:

```
python3 -m src.infer
```

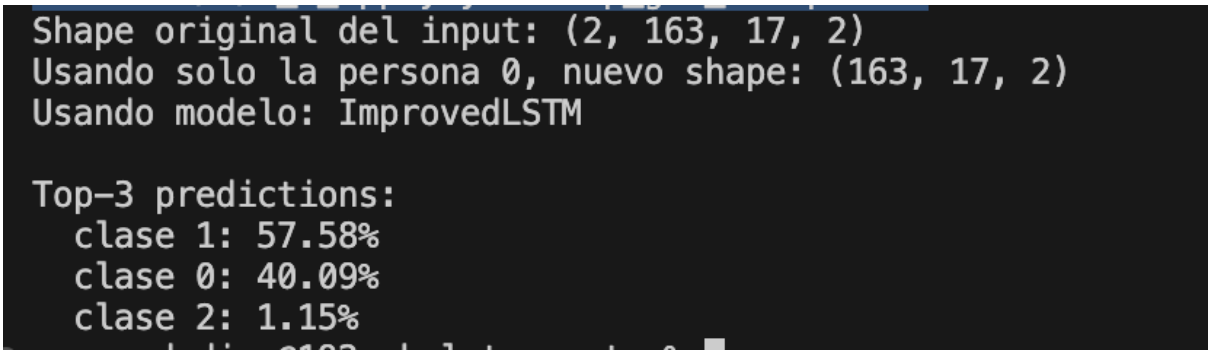
--config configs/default.yaml

--ckpt runs/improved/best.pt

--input data/ucf101_skeletons/0/0_v_ApplyEyeMakeup_g01_c01.pkl

Baseline:

Improved



Donde Shape (a,b,c,d) =

a = número de personas detectadas

b= número de frames

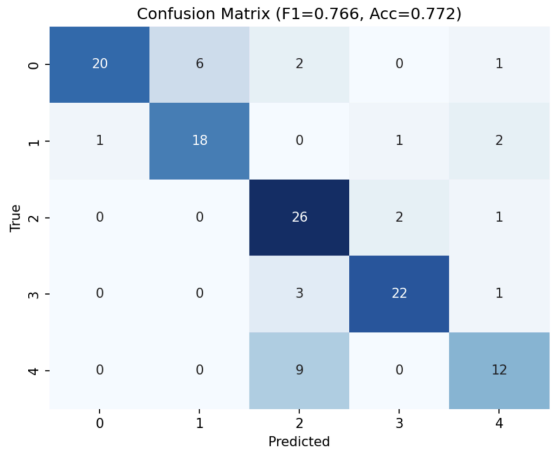
c = número de articulaciones (formato COCO)

d = coordenadas (x, y)

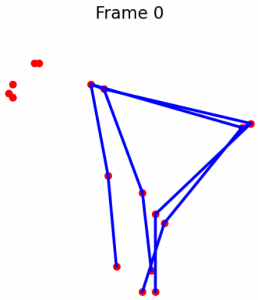
Y para la clasificación solo se usa una persona (persona 0), lo cual shape se convierte a (b,c,d) y es el esqueleto el cual se entrena al modelo.

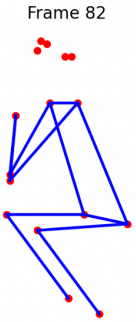
Y Top-3 predictions nos da la probabilidad de que el video dado pertenezca a tal clase, donde la clase 1 es la más probable.

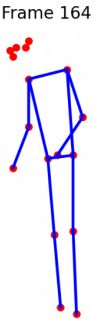
Igual se generó una matriz de confusión, que resulta muy útil y facilita ver que es lo que está haciendo el modelo. Al visualizarla permite notar si hay grupos de acciones que tienen parecido entre sí y tienden a confundirse, por ejemplo, actividades con patrones de movimiento de brazos similares.



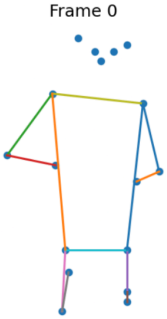
Skeleton Visualization
20_v_CleanAndJerk_g12_c01.pkl

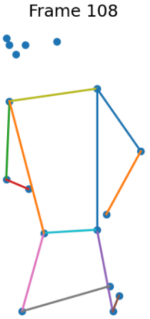


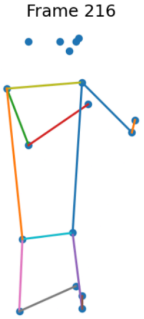


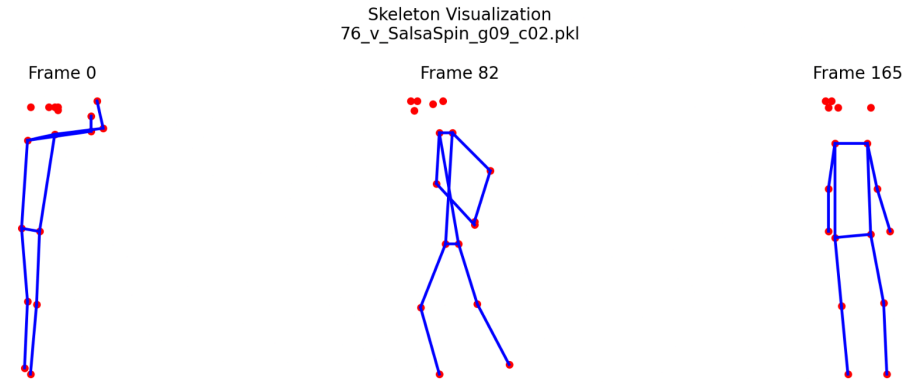


Skeleton visualization for sample:
100_v_YoYo_g23_c05.pkl









Ejemplos random de 3 frames de un esqueleto del dataset, visualize_skeleton.py. Los cuales elige un .pkl al azar y los representa, este normaliza las dimensiones para una escala facil de dimensionar, mientras para elegir los 3 frames de toda la secuencia escoge el inicio, mitad y final.

Permite inspeccionar que la estructura de los esqueletos tienen coherencia y los archivos .pkl se leen correctamente.