Facultad de Ingeniería e Informática

Ingeniería en Informática

PROYECTO FINAL DE GRADO

Evaluación Automatizada de Técnica en Ejercicios de Fitness mediante Aprendizaje Profundo y Visión por Computadora

Autor: Ignacio Villanueva

Tutor: Mg. Lic. Lorena Talamé

Salta, Argentina
2026

HOJA DE APROBACIÓN

Presidente del Tribunal

Vocal 1

Vocal 2

Fecha de Exposición: _____

Agradecimientos

A mis padres, Silvia y Gabriel, por volcar todos sus esfuerzos en que podamos forjar herramientas y vivir una mejor vida. Por el cariño de siempre.

A mi hermano, Mateo, por ser mi par en este camino desde que tengo memoria y compartir la misma neurona.

A Agustina, por su presencia incondicional durante toda la carrera y el resto de nuestras vidas.

A Luciana, por empujarme a ser mi mejor versión. Por ayudarme a encontrarla. Por el bilateralismo elegido y compartido.

A aquellos a quienes pertenecen los rostros que aparecen en mi mente cuando pienso en la palabra *amigo*. Por las incontables formas en que me han apoyado a lo largo de estos años.

A aquellos profesores que me inspiraron durante el camino recorrido a ser un humano íntegro y mejor profesional, y me heredaron la vocación de instruir y entender.

A la Universidad Católica de Salta, por la formación, el espacio y las oportunidades brindadas. A POSTECH y la Universidad de Cádiz, que me cambiaron la forma de ver el mundo y brindaron el conocimiento que desarrolló este Trabajo Final.

A la educación, por ser la herramienta de movilidad social ascendente que nos queda.

A mi nación, Argentina, por absolutamente todo.

Índice General

Agradecimientos I

Resumen VI

Abstract VII

1 Introducción 1
- 1.1 Objetivos . . . 1
  - 1.1.1 Objetivo General . . . 1
  - 1.1.2 Objetivos Específicos . . . 1
- 1.2 Motivación . . . 1
- 1.3 Alcance . . . 2

2 Marco Teórico 3
- 2.1 Fundamentos de Inteligencia Artificial y Aprendizaje Automático . . . 3
- 2.2 Visión por Computadora y Redes Convolucionales . . . 8
- 2.3 Arquitectura Transformer y Vision Transformers . . . 12
- 2.4 Estimación de Poses Humanas . . . 15
- 2.5 Procesamiento Temporal y Reconocimiento de Acciones . . . 18
- 2.6 Transfer Learning . . . 21

3 Desarrollo e Implementación 22
- 3.1 Arquitectura Inicial del Sistema . . . 22
- 3.2 Dataset . . . 22
- 3.3 Módulo Estimador de Pose . . . 23
  - 3.3.1 Estructura de Salida (JSON) . . . 24
- 3.4 Preparación de Keypoints para Entrenamiento . . . 26
  - 3.4.1 Iteraciones de Desarrollo: Selección del Sujeto Principal . . . 26
  - 3.4.2 Pipeline de Procesamiento de Datos . . . 29
- 3.5 Proceso Iterativo de Entrenamiento del Modelo . . . 31
  - 3.5.1 Experimento Base: Identificación del Problema de Overfitting . . . 31
  - 3.5.2 Iteración con Transfer Learning con Fine-Tuning . . . 35
  - 3.5.3 Optimización Avanzada: Aumentación de Datos y AdamW . . . 36
- 3.6 Análisis de Costos y Factibilidad . . . 38
  - 3.6.1 Costos del Proyecto Académico . . . 38
  - 3.6.2 Especificaciones de Hardware para Replicación . . . 39
  - 3.6.3 Comparación con Soluciones Comerciales . . . 40
  - 3.6.4 Estimación para Implementación Comercial . . . 41
  - 3.6.5 Análisis de Factibilidad . . . 43

4 Resultados 46
- 4.1 Progreso Iterativo del Modelo y Métricas . . . 46
  - 4.1.1 Experimento Base . . . 47
  - 4.1.2 Iteración con Transfer Learning con Fine-Tuning . . . 48

4.1.3 Configuración Final 50
4.1.4 Análisis Detallado de Clasificación 51
4.2 Discusión 52

5 Conclusiones y Trabajo Futuro **54**
5.1 Cumplimiento de Objetivos y Líneas Futuras 54
5.2 Reflexión Final 56

Referencias **58**

Índice de Figuras

1 Diagrama de una neurona artificial mostrando entradas ponderadas, unidad de suma y función de activación. Fuente: Adaptado de GeeksforGeeks. 4
2 Operación de convolución 2D: un *kernel* se aplica sobre una matriz de entrada para generar un mapa de características. Fuente: Adaptado de Goodfellow et al. (2016). 9
3 Aplicación de distintos filtros de convolución sobre una misma imagen: detección de bordes horizontales, verticales, operadores Sobel, Laplaciano y Canny. Fuente: Adaptado de Codingwalks. 10
4 Bloque residual de *ResNet*. La entrada x atraviesa las capas convolucionales produciendo F(x), mientras la conexión de identidad (*skip connection*) preserva x para sumarse a la salida. 12
5 Arquitectura del *Vision Transformer (ViT)*: división de imagen en parches, proyección lineal, *embeddings* posicionales y procesamiento mediante *Transformer Encoder*. 15
6 Los 17 *keypoints* del formato estándar *COCO* para poses humanas, con índices numéricos y nombres anatómicos. Fuente: Adaptado de documentación de *COCO*. 16
7 Pipeline de *OpenPose*: (a) imagen de entrada, (b) mapas de confianza de *keypoints*, (c) *Part Affinity Fields*, (d) emparejamiento bipartito, (e) esqueletos resultantes. 17
8 Convolución 3D: un *kernel* tridimensional (3×3×3) opera sobre un volumen espacio-temporal. 19
9 Grafo espacio-temporal de *ST-GCN*: nodos representando articulaciones conectadas espacialmente (dentro de cada *frame*) y temporalmente (entre *frames* consecutivos). 20

10 Flujo de trabajo del *pipeline* de clasificación. 22
11 Arquitectura detallada del *pipeline* implementado. 31

12 Curva de pérdida del experimento base. 47
13 Curva de precisión del experimento base. 48
14 Curva de pérdida con *transfer learning* y *fine-tuning*. 49
15 Curva de precisión con *transfer learning* y *fine-tuning*. 49
16 Curva de precisión de la configuración final. 50
17 Curva de pérdida de la configuración final. 51
18 Matriz de confusión del modelo final en conjunto de validación. 51

Índice de Tablas

1 Costos del proyecto académico. 39
2 Especificaciones de hardware recomendadas. 39
3 Comparación con soluciones comerciales. 40
4 Inversión en hardware para implementación comercial. 41
5 Estimación de recursos humanos para implementación comercial. 41

6 Comparación de configuraciones experimentales y métricas de performance. . . 46
7 Métricas de clasificación por clase en conjunto de validación. 52

Resumen

La evaluación objetiva de técnica en ejercicios de entrenamiento físico representa un desafío fundamental debido a la subjetividad inherente del análisis manual y los costos prohibitivos de sistemas comerciales de captura de movimiento. Este trabajo presenta el desarrollo de un modelo de clasificación de técnica en el ejercicio de *press* de hombro, mediante técnicas de visión por computadora y aprendizaje profundo. Se toma el *press* militar como caso de estudio, combinando *Vision Transformers* para estimación de poses con redes convolucionales 3D para clasificación temporal.

El *pipeline* desarrollado procesa videos del ejercicio en dos etapas: extracción de puntos articulares (*keypoints*) 2D utilizando *ViTPose* sobre 17 puntos anatómicos del estándar *COCO*, seguido de análisis temporal a través de *PoseConv3D* que clasifica secuencias de movimiento en tres categorías (técnica correcta, errores de codos, errores de rodillas). El desarrollo de un algoritmo de corrección de seguimiento (*tracking*) representa una contribución técnica del proyecto, resolviendo inconsistencias de identificación en videos multi-persona mediante proximidad espacial.

La metodología experimental siguió un desarrollo iterativo basado en evidencia, progresando desde un experimento base con sobreajuste severo hasta la implementación final optimizada mediante *transfer learning* (aprendizaje por transferencia) con *fine-tuning* (ajuste fino) desde modelos preentrenados en *NTU60*. El modelo final alcanza *F1-score* macro de 0.91 y precisión de 91.6 % en el *dataset Fitness-AQA*, demostrando la efectividad del enfoque multi-clase unificado comparado con aproximaciones binarias especializadas.

Los resultados validan la viabilidad de modelos de clasificación biomecánica basados en análisis visual, ofreciendo una alternativa accesible a tecnologías comerciales disponibles. La arquitectura desarrollada establece un marco de trabajo replicable para análisis automatizado de movimiento humano, con aplicaciones potenciales en entrenamiento físico, rehabilitación y deporte.

Palabras clave: estimación de poses, visión por computadora, aprendizaje profundo, clasificación de movimiento, *press* de hombro, evaluación de técnica.

Abstract

The objective evaluation of technique in physical training exercises represents a fundamental challenge due to the inherent subjectivity of manual analysis and the prohibitive costs of commercial motion capture systems. This work presents the development of a technique classification model for the overhead press exercise, using computer vision and deep learning techniques. The overhead press is used as a case study, combining Vision Transformers for pose estimation with 3D convolutional networks for temporal classification.

The developed pipeline processes exercise videos in two stages: extraction of 2D joint keypoints using ViTPose over 17 anatomical points of the COCO standard, followed by temporal analysis through PoseConv3D that classifies motion sequences into three categories (correct technique, elbow errors, knee errors). The development of a tracking correction algorithm represents a technical contribution of the project, resolving identification inconsistencies in multi-person videos through spatial proximity.

The experimental methodology followed an evidence-based iterative development, progressing from a baseline experiment with severe overfitting to the final optimized implementation using transfer learning with fine-tuning from models pretrained on NTU60. The final model achieves a macro F1-score of 0.91 and 91.6 % accuracy on the Fitness-AQA dataset, demonstrating the effectiveness of the unified multi-class approach compared to specialized binary approaches.

The results validate the viability of biomechanical classification models based on visual analysis, offering an accessible alternative to available commercial technologies. The developed architecture establishes a replicable framework for automated human motion analysis, with potential applications in physical training, rehabilitation, and sport.

Keywords: pose estimation, computer vision, deep learning, motion classification, overhead press, technique assessment.

Capítulo 1

Introducción

1.1 Objetivos

1.1.1 Objetivo General

Desarrollar un modelo de clasificación de técnica en el ejercicio de *press* de hombro mediante técnicas de visión por computadora y aprendizaje profundo, capaz de identificar y categorizar errores de ejecución.

1.1.2 Objetivos Específicos

1. Conformar un conjunto de videos de entrenamiento que sea útil para entrenar e implementar la solución propuesta al ejercicio seleccionado.
2. Construir un *pipeline* de extracción y transformación de información biomecánica que convierta videos crudos de ejecución de ejercicios en conjuntos de datos estructurados compatibles con modelos de aprendizaje profundo para clasificación de acciones.
3. Entrenar y optimizar iterativamente modelos de clasificación de acciones hasta alcanzar un desempeño robusto en la evaluación automatizada de la forma en ejercicios.
4. Evaluar el desempeño del modelo desarrollado utilizando métricas estándar de clasificación y protocolos de validación que aseguren capacidad de generalización.

1.2 Motivación

La evaluación de la forma física y la técnica de ejercicio representa un desafío fundamental en el ámbito del entrenamiento físico y la rehabilitación. Muchas personas realizan entrenamientos sin supervisión profesional adecuada, lo que puede llevar a lesiones por ejecución incorrecta de ejercicios y limitar la efectividad del entrenamiento. Incluso con la guía de entrenadores, la evaluación manual de la forma puede ser inconsistente y subjetiva, dependiendo de la experiencia y criterio individual del profesional.

En este contexto, existe una oportunidad de aplicar tecnologías de procesamiento de imágenes y análisis de movimiento para desarrollar herramientas que permitan obtener información objetiva y específica sobre la ejecución de ejercicios. Las soluciones comerciales disponibles en el mercado, como sistemas especializados de análisis biomecánico, pueden resultar costosas e inaccesibles para muchas instituciones educativas, centros de rehabilitación y gimnasios.

Los avances recientes en visión por computadora y aprendizaje profundo han habilitado nuevas capacidades para análisis automatizado de movimiento humano, reduciendo la complejidad de su aplicación. El desarrollo de técnicas precisas para detección de poses corporales, combinado con métodos de análisis temporal de secuencias de movimiento, permite ahora extraer y clasificar patrones biomecánicos complejos directamente desde video estándar. La dis-

ponibilidad de estas tecnologías mediante *frameworks* de código abierto democratiza el acceso a herramientas de análisis de movimiento que tradicionalmente requerían sistemas comerciales especializados, abriendo posibilidades para el desarrollo de soluciones accesibles y adaptables a contextos educativos, de rehabilitación y entrenamiento deportivo, particularmente relevantes en instituciones con recursos limitados.

El desarrollo de soluciones basadas en tecnologías de código abierto reduce los costos de implementación y permite mayor flexibilidad y personalización para adaptarse a necesidades específicas de diferentes contextos de aplicación. Esto es particularmente relevante en el contexto educativo y de investigación, donde la posibilidad de modificar y extender las herramientas es fundamental para la experimentación y adaptación a casos de uso específicos.

Si bien las técnicas de análisis automatizado de movimiento son aplicables a diversos tipos de ejercicios, este proyecto busca establecer y validar una metodología sistemática utilizando el *press* de hombro como caso de estudio. Esta decisión de enfoque permite desarrollar un análisis profundo de los desafíos técnicos y biomecánicos específicos, validar rigurosamente el enfoque propuesto, y documentar un proceso reproducible que sirva como base para desarrollos posteriores orientados a la asistencia en entrenamiento físico, rehabilitación y educación deportiva.

1.3 Alcance

El proyecto comprende el diseño, implementación y validación de un modelo de clasificación de técnica en el ejercicio de *press* de hombro. El alcance abarca la construcción del *pipeline* completo de procesamiento, desde video crudo hasta clasificación etiquetada, incluyendo la extracción de *keypoints*, la corrección de *tracking* y el entrenamiento iterativo del modelo clasificador.

El conjunto de datos utilizado es el subconjunto de *press* de hombro del *dataset Fitness-AQA* (Parmar, Gharat, y Rhodin, 2022), con 2,135 videos etiquetados en tres categorías: técnica correcta, errores de codos y errores de rodillas. El resultado concreto del proyecto comprende el modelo entrenado, el *pipeline* documentado que recorre el proceso de extremo a extremo desde un video hasta una clasificación de la técnica, y un caso de estudio documentado del proceso completo que constituye una referencia técnica para desarrollos posteriores en el área de análisis automatizado de movimiento humano.

Capítulo 2

Marco Teórico

El presente capítulo establece los fundamentos conceptuales necesarios para comprender el modelo de clasificación desarrollado en este proyecto. Se construye progresivamente desde los principios generales de inteligencia artificial y aprendizaje automático hasta las técnicas específicas de visión por computadora y procesamiento temporal que sustentan las decisiones que definirán la arquitectura de la solución.

2.1 Fundamentos de Inteligencia Artificial y Aprendizaje Automático

La inteligencia artificial constituye un campo de la informática dedicado al desarrollo de sistemas computacionales capaces de realizar tareas que tradicionalmente requieren inteligencia humana. Entre estas tareas se incluyen el reconocimiento de patrones visuales, la comprensión del lenguaje natural, la toma de decisiones en entornos complejos, y el aprendizaje a partir de la experiencia. A diferencia de la programación tradicional, donde se especifican reglas explícitas para cada situación posible, los sistemas de inteligencia artificial buscan desarrollar capacidades emergentes que permitan generalizar conocimiento a situaciones no anticipadas durante el diseño del sistema (Goodfellow, Bengio, y Courville, 2016).

El aprendizaje automático (*machine learning*) representa un paradigma dentro de la inteligencia artificial donde los sistemas adquieren conocimiento mediante la exposición a datos en lugar de seguir instrucciones programadas explícitamente. En términos formales, se considera que un programa aprende de la experiencia E con respecto a alguna clase de tareas T y medida de desempeño P, si su desempeño en tareas de T, medido por P, mejora con la experiencia E (Mitchell, 1997). Este enfoque resulta particularmente valioso para problemas donde las reglas subyacentes son demasiado complejas o numerosas para especificarse manualmente, como ocurre en el análisis de imágenes o la interpretación de movimiento humano.

Dentro del aprendizaje automático se distinguen tres paradigmas principales según la naturaleza de la supervisión disponible durante el entrenamiento. El aprendizaje supervisado utiliza conjuntos de datos donde cada ejemplo posee una etiqueta correcta asociada, permitiendo al modelo aprender la relación entre entradas y salidas deseadas. El aprendizaje no supervisado opera sobre datos sin etiquetas, buscando descubrir estructura inherente como agrupamientos naturales o representaciones compactas. El aprendizaje por refuerzo, por su parte, aprende mediante interacción con un entorno que proporciona señales de recompensa o penalización según las acciones tomadas (LeCun, Bengio, y Hinton, 2015). El presente proyecto utiliza aprendizaje supervisado, donde videos de ejercicios etiquetados como correctos o con errores específicos permitirán entrenar modelos que clasifiquen nuevas ejecuciones.

El aprendizaje profundo (*deep learning*) constituye una rama del aprendizaje automático basada en redes neuronales artificiales con múltiples capas de procesamiento. La denominación "profundo" refiere a la arquitectura jerárquica donde cada capa transforma progresivamente las representaciones de datos, extrayendo características de abstracción creciente. Las capas iniciales típicamente detectan patrones simples como bordes o texturas, mientras las capas profundas combinan estas características básicas para reconocer conceptos complejos como objetos o ros-

tros (LeCun y cols., 2015).

Para ilustrar este principio, considérese el ejemplo de reconocimiento de dígitos escritos a mano: la primera capa de una red profunda podría aprender a detectar segmentos de línea en distintas orientaciones; la segunda capa combinaría estos segmentos para reconocer curvas y esquinas; capas posteriores integrarían estas formas para identificar cada dígito específico. Esta capacidad de construir representaciones jerárquicas de forma automática, sin necesidad de diseñar manualmente las características relevantes, constituye la ventaja fundamental del aprendizaje profundo frente a enfoques tradicionales.

Dada la naturaleza del problema abordado en este proyecto, donde se requiere analizar información visual compleja y extraer patrones de movimiento a partir de video, las técnicas de aprendizaje profundo resultan particularmente apropiadas. Los fundamentos presentados en esta sección establecen la base conceptual necesaria para comprender las arquitecturas específicas que se discutirán en secciones posteriores.

Redes neuronales artificiales

Las redes neuronales artificiales constituyen la arquitectura fundamental del aprendizaje profundo, inspiradas conceptualmente en la estructura del sistema nervioso biológico. La unidad básica de procesamiento es la neurona artificial, que recibe múltiples señales de entrada, las combina mediante una suma ponderada, agrega un término de sesgo, y aplica una función de activación para producir su salida. Formalmente, la salida de una neurona puede expresarse como:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

donde x_i representa cada entrada, w_i el peso asociado que determina la importancia de esa entrada, b el sesgo que permite ajustar el umbral de activación, y f la función de activación que introduce no linealidad en el modelo (Goodfellow y cols., 2016).

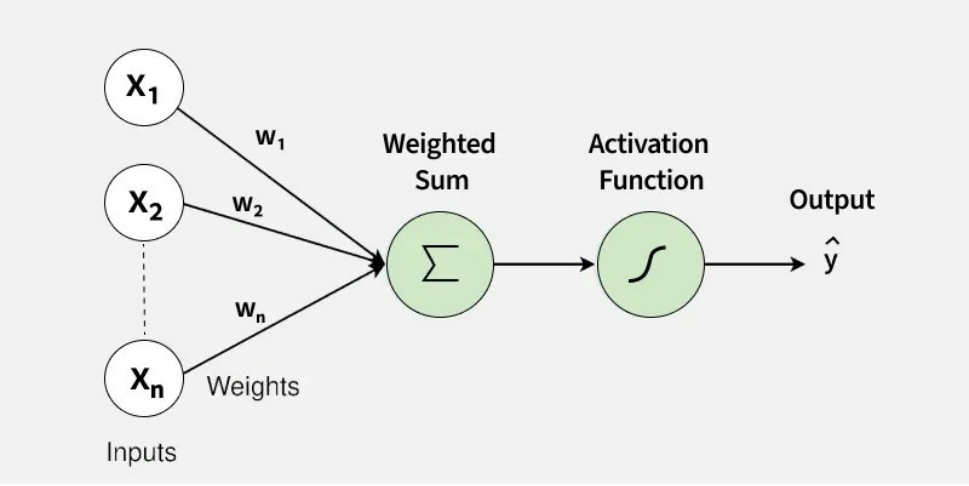


Figura 1. Diagrama de una neurona artificial mostrando entradas ponderadas, unidad de suma y función de activación. Fuente: Adaptado de GeeksforGeeks.

Las funciones de activación desempeñan un rol crítico al introducir no linealidad, permitiendo que la red aprenda relaciones complejas que una combinación lineal de entradas no podría capturar. Entre las funciones de activación más utilizadas se encuentra ReLU (*Rectified Linear Unit*), definida como:

$$f(x) = \text{máx}(0, x) \tag{2.2}$$

Esta función ha demostrado facilitar el entrenamiento de redes profundas al mitigar problemas de gradiente desvaneciente. Otras funciones históricamente relevantes incluyen la sigmoide $\sigma(x) = 1/(1+e^{-x})$ y la tangente hiperbólica $\tanh(x)$, aunque su uso ha disminuido en capas intermedias por presentar gradientes pequeños en regiones de saturación (Goodfellow y cols., 2016).

Las neuronas se organizan en capas sucesivas formando arquitecturas de red. Una capa densa o completamente conectada (*fully connected*) conecta cada neurona de una capa con todas las neuronas de la capa siguiente. La primera capa recibe los datos de entrada, las capas intermedias (denominadas capas ocultas) realizan transformaciones sucesivas, y la capa final produce la salida del modelo. Para tareas de clasificación, la capa de salida típicamente utiliza la función *softmax* que transforma los valores en una distribución de probabilidad sobre las clases posibles.

Proceso de entrenamiento

El entrenamiento de redes neuronales consiste en ajustar los pesos y sesgos para minimizar una función de pérdida que cuantifica la discrepancia entre las predicciones del modelo y las etiquetas correctas. Este proceso opera mediante dos fases alternadas: propagación hacia adelante (*forward propagation*) y retropropagación (*backpropagation*).

Durante la propagación hacia adelante, los datos de entrada atraviesan la red capa por capa, aplicando en cada una las transformaciones definidas por los pesos actuales, hasta generar una predicción en la capa de salida. Esta predicción se compara con la etiqueta correcta mediante la función de pérdida, produciendo un valor escalar que indica qué tan alejada está la predicción del resultado deseado.

La retropropagación, formalizada por Rumelhart, Hinton, y Williams (1986), constituye el algoritmo que permite calcular eficientemente el gradiente de la función de pérdida respecto a cada peso de la red. Aplicando la regla de la cadena del cálculo diferencial, se propaga el error desde la capa de salida hacia las capas anteriores, determinando cómo cada peso contribuyó al error final. Estos gradientes indican la dirección y magnitud del ajuste necesario en cada peso para reducir la pérdida.

La función de pérdida varía según el tipo de tarea. Para clasificación multiclase, donde el modelo debe distinguir entre más de dos categorías posibles, se utiliza comúnmente la entropía cruzada categórica (*cross-entropy*), definida como:

$$L = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{2.3}$$

donde C es el número de clases, y_c indica si la clase c es la correcta (1) o no (0), y $\hat{y}_c$ representa la probabilidad predicha para la clase c. Esta función penaliza fuertemente las predicciones confiadas pero incorrectas, incentivando al modelo a asignar alta probabilidad a la clase correcta (Goodfellow y cols., 2016).

Métricas de evaluación

Es importante distinguir entre la función de pérdida utilizada durante el entrenamiento y las métricas empleadas para evaluar el desempeño del modelo. Mientras la función de pérdida (como *cross-entropy*) guía la optimización de los pesos mediante gradientes, las métricas de evaluación cuantifican el rendimiento del modelo en términos interpretables para el dominio de

aplicación.

La precisión (*accuracy*) constituye la métrica más intuitiva para tareas de clasificación, definida como la proporción de predicciones correctas sobre el total de ejemplos:

$$\text{Accuracy} = \frac{\text{Predicciones correctas}}{\text{Total de ejemplos}} \tag{2.4}$$

Sin embargo, la precisión puede resultar engañosa cuando las clases están desbalanceadas, ya que un modelo que prediga siempre la clase mayoritaria obtendría alta precisión sin haber aprendido patrones útiles.

El *F1-score* proporciona una medida más robusta al combinar precisión (*precision*) y exhaustividad (*recall*) mediante su media armónica. La precisión mide qué proporción de las predicciones positivas fueron correctas, mientras la exhaustividad mide qué proporción de los casos positivos reales fueron identificados. El *F1-score* se define como:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.5}$$

Para problemas multiclase, el *F1-score* macro calcula el F1 para cada clase independientemente y promedia los resultados, otorgando igual importancia a todas las clases independientemente de su frecuencia en el *dataset*. Esta métrica resulta particularmente apropiada cuando todas las categorías tienen relevancia equivalente para la aplicación. Por ejemplo, en un sistema de diagnóstico médico que clasifica entre "sano", "enfermedad A" y "enfermedad B", identificar correctamente cada categoría es igualmente crítico, independientemente de cuál sea más frecuente en la población.

Optimizadores y actualización de pesos

Una vez calculados los gradientes, los pesos se actualizan utilizando algoritmos de optimización. El descenso de gradiente estocástico (SGD por sus siglas en inglés, *Stochastic Gradient Descent*) constituye el método fundamental, actualizando los pesos en la dirección opuesta al gradiente:

$$w_{t+1} = w_t - \eta \nabla L(w_t) \tag{2.6}$$

donde η representa la tasa de aprendizaje (*learning rate*), un hiperparámetro crítico que controla la magnitud de los ajustes. Una tasa de aprendizaje demasiado alta puede causar oscilaciones que impidan la convergencia, mientras una demasiado baja resulta en entrenamiento excesivamente lento.

El término "estocástico" refiere a que, en lugar de calcular el gradiente sobre todo el conjunto de datos (computacionalmente prohibitivo para *datasets* grandes), se utilizan subconjuntos aleatorios denominados mini-lotes (*mini-batches*). El tamaño del lote (*batch size*) representa otro hiperparámetro importante: lotes más pequeños introducen más ruido en las estimaciones del gradiente pero permiten actualizaciones más frecuentes.

Para acelerar la convergencia y suavizar las actualizaciones, se introdujo el concepto de momento (*momentum*), que acumula un promedio móvil de gradientes pasados:

$$v_{t+1} = \beta v_t + \nabla L(w_t) \tag{2.7}$$

$$w_{t+1} = w_t - \eta v_{t+1} \tag{2.8}$$

donde β (típicamente 0.9) controla cuánto peso se asigna a gradientes históricos. El momento ayuda a atravesar regiones planas de la superficie de pérdida y amortigua oscilaciones en regiones con gradientes inconsistentes (Goodfellow y cols., 2016).

Optimizadores más sofisticados como *Adam* (Kingma y Ba, 2015) combinan momento con tasas de aprendizaje adaptativas por parámetro, ajustando automáticamente la magnitud de actualización según la historia de gradientes de cada peso. *AdamW* (Loshchilov y Hutter, 2019) representa una variante que implementa correctamente la regularización por decaimiento de pesos (*weight decay*) de forma desacoplada, mejorando la generalización en muchos contextos.

Hiperparámetros y ciclo de entrenamiento

El entrenamiento se organiza en *epochs*, donde cada *epoch* representa una pasada completa por todo el conjunto de datos de entrenamiento. Los hiperparámetros principales que controlan el entrenamiento incluyen:

- **Tasa de aprendizaje:** Magnitud de los ajustes de pesos, frecuentemente iniciada en valores como 0.01 o 0.001 y reducida progresivamente.
- **Tamaño del lote:** Número de ejemplos procesados antes de cada actualización de pesos.
- **Número de *epochs*:** Cuántas veces el modelo procesa el conjunto de entrenamiento completo.
- **Decaimiento de pesos:** Término de penalización que incentiva pesos pequeños, previniendo *overfitting*.

La selección de hiperparámetros impacta significativamente el desempeño final y típicamente requiere experimentación sistemática, aunque técnicas como *learning rate schedulers* que reducen la tasa de aprendizaje según progresa el entrenamiento, o *warm-up* que la incrementa gradualmente al inicio, ayudan a estabilizar el proceso.

Generalización y *overfitting*

El objetivo fundamental del aprendizaje automático no es memorizar los datos de entrenamiento sino generalizar a ejemplos nuevos no vistos durante el entrenamiento. El *overfitting* (sobreajuste) ocurre cuando un modelo se ajusta excesivamente a las particularidades del conjunto de entrenamiento, perdiendo capacidad de generalización. Este fenómeno se manifiesta cuando el desempeño en datos de entrenamiento es significativamente superior al desempeño en datos de validación o prueba.

El *overfitting* resulta especialmente problemático cuando el *dataset* es pequeño relativo a la complejidad del modelo, situación común en dominios especializados como la evaluación de técnica en ejercicios donde recopilar datos etiquetados requiere esfuerzo significativo. Las arquitecturas de aprendizaje profundo, con millones de parámetros ajustables, poseen capacidad suficiente para memorizar conjuntos de datos pequeños sin aprender patrones generalizables.

Para mitigar el *overfitting* se emplean diversas técnicas de regularización. El abandono (*dropout*) consiste en desactivar aleatoriamente un porcentaje de neuronas durante cada paso de entrenamiento, forzando a la red a desarrollar representaciones redundantes y robustas. El decaimiento de pesos (*weight decay*) agrega a la función de pérdida un término proporcional a la magnitud de los pesos, penalizando modelos excesivamente complejos. La parada temprana (*early stopping*) detiene el entrenamiento cuando el desempeño en validación comienza a deteriorarse aunque el desempeño en entrenamiento continúe mejorando.

La aumentación de datos (*data augmentation*) constituye otra estrategia fundamental, especialmente relevante para dominios con *datasets* limitados. Consiste en aplicar transformaciones a los datos de entrenamiento que preservan su significado semántico pero introducen variabilidad. Para imágenes, transformaciones típicas incluyen rotaciones, traslaciones, cam-

bios de escala, y variaciones de iluminación. En procesamiento de audio, se emplean técnicas como cambios de velocidad, adición de ruido de fondo, o desplazamiento temporal. Para datos de poses humanas, técnicas como *Pose Jittering* agregan ruido controlado a las coordenadas de los puntos anatómicos, simulando la variabilidad natural del movimiento y la imprecisión inherente en sistemas de detección (Duan, Zhao, Chen, Lin, y Dai, 2022).

El suavizado de etiquetas (*label smoothing*) representa una técnica de regularización que modifica las etiquetas de entrenamiento para ser menos categóricas. En lugar de asignar probabilidad 1 a la clase correcta y 0 a las demás, se distribuye una pequeña fracción de probabilidad entre todas las clases. Esto previene que el modelo se vuelva excesivamente confiado en sus predicciones, mejorando la calibración y generalización (Szegedy, Vanhoucke, Ioffe, Shlens, y Wojna, 2016).

2.2 Visión por Computadora y Redes Convolucionales

La visión por computadora constituye un campo interdisciplinario que busca dotar a sistemas computacionales de la capacidad de interpretar y comprender información visual del mundo real. Esta disciplina abarca desde el procesamiento básico de imágenes hasta tareas complejas como reconocimiento de objetos, segmentación semántica, y en el contexto de este proyecto, la detección de posturas corporales humanas. El desarrollo del aprendizaje profundo, particularmente las redes neuronales convolucionales, ha revolucionado este campo permitiendo alcanzar desempeños que igualan o superan la capacidad humana en tareas específicas de reconocimiento visual (LeCun y cols., 2015).

Representación digital de imágenes

Las imágenes digitales se representan como matrices bidimensionales de valores numéricos, donde cada elemento corresponde a un píxel. En imágenes en escala de grises, cada píxel almacena un único valor de intensidad, típicamente en el rango [0, 255] para representaciones de 8 bits, donde 0 corresponde a negro y 255 a blanco. Las imágenes a color utilizan múltiples canales para representar la información cromática; el formato RGB (*Red, Green, Blue*) emplea tres canales independientes que codifican la intensidad de los componentes rojo, verde y azul respectivamente. Una imagen RGB de dimensiones H×W se representa entonces como un tensor tridimensional de forma H×W×3, donde cada posición espacial contiene un triplete de valores que determina el color del píxel correspondiente.

Esta representación matricial permite aplicar operaciones matemáticas sobre las imágenes de forma sistemática. Sin embargo, las imágenes naturales presentan características que dificultan su análisis mediante técnicas tradicionales: alta dimensionalidad (una imagen de 224×224 píxeles RGB contiene aproximadamente 150,000 valores), variabilidad en iluminación, escala, rotación y punto de vista, además de la presencia de ruido y oclusiones parciales. Las redes neuronales convolucionales abordan estos desafíos mediante arquitecturas especializadas que explotan la estructura espacial inherente de las imágenes.

Operación de convolución

La operación de convolución constituye el fundamento de las redes neuronales convolucionales, aplicando filtros (*kernels*) que se deslizan sobre la imagen para detectar patrones locales. Un *kernel* es una matriz pequeña de pesos, típicamente de dimensiones 3×3 o 5×5, que define qué patrón específico detecta esa operación. Durante la convolución, el *kernel* se posiciona en cada ubicación de la imagen, se calcula el producto elemento a elemento entre el *kernel* y la región de la imagen que cubre, y se suman los resultados para producir un único valor de salida. Al desplazar el *kernel* por toda la imagen, se genera un mapa de características (*feature*

map) que indica dónde y con qué intensidad aparece el patrón que el *kernel* detecta.

Para ilustrar concretamente esta operación, considérese un *kernel* 3×3 diseñado para detectar bordes verticales:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Este *kernel* asigna valores negativos a la columna izquierda, cero al centro, y valores positivos a la columna derecha. Cuando se aplica sobre una región de la imagen donde los píxeles de la izquierda son oscuros (valores bajos) y los de la derecha son claros (valores altos), el resultado de la suma ponderada será un valor alto positivo, indicando la presencia de un borde vertical en esa ubicación. Si la región es uniforme (sin bordes), los valores negativos y positivos se cancelan, produciendo un resultado cercano a cero. De esta forma, el mapa de características resultante "ilumina" las zonas de la imagen donde existen bordes verticales.

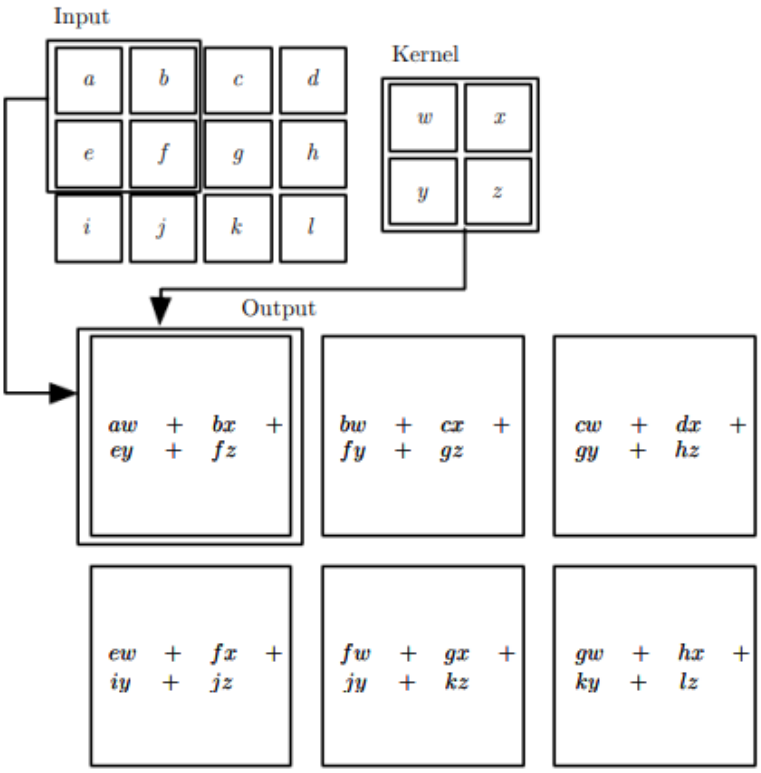


Figura 2. Operación de convolución 2D: un *kernel* se aplica sobre una matriz de entrada para generar un mapa de características. Fuente: Adaptado de Goodfellow et al. (2016).

Dependiendo de los valores que contenga el *kernel*, la convolución detectará diferentes tipos de patrones en la imagen. La Figura 3 ilustra cómo distintos filtros clásicos de procesamiento de imágenes producen resultados radicalmente diferentes al aplicarse sobre la misma imagen: filtros para detección de bordes horizontales y verticales, el operador Sobel que combina información de gradiente en ambas direcciones, el operador Laplaciano que detecta cambios bruscos de intensidad, y el detector de bordes de Canny que produce contornos más definidos. En redes neuronales convolucionales, los valores de estos *kernels* no se diseñan manualmente sino que se aprenden automáticamente durante el entrenamiento, permitiendo que la red descubra los patrones más relevantes para la tarea específica.

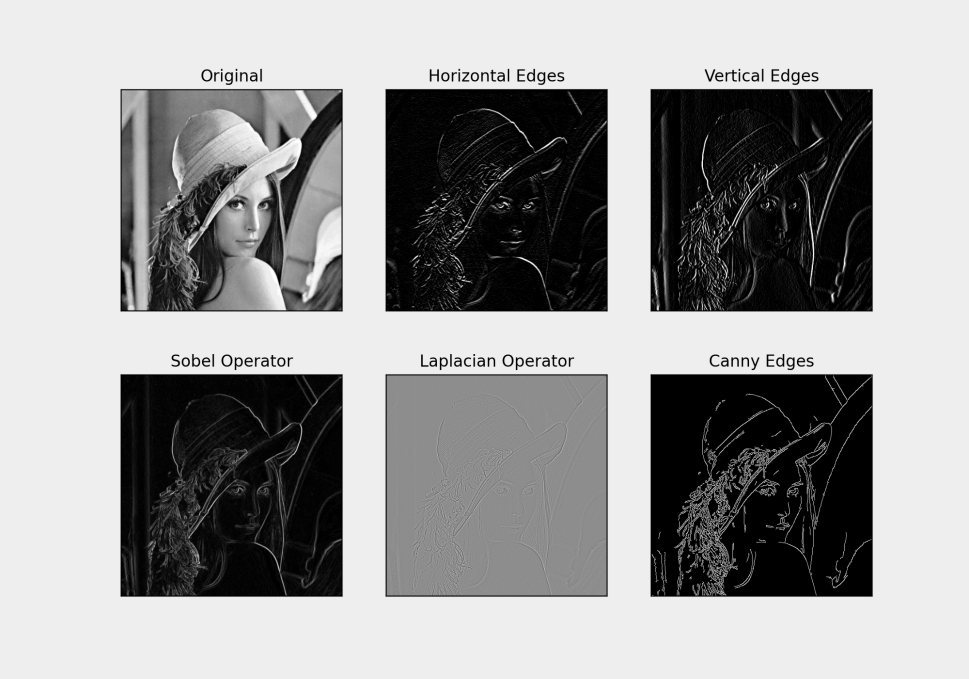


Figura 3. Aplicación de distintos filtros de convolución sobre una misma imagen: detección de bordes horizontales, verticales, operadores Sobel, Laplaciano y Canny. Fuente: Adaptado de Codingwalks.

Formalmente, la convolución discreta bidimensional entre una imagen I y un *kernel* K de tamaño k×k se define como:

$$(I * K)[i,j] = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I[i+m, j+n] \cdot K[m,n] \tag{2.10}$$

donde [i,j] representa la posición en el mapa de características de salida.

En redes neuronales convolucionales, los valores del *kernel* no se diseñan manualmente como en el ejemplo anterior, sino que se aprenden automáticamente durante el entrenamiento. La red descubre por sí misma qué patrones son útiles para la tarea: detectores de bordes en distintas orientaciones, detectores de texturas, gradientes de color, y progresivamente patrones más complejos en capas profundas.

Esta operación posee dos propiedades fundamentales que la hacen particularmente adecuada para procesamiento de imágenes. La primera es la conectividad local: cada neurona de salida depende únicamente de una región pequeña de la entrada (el campo receptivo), asumiendo que la información relevante para detectar patrones visuales es espacialmente localizada. La segunda es el compartimiento de parámetros: el mismo *kernel* se aplica en todas las posiciones de la imagen, reduciendo drásticamente el número de parámetros respecto a capas densas y permitiendo detectar el mismo patrón independientemente de su ubicación espacial (invariancia a la traslación).

Arquitectura de redes convolucionales

Una red neuronal convolucional típica combina múltiples tipos de capas organizadas jerárquicamente. Las capas convolucionales aplican conjuntos de *kernels* a la entrada, generando múltiples mapas de características que capturan diferentes aspectos visuales. En las capas iniciales, los *kernels* aprenden a detectar características de bajo nivel como bordes horizontales, verticales, diagonales, y gradientes de color. Las capas intermedias combinan estas características básicas para reconocer texturas, esquinas, y patrones más complejos. Las capas profundas

integran información de capas anteriores para detectar conceptos semánticos de alto nivel como partes de objetos o, en el contexto de análisis de pose, configuraciones articulares específicas.

La zancada (*stride*) controla el desplazamiento del *kernel* entre posiciones consecutivas. Un *stride* de 1 desplaza el *kernel* un píxel por vez, mientras un *stride* de 2 lo desplaza dos píxeles, reduciendo las dimensiones espaciales de salida a la mitad. El relleno (*padding*) agrega píxeles adicionales (típicamente ceros) alrededor de los bordes de la entrada, permitiendo controlar las dimensiones de salida y asegurando que los píxeles de borde reciban el mismo tratamiento que los interiores.

Las capas de *pooling* (agrupamiento) reducen la resolución espacial de los mapas de características, consolidando información de regiones locales. El *max pooling* selecciona el valor máximo dentro de cada ventana (típicamente 2×2), preservando las activaciones más fuertes mientras reduce las dimensiones a la mitad. Esta operación proporciona cierta invariancia a pequeñas traslaciones: si un patrón se desplaza ligeramente dentro de la ventana de *pooling*, la salida permanece similar. Adicionalmente, la reducción de dimensionalidad disminuye el costo computacional de capas subsiguientes y aumenta el campo receptivo efectivo de las neuronas en capas profundas.

Las capas de activación introducen no linealidad después de cada operación convolucional. Como se mencionó anteriormente, ReLU (*Rectified Linear Unit*) se ha convertido en la función de activación estándar en redes convolucionales modernas, permitiendo gradientes que no se desvanecen para valores positivos y computación eficiente.

El problema del gradiente desvaneciente y redes profundas

A medida que las redes convolucionales se hicieron más profundas para capturar representaciones más abstractas, surgió un problema fundamental: el gradiente desvaneciente. Durante la retropropagación, los gradientes se multiplican sucesivamente al atravesar cada capa hacia atrás. Si estos factores multiplicativos son consistentemente menores que 1 (como ocurre con funciones de activación saturadas), el gradiente decrece exponencialmente, resultando en actualizaciones de peso insignificantes en las capas iniciales. Paradójicamente, las capas más alejadas de la salida, que procesan características fundamentales de bajo nivel, reciben los gradientes más débiles y aprenden más lentamente.

Este problema limitó durante años la profundidad práctica de las redes neuronales. Aunque teóricamente redes más profundas deberían poder representar funciones más complejas, en la práctica el entrenamiento fallaba cuando se superaban ciertas profundidades. He, Zhang, Ren, y Sun (2016) demostraron experimentalmente que redes de 56 capas tenían mayor error de entrenamiento que redes de 20 capas, un resultado contraintuitivo que evidenciaba dificultades de optimización más que limitaciones de capacidad representacional.

ResNet y conexiones residuales

La arquitectura *ResNet (Residual Network)*, propuesta por He y cols. (2016), revolucionó el diseño de redes profundas mediante la introducción de conexiones residuales (*skip connections*). En lugar de aprender directamente una función objetivo H(x), cada bloque residual aprende una función residual F(x) = H(x) - x, de modo que la salida se expresa como:

$$y = F(x) + x \tag{2.11}$$

donde x representa la entrada al bloque y F(x) las transformaciones convolucionales dentro del bloque. La conexión de atajo (*skip connection*) que suma directamente la entrada a la salida permite que los gradientes fluyan sin obstáculos hacia capas anteriores durante la retropropagación. Si las transformaciones F(x) se vuelven insignificantes, el bloque simplemente

implementa una función identidad, no degradando el desempeño.

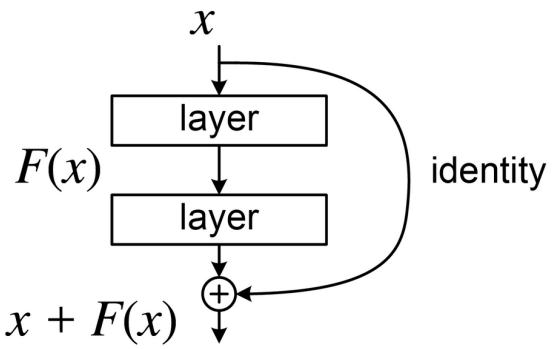


Figura 4. Bloque residual de *ResNet*. La entrada x atraviesa las capas convolucionales produciendo F(x), mientras la conexión de identidad (*skip connection*) preserva x para sumarse a la salida.

Esta innovación aparentemente simple tuvo impacto profundo: permitió entrenar redes de cientos e incluso miles de capas. *ResNet-50*, *ResNet-101* y *ResNet-152* se convirtieron en arquitecturas estándar para extracción de características visuales, demostrando que la profundidad efectivamente mejora el desempeño cuando el problema de optimización se resuelve adecuadamente. El concepto de conexiones residuales se ha extendido a prácticamente todas las arquitecturas modernas de aprendizaje profundo, incluyendo los *Transformers* que se discutirán en la siguiente sección.

Detección de objetos

Las redes convolucionales no solo permiten clasificar imágenes completas sino también localizar y clasificar múltiples objetos dentro de una imagen. La detección de objetos combina dos tareas: determinar las coordenadas espaciales donde se encuentran los objetos (localización) y asignar una categoría a cada objeto detectado (clasificación). La localización se expresa mediante *bounding boxes* (cajas delimitadoras), rectángulos definidos por coordenadas (x, y, ancho, alto) que encierran cada objeto detectado.

Entre las arquitecturas de detección de objetos, *YOLO (You Only Look Once)* (Redmon, Divvala, Girshick, y Farhadi, 2016) introdujo un enfoque de detección en una sola pasada que procesa la imagen completa simultáneamente, logrando velocidades de inferencia en tiempo real. A diferencia de métodos anteriores que aplicaban clasificadores en múltiples regiones propuestas, *YOLO* divide la imagen en una grilla y predice directamente *bounding boxes* y probabilidades de clase para cada celda. Las versiones posteriores, particularmente *YOLOv8*, han refinado esta arquitectura alcanzando un balance entre precisión y velocidad que las hace ideales para aplicaciones en tiempo real como vigilancia, conducción autónoma, o detección de personas como paso previo a otras tareas de análisis visual.

2.3 Arquitectura Transformer y Vision Transformers

Las redes convolucionales, a pesar de su éxito en visión por computadora, presentan una limitación estructural: su campo receptivo es inherentemente local. Cada neurona en una capa convolucional observa únicamente una región limitada de la entrada, determinada por el tamaño del *kernel*. Aunque el campo receptivo crece en capas profundas al componer múltiples convoluciones, capturar dependencias entre regiones distantes de la imagen requiere muchas

capas intermedias. Esta limitación motivó la exploración de arquitecturas alternativas capaces de modelar relaciones globales de forma más directa.

Mecanismo de atención

El mecanismo de atención, originalmente desarrollado para procesamiento de lenguaje natural, permite que cada elemento de una secuencia "atienda" selectivamente a todos los demás elementos, ponderando su relevancia de forma dinámica según el contenido. Conceptualmente, la atención responde la pregunta: "dado el elemento actual, ¿qué otros elementos de la secuencia son relevantes para procesarlo?".

El mecanismo de *Self-Attention* (autoatención) opera sobre tres representaciones derivadas de cada elemento de entrada: consultas (*queries*), claves (*keys*) y valores (*values*), obtenidas mediante transformaciones lineales aprendibles.

Para comprender intuitivamente este mecanismo, resulta útil una analogía con búsqueda en una base de datos. Imaginemos una biblioteca donde cada libro tiene una ficha descriptiva (la clave) y un contenido (el valor). Cuando un lector llega con una pregunta específica (la consulta), el bibliotecario compara la pregunta con cada ficha, determina qué tan relevante es cada libro para esa pregunta, y proporciona una respuesta que combina información de los libros más relevantes. En *Self-Attention*, cada elemento de la secuencia actúa simultáneamente como libro y como lector: genera su propia consulta para buscar información relevante, mientras ofrece su clave y valor para que otros elementos lo consulten.

El proceso computacional procede de la siguiente manera: para cada posición, se calcula la similitud entre su consulta y las claves de todas las posiciones mediante producto punto. Estas similitudes se normalizan mediante la función *softmax*, que las convierte en pesos de atención que suman 1. Finalmente, se computa una suma ponderada de los valores según estos pesos, produciendo una representación que integra información de toda la secuencia. Formalmente:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.12}$$

donde Q, K, V representan las matrices de consultas, claves y valores respectivamente. El factor $\sqrt{d_k}$ (donde d_k es la dimensión de las claves) escala los productos punto para evitar que valores muy grandes saturen la función *softmax*, lo que produciría gradientes cercanos a cero durante el entrenamiento.

La potencia del *Self-Attention* radica en que los pesos de atención se calculan dinámicamente según el contenido, no según posiciones fijas. Esto permite que el modelo aprenda qué relaciones son relevantes para cada tarea: en traducción, una palabra puede atender a su sujeto aunque estén separados por muchas palabras; en análisis de imágenes, una región puede atender a regiones distantes que contienen contexto relevante.

Arquitectura Transformer

La arquitectura *Transformer*, introducida por Vaswani y cols. (2017) en el artículo "Attention is All You Need", demostró que mecanismos de atención por sí solos, sin recurrencia ni convoluciones, podían alcanzar resultados superiores en tareas de procesamiento de lenguaje. Esta arquitectura marcó un punto de inflexión en el campo, ya que las aproximaciones previas dependían de redes recurrentes (RNN, LSTM) que procesaban secuencias elemento por elemento, limitando la paralelización y dificultando capturar dependencias de largo alcance.

El *Transformer* organiza sus componentes en bloques que se apilan repetidamente. Cada bloque contiene dos subcomponentes principales conectados mediante conexiones residuales y

normalización de capa.

El primer subcomponente es la atención multi-cabeza (*Multi-Head Attention*). En lugar de ejecutar una única operación de atención, se ejecutan múltiples operaciones en paralelo (típicamente 8 o 12 "cabezas"), cada una con sus propias matrices de proyección para Q, K y V. Intuitivamente, cada cabeza puede especializarse en capturar diferentes tipos de relaciones: una cabeza podría aprender a conectar sujetos con verbos, otra a relacionar pronombres con sus antecedentes, otra a capturar dependencias sintácticas. Los resultados de todas las cabezas se concatenan y proyectan linealmente para producir la salida final. Esta multiplicidad permite al modelo atender simultáneamente a información de diferentes subespacios de representación.

El segundo subcomponente es una red *feed-forward* de dos capas aplicada independientemente a cada posición:

$$\text{FFN}(x) = \text{máx}(0, xW_1 + b_1)W_2 + b_2 \tag{2.13}$$

Esta red expande la dimensionalidad de las representaciones (típicamente por un factor de 4), aplica una no linealidad ReLU, y la reduce nuevamente. Aunque aparentemente simple, este componente es donde ocurre gran parte del "procesamiento" de la información, transformando las representaciones enriquecidas por la atención.

Las conexiones residuales, heredadas de *ResNet*, suman la entrada de cada subcomponente a su salida, facilitando el flujo de gradientes y permitiendo apilar muchas capas. La normalización de capa (*Layer Normalization*) estabiliza las activaciones, normalizando las representaciones de cada elemento independientemente.

Una ventaja fundamental del *Transformer* sobre arquitecturas recurrentes es la paralelización: como todas las posiciones se procesan simultáneamente (no secuencialmente), el entrenamiento aprovecha plenamente el paralelismo de GPUs modernas, reduciendo drásticamente los tiempos de entrenamiento para secuencias largas.

Vision Transformer (ViT)

Dosovitskiy y cols. (2021) demostraron que la arquitectura *Transformer*, con mínimas modificaciones, podía aplicarse directamente a imágenes alcanzando resultados competitivos con redes convolucionales. La idea central es tratar una imagen como una secuencia de parches (*patches*), de manera análoga a cómo el *Transformer* original trata un texto como una secuencia de palabras (tokens).

Esta analogía funciona porque el mecanismo de *Self-Attention* fue diseñado para procesar secuencias de elementos, calculando relaciones entre todos los pares. Al *Transformer* no le importa si esos elementos son palabras, fragmentos de audio, o regiones de una imagen: solo necesita que cada elemento esté representado como un vector numérico. Dividir una imagen en parches convierte el problema de "procesar una imagen" en "procesar una secuencia de vectores", que es exactamente lo que el *Transformer* sabe hacer.

El procesamiento en *ViT* comienza dividiendo la imagen de entrada en parches no superpuestos de tamaño fijo, típicamente 16×16 píxeles. Por ejemplo, una imagen de 224×224 píxeles produce una grilla de 14×14, es decir, 196 parches. Cada parche contiene 16×16×3 = 768 valores (para los tres canales RGB), que se aplanan en un vector unidimensional y se proyectan mediante una capa lineal a la dimensión interna del modelo.

El mecanismo de atención, por sí solo, no tiene noción de posición: trata a todos los elementos como un conjunto desordenado. Para que el modelo comprenda que un parche está "arriba a la izquierda" mientras otro está "abajo a la derecha", se suma a cada *embedding* de

parche un vector posicional aprendible que codifica su ubicación en la grilla original. De esta forma, el modelo puede aprender que ciertos patrones espaciales son significativos.

Adicionalmente, se agrega al inicio de la secuencia un token especial denominado [CLS] (*class token*), un vector aprendible que no corresponde a ningún parche de la imagen. Este token participa en todas las operaciones de atención a lo largo de las capas del *Transformer*, "recolectando" progresivamente información de todos los parches. Al final del procesamiento, la representación del token [CLS] sirve como representación global de la imagen completa, utilizada para tareas de clasificación.

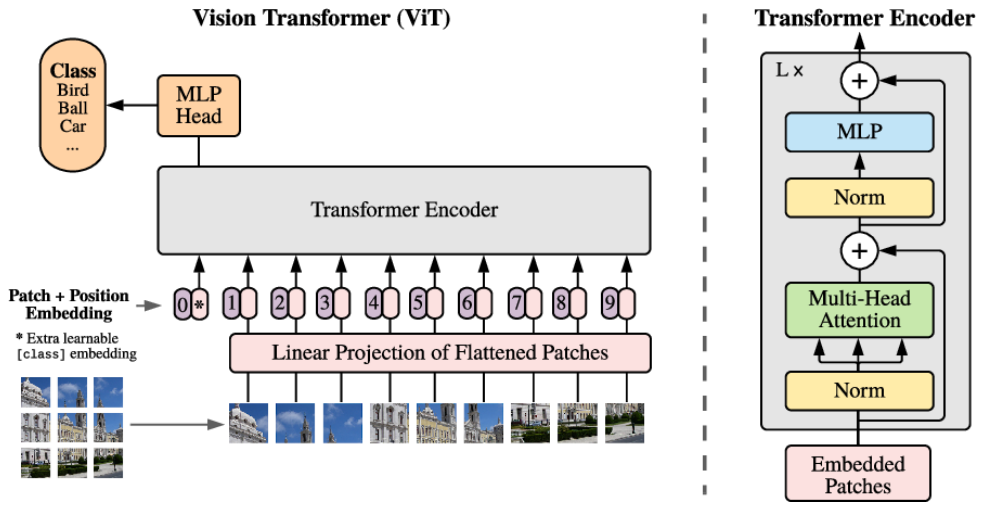


Figura 5. Arquitectura del *Vision Transformer (ViT)*: división de imagen en parches, proyección lineal, *embeddings* posicionales y procesamiento mediante *Transformer Encoder*.

La secuencia completa (token [CLS] más los 196 parches, totalizando 197 elementos) se procesa mediante múltiples capas de *Transformer encoder*, idénticas a las descritas en la sección anterior. En cada capa de atención, cada parche puede atender a todos los demás parches de la imagen, capturando relaciones globales desde la primera capa.

La diferencia fundamental con las redes convolucionales radica en el alcance de las interacciones. Una CNN con *kernel* 3×3 en su primera capa solo relaciona píxeles adyacentes, requiriendo muchas capas apiladas para que información de regiones distantes se comunique. En *ViT*, desde la primera capa de atención, un parche en una esquina puede relacionarse directamente con un parche en la esquina opuesta. Esta capacidad de modelar dependencias globales resulta valiosa para tareas donde la configuración espacial completa es relevante, como entender escenas complejas o analizar relaciones entre partes distantes de un objeto.

Los *Vision Transformers* requieren grandes cantidades de datos para entrenar desde cero, ya que carecen de los sesgos inductivos de las CNNs como localidad e invariancia a traslación. Sin embargo, mediante preentrenamiento a gran escala seguido de *fine-tuning* en tareas específicas, han alcanzado resultados competitivos o superiores en múltiples aplicaciones de visión por computadora.

2.4 Estimación de Poses Humanas

La estimación de poses humanas constituye una tarea fundamental en visión por computadora que consiste en detectar y localizar puntos anatómicos clave (*keypoints*) del cuerpo humano en imágenes o videos. Estos *keypoints* típicamente corresponden a articulaciones como

hombros, codos, muñecas, caderas, rodillas y tobillos, junto con puntos de referencia como nariz, ojos y orejas. La representación resultante, denominada esqueleto o pose, captura la configuración espacial del cuerpo humano, habilitando análisis de movimiento, reconocimiento de acciones, y en el contexto de este proyecto, evaluación de técnica en ejercicios.

Formato estándar COCO

El *dataset COCO (Common Objects in Context)* estableció un formato estándar de 17 *keypoints* para representar poses humanas que se ha convertido en referencia para la comunidad de investigación. Estos puntos incluyen: nariz (0), ojo izquierdo (1), ojo derecho (2), oreja izquierda (3), oreja derecha (4), hombro izquierdo (5), hombro derecho (6), codo izquierdo (7), codo derecho (8), muñeca izquierda (9), muñeca derecha (10), cadera izquierda (11), cadera derecha (12), rodilla izquierda (13), rodilla derecha (14), tobillo izquierdo (15) y tobillo derecho (16). Cada *keypoint* se representa mediante coordenadas (x, y) en píxeles junto con un valor de confianza que indica la certeza de la detección.

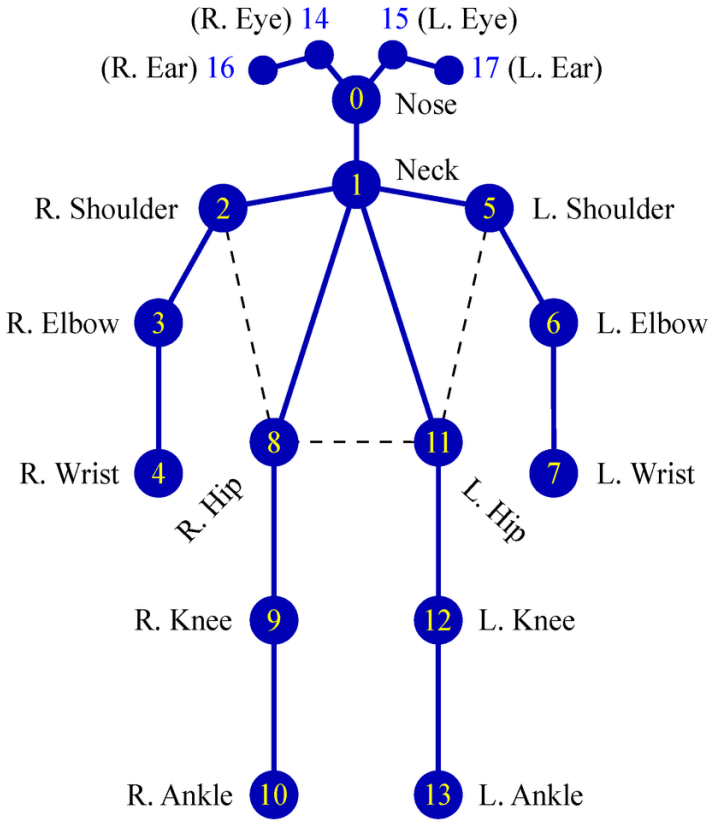


Figura 6. Los 17 *keypoints* del formato estándar *COCO* para poses humanas, con índices numéricos y nombres anatómicos. Fuente: Adaptado de documentación de *COCO*.

Enfoques top-down versus bottom-up

Los métodos de estimación de poses se clasifican en dos paradigmas principales según su estrategia para manejar múltiples personas en una imagen. Los enfoques *top-down* primero detectan todas las personas en la imagen mediante un detector de objetos, generando *bounding boxes* individuales, y luego aplican un modelo de estimación de pose a cada persona de forma independiente. Este enfoque desacopla la detección de personas de la estimación de *keypoints*, permitiendo utilizar arquitecturas especializadas para cada subtarea. Su desventaja es que el tiempo de procesamiento escala linealmente con el número de personas, y errores en la detección de personas se propagan a la estimación de poses.

Los enfoques *bottom-up* operan en dirección opuesta: primero detectan todos los *keypoints* de todas las personas simultáneamente, sin conocimiento previo de cuántas personas hay ni dónde están, y luego agrupan estos *keypoints* en personas individuales mediante algoritmos de asociación. Este paradigma procesa la imagen una única vez independientemente del número de personas, resultando en tiempo de inferencia constante. Sin embargo, la etapa de agrupamiento presenta desafíos cuando las personas están muy cercanas o parcialmente ocluidas.

OpenPose: Enfoque bottom-up con Part Affinity Fields

OpenPose, desarrollado por Cao, Simon, Wei, y Sheikh (2017), representa una arquitectura pionera de enfoque *bottom-up* que introdujo los *Part Affinity Fields (PAFs)* como mecanismo para asociar *keypoints* detectados a personas específicas. El sistema procesa la imagen mediante una red convolucional que produce dos tipos de salidas en paralelo: mapas de confianza (*confidence maps*) para localización de *keypoints* y campos de afinidad de partes (*Part Affinity Fields*) para asociación entre *keypoints*.

Los mapas de confianza son representaciones bidimensionales donde cada píxel indica la probabilidad de que un *keypoint* específico se encuentre en esa ubicación. Se genera un mapa de confianza separado para cada tipo de *keypoint* (hombro izquierdo, hombro derecho, codo izquierdo, etc.). Los picos locales en estos mapas corresponden a las ubicaciones de los *keypoints* detectados, y su valor indica la confianza de la detección. Cuando hay múltiples personas, cada mapa de confianza contiene múltiples picos, uno por cada instancia del *keypoint* correspondiente en la imagen.

Los *Part Affinity Fields* constituyen la innovación central de *OpenPose* para resolver el problema de asociación. Un PAF es un campo vectorial 2D que codifica la dirección y ubicación de las conexiones entre pares de *keypoints* relacionados anatómicamente (por ejemplo, hombro-codo o cadera-rodilla). Para cada par de *keypoints* conectados, el PAF define vectores unitarios a lo largo del "hueso" que los conecta. En regiones donde no hay conexión, el campo tiene magnitud cero. Cuando múltiples personas están presentes, el campo codifica todas las conexiones de todas las personas simultáneamente.

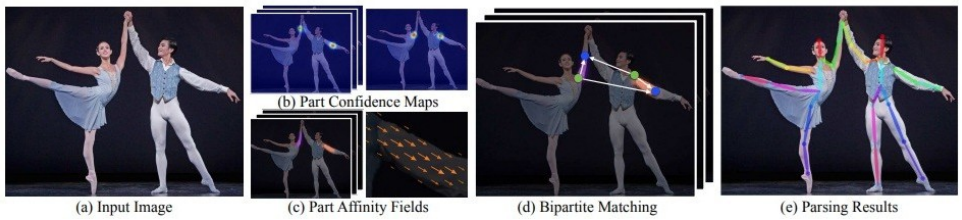


Figura 7. Pipeline de *OpenPose*: (a) imagen de entrada, (b) mapas de confianza de *keypoints*, (c) *Part Affinity Fields*, (d) emparejamiento bipartito, (e) esqueletos resultantes.

El proceso de asociación opera mediante un algoritmo de emparejamiento bipartito (*bipartite matching*). Para cada par de tipos de *keypoints* conectados (por ejemplo, todos los hombros izquierdos con todos los codos izquierdos), se evalúa la integral de línea del PAF correspondiente a lo largo del segmento que conecta cada par candidato. Las conexiones con integrales altas indican que el PAF "apoya" esa conexión, sugiriendo que ambos *keypoints* pertenecen a la misma persona. El algoritmo húngaro u otras técnicas de optimización combinatoria seleccionan el conjunto de conexiones que maximiza la suma total de afinidades respetando restricciones de exclusividad (cada *keypoint* puede pertenecer a una sola persona).

La arquitectura de *OpenPose* utiliza refinamiento iterativo: las predicciones iniciales de mapas de confianza y PAFs se refinan mediante etapas sucesivas de la red, donde cada etapa to-

ma como entrada tanto la imagen original como las predicciones de la etapa anterior. Este diseño permite corregir errores progresivamente, mejorando la precisión de detección y asociación.

Una ventaja significativa de *OpenPose* es que su tiempo de inferencia es independiente del número de personas en la imagen, procesando escenas con decenas de personas en tiempo similar a una persona sola. Sin embargo, su precisión puede degradarse en situaciones con oclusiones severas o poses muy cercanas donde los PAFs se superponen ambiguamente.

ViTPose: Vision Transformers para estimación de poses

ViTPose, propuesto por Xu, Zhang, Zhang, y Tao (2022), aplica la arquitectura *Vision Transformer* directamente a la tarea de estimación de poses humanas, demostrando que un diseño simple basado en *Transformers* puede alcanzar resultados competitivos sin componentes específicos de pose elaborados. La arquitectura sigue un enfoque *top-down*: asume que las personas ya han sido detectadas (típicamente mediante *YOLO* u otro detector) y estima los *keypoints* dentro de cada *bounding box* individualmente.

El procesamiento en *ViTPose* comienza con el recorte de la imagen según el *bounding box* de cada persona detectada, seguido de la división en parches que se procesan mediante un *backbone Vision Transformer* preentrenado. A diferencia de arquitecturas convolucionales tradicionales para estimación de pose que emplean diseños complejos con múltiples ramas o etapas de refinamiento, *ViTPose* utiliza el *ViT* como extractor de características seguido de un decodificador simple que genera mapas de calor (*heatmaps*) de salida para cada *keypoint*.

La simplicidad arquitectónica de *ViTPose* contrasta con su alto desempeño. Los autores argumentan que la capacidad del *Transformer* de capturar relaciones globales entre parches resulta particularmente beneficiosa para estimación de poses, donde la ubicación de una articulación está fuertemente condicionada por la configuración del resto del cuerpo. Por ejemplo, conocer la posición del hombro informa sobre la ubicación probable del codo, y esta información de largo alcance se captura naturalmente mediante los mecanismos de atención.

La salida de *ViTPose* consiste en 17 mapas de calor (uno por *keypoint COCO*), donde cada mapa indica la probabilidad espacial de ubicación del *keypoint* correspondiente. Las coordenadas finales se obtienen identificando el pico de mayor valor en cada mapa de calor. Adicionalmente, el sistema proporciona valores de confianza basados en la magnitud de estos picos, permitiendo filtrar detecciones poco fiables.

2.5 Procesamiento Temporal y Reconocimiento de Acciones

Las secciones anteriores abordaron el procesamiento de imágenes estáticas individuales. Sin embargo, el análisis de técnica en ejercicios requiere comprender cómo evoluciona la configuración corporal a lo largo del tiempo: un ejercicio correctamente ejecutado se distingue de uno incorrecto no solo por posturas instantáneas sino por la secuencia y dinámica del movimiento completo. El procesamiento temporal extiende las técnicas de visión por computadora al dominio de video, capturando información que se desarrolla a través de múltiples *frames*.

Representación de video y convoluciones 3D

Un video puede conceptualizarse como una secuencia ordenada de imágenes (*frames*) capturadas a intervalos regulares. Mientras una imagen RGB tiene dimensiones H×W×3 (alto, ancho, canales de color), un *clip* de video agrega la dimensión temporal, resultando en un tensor de forma T×H×W×3 donde T representa el número de *frames*. Esta representación volumétrica motiva la extensión de operaciones convolucionales al dominio tridimensional.

Las convoluciones 3D extienden los *kernels* convolucionales para operar simultáneamente en las dimensiones espaciales y temporal. Mientras un *kernel* 2D de dimensiones 3×3 detecta patrones espaciales locales, un *kernel* 3D de dimensiones 3×3×3 detecta patrones espacio-temporales: configuraciones espaciales que coocurren con cambios específicos entre *frames* consecutivos. Tran, Bourdev, Fergus, Torresani, y Paluri (2015) demostraron que redes basadas en convoluciones 3D podían aprender representaciones espacio-temporales efectivas para reconocimiento de acciones, superando enfoques que procesaban las dimensiones espacial y temporal de forma separada.

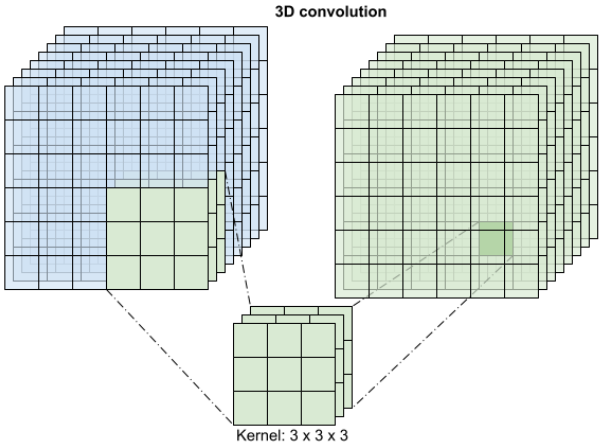


Figura 8. Convolución 3D: un *kernel* tridimensional (3×3×3) opera sobre un volumen espacio-temporal.

Las arquitecturas para video, como *SlowOnly* y *SlowFast* (Feichtenhofer, Fan, Malik, y He, 2019), aplican convoluciones 3D sobre secuencias de *frames* para capturar tanto información semántica (qué objetos y configuraciones están presentes) como información de movimiento (cómo cambian a través del tiempo). *SlowFast* emplea dos vías de procesamiento paralelas: una vía "lenta" que procesa pocos *frames* con muchos canales para capturar semántica, y una vía "rápida" que procesa más *frames* con menos canales para capturar movimiento fino. *SlowOnly* es una variante simplificada que utiliza únicamente la vía lenta, procesando *clips* de video a una única resolución temporal mediante convoluciones 3D uniformes. Esta simplificación reduce el costo computacional manteniendo desempeño competitivo en muchas tareas de reconocimiento de acciones.

Reconocimiento de acciones basado en poses

El reconocimiento de acciones basado en poses (*skeleton-based action recognition*) opera sobre secuencias temporales de *keypoints* en lugar de píxeles de video directamente. Esta representación ofrece varias ventajas: abstrae información irrelevante como vestimenta, fondo y condiciones de iluminación; reduce drásticamente la dimensionalidad de los datos; y captura directamente la información biomecánica relevante para análisis de movimiento. Para evaluación de técnica en ejercicios, donde lo relevante es la configuración y movimiento de articulaciones específicas, esta representación resulta particularmente apropiada.

ST-GCN: Grafos espacio-temporales

ST-GCN (Spatial Temporal Graph Convolutional Network), propuesto por Yan, Xiong, y Lin (2018), introdujo una representación del esqueleto humano como grafo donde los *keypoints* son nodos y las conexiones anatómicas (huesos) son aristas. Esta formulación permite aplicar convoluciones sobre grafos (*Graph Convolutional Networks*) para capturar relaciones entre articulaciones que no necesariamente son espacialmente adyacentes en la imagen pero están anatómicamente conectadas.

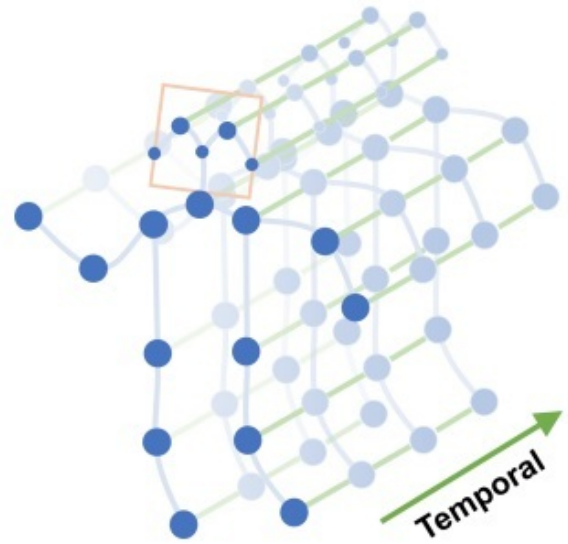


Figura 9. Grafo espacio-temporal de *ST-GCN*: nodos representando articulaciones conectadas espacialmente (dentro de cada *frame*) y temporalmente (entre *frames* consecutivos).

La arquitectura *ST-GCN* alterna convoluciones espaciales sobre el grafo (que propagan información entre articulaciones conectadas dentro de cada *frame*) con convoluciones temporales (que capturan cómo cambia cada articulación a través del tiempo). Este diseño modela explícitamente la estructura del cuerpo humano, asumiendo que articulaciones conectadas físicamente tienen mayor relevancia mutua que articulaciones distantes.

PoseConv3D: Convoluciones 3D sobre heatmaps de poses

PoseConv3D, desarrollado por Duan y cols. (2022), propone un enfoque alternativo para reconocimiento de acciones basado en poses que aplica convoluciones 3D estándar sobre secuencias temporales de mapas de calor (*heatmaps*) de *keypoints*, en lugar de operar sobre coordenadas discretas o grafos.

El procesamiento comienza generando un mapa de calor espacial para cada *keypoint* en cada *frame*, donde cada mapa es una distribución gaussiana centrada en las coordenadas del *keypoint* correspondiente. Estos mapas se apilan formando un volumen de dimensiones T×K×H×W, donde T es el número de *frames*, K el número de *keypoints* (17 para *COCO*), y H×W las dimensiones espaciales de los mapas de calor. Este volumen se procesa mediante una arquitectura de video estándar basada en convoluciones 3D, como *ResNet3D SlowOnly*.

La ventaja de este enfoque es que reutiliza directamente arquitecturas de video bien establecidas y optimizadas, sin requerir operadores especializados para grafos. Las convoluciones 3D capturan simultáneamente relaciones espaciales entre *keypoints* (qué configuraciones articulares son características de cada acción) y patrones temporales (cómo evolucionan estas configuraciones durante el movimiento). El modelo aprende implícitamente qué relaciones entre *keypoints* son relevantes para cada tarea de clasificación, sin imponer estructura de grafo predefinida.

PoseConv3D ha demostrado desempeño competitivo con arquitecturas basadas en grafos como *ST-GCN*, con la ventaja de simplicidad arquitectónica y eficiencia computacional al aprovechar implementaciones altamente optimizadas de convoluciones 3D disponibles en *frameworks* de aprendizaje profundo estándar.

2.6 Transfer Learning

El *transfer learning* (aprendizaje por transferencia) constituye una estrategia fundamental para entrenar modelos de aprendizaje profundo cuando los *datasets* específicos del dominio objetivo son limitados. En lugar de inicializar los pesos del modelo aleatoriamente y aprender todo desde cero, el *transfer learning* aprovecha conocimiento adquirido durante el entrenamiento en tareas relacionadas, típicamente con *datasets* mucho más extensos.

El principio subyacente es que las representaciones aprendidas en capas tempranas de redes profundas tienden a ser genéricas y transferibles entre tareas. Yosinski, Clune, Bengio, y Lipson (2014) demostraron que las primeras capas de redes convolucionales entrenadas en *ImageNet* aprenden detectores de bordes, texturas y patrones básicos que son útiles para prácticamente cualquier tarea de visión por computadora. Las capas intermedias capturan características de complejidad creciente que siguen siendo ampliamente aplicables. Solo las capas finales desarrollan representaciones altamente específicas de la tarea de entrenamiento original.

El *fine-tuning* (ajuste fino) es la estrategia más común de *transfer learning*, que consiste en inicializar un modelo con pesos preentrenados y continuar el entrenamiento en el *dataset* objetivo con tasa de aprendizaje reducida. Este proceso permite que el modelo adapte sus representaciones al nuevo dominio mientras preserva el conocimiento útil adquirido previamente. Es frecuente reemplazar completamente la capa de clasificación final (diseñada para las clases del *dataset* de preentrenamiento) con una nueva capa adaptada a las clases del problema objetivo, mientras se mantienen y ajustan las capas de extracción de características.

Una variante es el congelamiento de capas (*frozen stages*), donde los pesos de ciertas capas (típicamente las más tempranas) se mantienen fijos durante el *fine-tuning*, actualizando únicamente las capas superiores. Esta estrategia es particularmente útil cuando el *dataset* objetivo es muy pequeño: las capas congeladas preservan representaciones robustas aprendidas con grandes cantidades de datos, evitando que el *fine-tuning* las degrade por *overfitting* al *dataset* limitado.

El *transfer learning* resulta especialmente valioso en dominios donde los *datasets* etiquetados son limitados pero existen modelos preentrenados en tareas relacionadas con grandes volúmenes de datos. Por ejemplo, en reconocimiento de acciones basado en poses, existen *datasets* de referencia como *NTU RGB+D* (Shahroudy, Liu, Ng, y Wang, 2016) con decenas de miles de videos de acciones humanas diversas. Los modelos preentrenados en estos *datasets* capturan representaciones genéricas de movimiento humano que pueden transferirse a dominios específicos con *datasets* más reducidos, como aplicaciones médicas, deportivas o industriales, proporcionando un punto de partida significativamente mejor que la inicialización aleatoria.

Capítulo 3

Desarrollo e Implementación

Este capítulo describe el proceso de desarrollo desde la conceptualización inicial de la solución hasta la implementación de cada componente y su integración en un *pipeline* completo para evaluación automatizada de técnica a partir de videos de ejercicio.

Se documentan las decisiones de diseño, el proceso de desarrollo iterativo, la resolución de desafíos técnicos y la arquitectura final que alcanza los resultados reportados en el Capítulo 4.

Por fines didácticos la evolución en cada aspecto del proyecto será documentada siguiendo un orden lineal y lógico, pero cabe aclarar que los avances en cada componente fueron paralelos y no necesariamente en el orden presentado.

3.1 Arquitectura Inicial del Sistema

El planteo inicial de la arquitectura se basó en un principio fundamental entendido tras experiencias previas en proyectos de aprendizaje automático: la calidad y estructura de los datos determina en gran medida el éxito del entrenamiento de modelos.

Por lo tanto, las tareas iniciales a resolver implican:

1. Obtener un *dataset* de videos suficientemente extenso para entrenar modelos de clasificación robustos.
2. Extraer información biomecánica relevante desde los videos (secuencias temporales de poses corporales).
3. Estructurar los datos extraídos en formatos compatibles con *frameworks* de entrenamiento de modelos.
4. Entrenar y optimizar iterativamente los modelos hasta conseguir resultados aceptables.

Los pasos incluidos en el *pipeline* se idearon originalmente como ilustra el flujo de trabajo de la Figura 10:

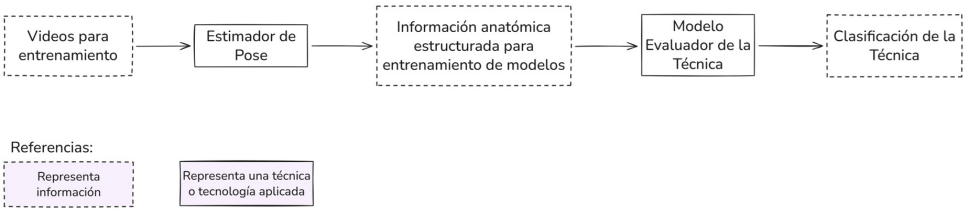


Figura 10. Flujo de trabajo del *pipeline* de clasificación.

3.2 Dataset

El proyecto comenzó con una perspectiva amplia de análisis kinesiológico general, considerando múltiples tipos de ejercicios y movimientos.

Se necesitaban videos capturados en diversos contextos: distintos ángulos de cámara, condiciones de iluminación variadas y sujetos con diferentes características físicas, entre otros factores. Esta diversidad resulta fundamental para asegurar que el modelo generalice adecuadamente los patrones de ejecución correcta.

Considerando estos requerimientos, la idea de crear un *dataset* propio fue descartada por razones de viabilidad y tiempo.

Durante la investigación de abordajes similares al problema planteado, se identificó el trabajo de Parmar y cols. (2022), *Domain Knowledge-Informed Self-supervised Representations for Workout Form Assessment*, donde se presenta un sistema de detección de errores de postura en tres ejercicios: sentadilla trasera, press militar y remo con barra. Este trabajo incluye el *dataset Fitness-AQA*, que se adoptó como base de datos para el proyecto.

El proyecto utiliza específicamente el subconjunto de *press* de hombro del *dataset Fitness-AQA* (Parmar y cols., 2022), que contiene 2,135 videos etiquetados en tres categorías: técnica correcta, errores de codos (apertura excesiva) y errores de rodillas (inestabilidad).

El *dataset* está disponible mediante formulario de solicitud para uso no comercial con cita apropiada de los autores.

Este hallazgo condujo a reformular el alcance del proyecto, enfocándose exclusivamente en el ejercicio *press* de hombro. Esta decisión se fundamenta en dos razones principales:

- Enfocarse en un ejercicio facilita profundizar en desafíos técnicos específicos y validar la metodología propuesta de manera rigurosa antes de considerar su extensión a otros movimientos.
- Permite validación rigurosa comparando contra el *benchmark* establecido por Parmar y cols. (2022) para el mismo ejercicio, con el mismo *dataset*.

El subconjunto de *press* de hombro está compuesto por:

- 2,135 videos en formato MP4 con resolución variable
- Duración promedio entre 2 y 5 segundos
- Formato de color *RGB* estándar
- Clips procesados en ventanas de 48 *frames* para entrenamiento

La distribución de videos por categoría es aproximadamente: técnica correcta (900 videos, 42 %), errores de codos (650 videos, 30 %) y errores de rodillas (585 videos, 28 %). Esta distribución presenta un desbalance moderado con predominio de la clase correcta, lo cual refleja la composición original del *dataset Fitness-AQA*.

3.3 Módulo Estimador de Pose

Para la extracción de *keypoints* anatómicos desde los videos de ejercicio se decidió utilizar *ViTPose*, un modelo basado en *Vision Transformers* para estimación de poses humanas.

La implementación de *ViTPose* se realizó utilizando el *framework easy_ViTPose* en una *notebook* de Google Colab de autoría propia[1], con aceleración GPU. Se seleccionó la configuración del modelo *ViT-B* (base) entrenado para el formato de *keypoints COCO wholebody*, que detecta 133 puntos anatómicos por persona.

[1] La *notebook* de extracción de *keypoints* está disponible en su versión original: https://colab.research.google.com/drive/1dzVgXmzdZLNV3O9QVqyqI-AIXBaim5dX, y en una versión comentada en español: https://colab.research.google.com/drive/1rfsN_JvYnRYqqZMJF-r025tSyiHDjfOi. El contenido de cada etapa se describe en detalle a lo largo de esta sección.

La configuración en la *notebook* incluye:

- Modelo: *ViTPose-B* (base) con *checkpoint* preentrenado
- Detector de personas: *YOLOv8-s* para detección multi-persona
- *Dataset* de entrenamiento del modelo: *COCO wholebody*

Debido a los requerimientos computacionales significativos del procesamiento de 2,135 videos, la extracción de *keypoints* se realizó mediante un *script* de procesamiento por lotes en Google Colab para respetar los límites de la licencia gratuita de la herramienta de Google. Este enfoque resultó necesario para manejar dos limitaciones principales: el tiempo de procesamiento (las horas de GPU continua necesarias) y los límites de sesión continua que Google Colab impone.

El procesamiento se estructuró procesando aproximadamente 600 videos por sesión, montando Google Drive para acceso a los datos y almacenamiento de resultados. El código implementado verifica la existencia del último archivo de salida antes de procesar cada video, permitiendo reanudar el procesamiento sin duplicar trabajo.

3.3.1 Estructura de Salida (JSON)

YOLOv8 detecta personas y genera *bounding boxes*, luego *ViTPose* toma cada región detectada y extrae 133 *keypoints* dentro de ella.

Al finalizar el procesamiento por lotes, se genera un archivo JSON por cada video procesado con la siguiente estructura jerárquica:

```
{
  "keypoints": [
    // Frame 0
    {
      "44": [  // ID de persona (string)
        [118.3, 189.5, 0.70],    // Keypoint 0: nariz
        [116.6, 190.8, 0.68],    // Keypoint 1: ojo izquierdo
        [116.6, 187.4, 0.67],    // Keypoint 2: ojo derecho
        // ... (130 keypoints mas)
      ],
      "43": [  // Otra persona en el mismo frame
        [102.5, 260.8, 0.35],
        [99.4, 254.2, 0.44],
        // ... (131 keypoints mas)
      ]
    },
    // Frame 1
    {
      "44": [
        [118.8, 189.2, 0.71],
        [116.9, 190.5, 0.69],
        // ...
      ],
      "43": [
        [102.8, 261.1, 0.36],
        // ...
      ]
    },
```

```
    // Frame 2
    {
      "44": [...],
      "51": [...]  // Nuevo ID
    }
  ],

  "skeleton": {
    "0": "nose",
    "1": "left_eye",
    "2": "right_eye",
    "3": "left_ear",
    "4": "right_ear",
    "5": "left_shoulder",
    "6": "right_shoulder",
    "7": "left_elbow",
    "8": "right_elbow",
    "9": "left_wrist",
    "10": "right_wrist",
    "11": "left_hip",
    "12": "right_hip",
    "13": "left_knee",
    "14": "right_knee",
    "15": "left_ankle",
    "16": "right_ankle",
    // ... (indice de los puntos anatomicos del esquema COCO)
  }
}
```

Fragmento de Código 3.1. Estructura de salida JSON de ViTPose.

Cada video se representa como una colección temporal de *frames*, donde cada *frame* es un diccionario con IDs de personas como claves. Cada persona contiene sus 133 *keypoints* como coordenadas (x, y) en píxeles junto con un valor de confianza.

Un aspecto fundamental descubierto durante la implementación es que *ViTPose* realiza detección *frame*-por-*frame* de manera independiente, sin mantener seguimiento temporal (*tracking*) de personas entre *frames* consecutivos. *ViTPose* procesa cada imagen de manera aislada aplicando el *Vision Transformer* sobre parches de la imagen para detectar *keypoints*, pero no incorpora información temporal del *frame* anterior ni posterior.

Esto causa que los IDs asignados a cada persona cambien inconsistentemente entre *frames* cuando ocurren oclusiones temporales o movimientos complejos. Por ejemplo, la persona con ID “44” en el *frame* 10 puede desaparecer temporalmente y reaparecer con ID “51” en el *frame* 12, a pesar de ser la misma persona físicamente. Esta inconsistencia imposibilita el seguimiento coherente del sujeto principal a lo largo del video.

Esta limitación tuvo implicaciones significativas para la estructuración de los datos de entrenamiento, requiriendo desarrollo de soluciones específicas documentadas en la sección siguiente.

3.4 Preparación de Keypoints para Entrenamiento

Entre las alternativas posibles para el modelo Evaluador de Técnica, se seleccionó *PoseConv3D* por su posicionamiento como arquitectura robusta en reconocimiento de acciones basado en poses. Esta selección se fundamenta en tres aspectos principales: primero, *PoseConv3D* está diseñado específicamente para procesar secuencias temporales de *keypoints* corporales, alineándose directamente con la naturaleza de los datos extraídos por *ViTPose*; segundo, forma parte del *framework MMAction2*, que ofrece implementaciones optimizadas y documentación robusta para experimentación; tercero, su arquitectura basada en convoluciones 3D permite capturar tanto relaciones espaciales entre articulaciones como patrones temporales del movimiento, resultando apropiado para detectar errores biomecánicos que se manifiestan a lo largo de la secuencia completa del ejercicio.

PoseConv3D requiere que los datos de entrenamiento estén en formato *pickle*, con una estructura específica documentada en las secciones siguientes. La transformación desde JSON (salida de *ViTPose*) hacia *pickle* (entrada de *PoseConv3D*) involucró un proceso iterativo de desarrollo seguido de un *pipeline* de tres etapas. Las iteraciones documentan la evolución del enfoque para seleccionar y rastrear al sujeto principal en videos multi-persona. Las etapas describen el procesamiento final implementado para estructurar los datos correctamente.

3.4.1 Iteraciones de Desarrollo: Selección del Sujeto Principal

Las siguientes iteraciones documentan la evolución del enfoque para identificar y rastrear consistentemente al sujeto ejecutando el ejercicio en videos con múltiples personas detectadas.

Iteración 1: Selección por Tamaño de Bounding Box

El enfoque inicial consiste en seleccionar la persona con el *bounding box* más grande en cada *frame*, asumiendo que sería el sujeto principal. Solo los *keypoints* de esta persona se incluyen en los datos de entrenamiento. El siguiente ejemplo ilustra este procesamiento:

```
// JSON original (Frame 10):
{
  "44": {
    "keypoints": [[118.3, 189.5, 0.70], [116.6, 190.8, 0.68], ...],
    "bbox": [98, 150, 120, 200]  // area = 24,000 px^2
  },
  "43": {
    "keypoints": [[102.5, 260.8, 0.35], [99.4, 254.2, 0.44], ...],
    "bbox": [80, 220, 190, 85]   // area = 16,150 px^2
  }
}

// JSON post-procesamiento (solo persona con mayor bounding box):
{
  "44": {
    "keypoints": [[118.3, 189.5, 0.70], [116.6, 190.8, 0.68], ...],
    "bbox": [98, 150, 120, 200]
  }
}
```

Fragmento de Código 3.2. Ejemplo de selección por bounding box.

La limitación identificada tras entrenar con este enfoque es que en algunos videos la persona ejecutando el ejercicio no siempre tiene el *bounding box* más grande. Este método causa que el sistema seleccione inconsistentemente entre el sujeto principal y personas en segundo plano según su tamaño aparente en cada *frame*, generando datos de entrenamiento que mezclan *keypoints* de diferentes personas.

Iteración 2: Procesamiento Multi-Persona Completo

Este enfoque descarta la selección prematura y mantiene todas las personas detectadas en cada *frame*, manteniendo información completa para permitir al modelo aprender a identificar el sujeto principal en contextos realistas multi-persona. Sin embargo, se identificó un problema crítico: los IDs de personas cambiaban inconsistentemente entre *frames* consecutivos a pesar de que correspondían a las mismas personas físicas. Esta limitación surge de la arquitectura de *ViTPose* (documentada en Sección 3.3), que procesa cada *frame* de manera independiente sin mantener seguimiento temporal entre *frames*.

El siguiente ejemplo real ilustra el problema en los *frames* 18-22 del video “62794_6.json”:

```
// Frame 18 - Ambas personas visibles
{
  "keypoints": [{
    "44": [[118.3, 189.5, 0.70], [129.5, 197.0, 0.79], ...],
    "43": [[265.2, 250.8, 0.82], [270.1, 245.3, 0.85], ...]
  }]
}

// Frame 20 - Oclusion parcial
{
  "keypoints": [{
    "43": [[267.5, 251.8, 0.81], [271.2, 246.1, 0.82], ...]
    // Persona "44" no detectada (temporalmente oculta)
  }]
}

// Frame 21 - Persona reaparece con nuevo ID
{
  "keypoints": [{
    "43": [[268.2, 252.1, 0.80], ...],
    "51": [[119.5, 191.2, 0.68], [130.2, 198.1, 0.76], ...]
    // Nueva persona "51" en posicion coherente con "44"
  }]
}
```

Fragmento de Código 3.3. Problema de inconsistencia de IDs entre frames.

A pesar de que las posiciones espaciales corresponden a las mismas personas, los IDs se reasignaron inconsistentemente. Este fenómeno ocurre por oclusiones temporales donde una persona queda oculta momentáneamente, movimientos rápidos que dificultan el emparejamiento automático, o errores de detección por condiciones de iluminación o calidad de cámara. El

resultado es que el modelo interpreta erróneamente que el video contiene tres personas: “44” con secuencia incompleta de 20 *frames*, “51” con 28 *frames* sin contexto inicial, y “43” como ruido de fondo. Estas secuencias fragmentadas imposibilitan que el modelo aprenda patrones temporales coherentes del movimiento completo del ejercicio a lo largo de los 48 *frames* de la ventana de entrenamiento.

Iteración 3: Algoritmo de Corrección de Tracking

La solución desarrollada, que constituye una contribución técnica original de este proyecto, consiste en un algoritmo de corrección basado en similitud de poses que mantiene la consistencia de IDs entre *frames*. Antes de describir el algoritmo, resulta necesario definir las métricas utilizadas para comparar poses entre *frames*.

La distancia euclidiana (también llamada distancia L2) es una métrica geométrica que mide la distancia en línea recta entre dos puntos en el espacio. Para dos puntos en el plano bidimensional con coordenadas (x_1, y_1) y (x_2, y_2), la distancia euclidiana se calcula mediante la fórmula derivada del teorema de Pitágoras:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.1}$$

Para comparar poses completas, el algoritmo calcula la distancia euclidiana entre cada par de *keypoints* correspondientes (nariz con nariz, hombro izquierdo con hombro izquierdo, etc.) y promedia estas distancias para obtener una medida de similitud global entre dos poses. Esta métrica resulta robusta para identificar si dos detecciones en *frames* diferentes corresponden a la misma persona, ya que personas distintas tendrán configuraciones espaciales significativamente diferentes incluso si están en posiciones cercanas en el *frame*.

El algoritmo implementado sigue el siguiente principio: si una persona con ID A desaparece del *frame* N (no es detectada) y reaparece una persona con ID B en el *frame* N+k (donde $k \leq 3$ *frames*), se compara la pose de A en el último *frame* donde fue visible con la pose de B en su primer *frame* de aparición. Si la distancia euclidiana promedio entre *keypoints* es menor a 25 píxeles, se reasigna el ID A a esta nueva detección, asumiendo que corresponde a la misma persona física. El umbral de 3 *frames* permite tolerar oclusiones breves sin perder el *tracking*, mientras que el umbral de 25 píxeles se estableció empíricamente como balance entre precisión (evitar fusionar personas diferentes) y robustez (mantener *tracking* ante movimientos moderados).

El siguiente ejemplo corregido ilustra el funcionamiento del algoritmo:

Secuencia temporal:

- *Frame* 10: Persona ID 44 visible en posición promedio (100, 200)
- *Frame* 11: Persona ID 44 no detectada (oclusión temporal)
- *Frame* 12: Persona ID 44 no detectada (continúa oculta)
- *Frame* 13: Aparece persona ID 52 en posición promedio (105, 198)

Análisis del algoritmo:

- Gap temporal: 2 *frames* (menor a límite de 3 *frames*)
- Distancia euclidiana promedio entre pose en *frame* 10 y pose en *frame* 13: $d_{promedio} \approx 5{,}4$ px
- Como 5.4 px <25 px, el algoritmo reasigna ID 44 a la detección del *frame* 13
- En los datos procesados, la persona mantiene ID 44 consistentemente

Esta solución representa una contribución técnica original de este proyecto, agregando consistencia temporal a la detección *frame*-por-*frame* de *ViTPose*. Los resultados experimentales demostraron que esta corrección resultó diferencial para el desempeño del modelo en las rondas finales de entrenamiento.

Solución Final Adoptada

La solución completa al problema de selección del sujeto principal integra los aprendizajes de las tres iteraciones anteriores. El enfoque final procesa todas las personas detectadas en cada video (Iteración 2) y aplica el algoritmo de corrección de *tracking* (Iteración 3) para mantener IDs consistentes a lo largo del video. Una vez corregida la consistencia temporal, se identifica el sujeto principal mediante un criterio heurístico basado en presencia temporal: la persona que posee la mayor cantidad de *frames* válidos tras la corrección de *tracking* y el filtrado de confianza se considera el sujeto ejecutando el ejercicio.

Este criterio de selección difiere fundamentalmente de la Iteración 1 en dos aspectos. Primero, la decisión se toma considerando la totalidad del video en lugar de *frame* por *frame*, evaluando qué persona mantiene presencia consistente a lo largo de toda la secuencia temporal. Segundo, el criterio es temporal (cantidad de *frames* donde aparece) en lugar de espacial (tamaño de *bounding box* en cada *frame*). Este enfoque asume razonablemente que la persona ejecutando el ejercicio aparece de manera prominente y consistente durante todo el video, mientras que instructores, personas en segundo plano o transeúntes tienden a aparecer parcialmente o en menor cantidad de *frames*. La combinación de *tracking* corregido con selección por presencia temporal resultó en datos de entrenamiento con secuencias coherentes del sujeto principal, habilitando que el modelo aprenda patrones biomecánicos temporales del ejercicio completo.

3.4.2 Pipeline de Procesamiento de Datos

Una vez desarrollada la solución para selección del sujeto principal documentada en la Sección 3.4.1, se implementó un *pipeline* de procesamiento secuencial para transformar los archivos JSON generados por *ViTPose* en el formato *pickle* requerido por *PoseConv3D*. Este formato es un método de serialización nativo de Python que permite almacenar estructuras de datos complejas (diccionarios anidados, *arrays* multidimensionales de *NumPy*, metadatos) en archivos binarios optimizados. El formato *pickle* se ha convertido en estándar para *datasets* de aprendizaje profundo debido a su velocidad de lectura durante el entrenamiento y su capacidad de preservar tipos de datos nativos sin necesidad de *parsing* o conversión, a diferencia de formatos basados en texto como JSON o CSV que requieren deserialización y conversión de tipos en cada lectura.

El procesamiento se divide en tres etapas que se ejecutan secuencialmente sobre el *dataset* completo de 2,135 videos. La primera etapa consiste en la transformación de los datos de pose desde el formato JSON de *ViTPose* hacia la estructura *pickle* compatible con *PoseConv3D*. Esta transformación se implementó como una *notebook* de *Jupyter*[2] que se ejecuta tres veces de forma separada, una por cada categoría de técnica: videos con ejecución correcta, videos con errores de codos, y videos con errores de rodillas. El procesamiento independiente por categoría permite controlar la división de datos entrenamiento/validación/test de manera balanceada para cada clase.

[2]La *notebook* de procesamiento está disponible en su versión original: https://colab.research.google.com/drive/1EbPZ_o69Gzn-2a-GvBmwqSJDQYTq2y3H, y en una versión comentada en español: https://colab.research.google.com/drive/1Gtrr7zFJtvn0Is1syKzkroWaglvrcTdU.

El primer subproceso aplicado es la reducción de *keypoints*. *ViTPose*, utilizando su modelo *whole-body*, detecta 133 puntos anatómicos por persona distribuidos en 17 puntos corporales del estándar *COCO*, 21 puntos por mano, 6 puntos por pie, y 68 puntos faciales. Sin embargo, para el análisis de técnica en *press* de hombro, solo resultan relevantes los 17 puntos corporales *COCO* estándar: nariz (0), ojos izquierdo y derecho (1-2), orejas izquierda y derecha (3-4), hombros (5-6), codos (7-8), muñecas (9-10), caderas (11-12), rodillas (13-14) y tobillos (15-16). Los 116 puntos restantes correspondientes a manos, pies y expresiones faciales se descartan, ya que no aportan información relevante para detectar errores de codos o rodillas en este ejercicio específico. Esta reducción de dimensionalidad no solo simplifica los datos sino que también reduce significativamente el costo computacional del entrenamiento.

Tras la reducción de *keypoints*, se aplica un filtrado por confianza para eliminar detecciones poco fiables. Cada *keypoint* viene acompañado de un valor de confianza entre 0 y 1 que indica la certeza del modelo sobre la ubicación detectada. Se estableció un umbral de confianza de 0.3, por debajo del cual los *keypoints* se marcan como inválidos asignándoles coordenadas (0.0, 0.0) y confianza 0.0. Esta representación permite mantener la estructura fija de 17 *keypoints* por *frame* que *PoseConv3D* requiere, marcando los puntos inválidos con valores nulos en lugar de eliminarlos. El filtrado previene que ruido o detecciones erróneas contaminen el conjunto de entrenamiento. Adicionalmente, se eliminan *keypoints* con valores numéricos inválidos como NaN que ocasionalmente aparecen por errores en el procesamiento de *ViTPose*.

Posteriormente se aplica la solución de selección del sujeto principal desarrollada en la Sección 3.4.1: el algoritmo de corrección de *tracking* para mantener IDs consistentes, seguido de la selección de la persona con mayor cantidad de *frames* válidos como sujeto ejecutante del ejercicio. Esta solución asegura que los datos de entrenamiento contengan secuencias temporalmente coherentes del sujeto principal, eliminando fragmentaciones de IDs y descartando personas secundarias.

Finalmente, el módulo divide los datos procesados en conjuntos de entrenamiento y validación siguiendo una distribución de 80 % y 20 % respectivamente, asegurando que la distribución de clases se mantenga balanceada en cada conjunto. El resultado de este primer paso son tres archivos *pickle* independientes, uno por cada categoría de error, conteniendo las secuencias de 17 *keypoints COCO* (coordenadas x, y, y valor de confianza) por *frame* para cada video, con IDs de persona consistentes y metadatos de división entrenamiento/validación.

La segunda etapa del *pipeline* consiste en la unificación de categorías de error. Hasta este punto, los datos procesados están organizados en tres conjuntos separados según la etiqueta de técnica. Para entrenar un modelo multi-clase capaz de distinguir simultáneamente entre las tres categorías, se desarrolló un *script* de fusión que combina estos tres conjuntos en un único *dataset* unificado. Este *script* lee secuencialmente los archivos correspondientes a técnica correcta, errores de codos y errores de rodillas, consolidando sus anotaciones en una estructura única mientras preserva las divisiones originales de entrenamiento/validación de cada categoría. La fusión resulta necesaria porque *PoseConv3D* requiere un único archivo de entrada para entrenamiento multi-clase.

La tercera y última etapa es la validación y corrección de metadatos. Durante las pruebas iniciales de entrenamiento con el *dataset* unificado, se identificó un problema crítico: inconsistencias entre el campo *total_frames* en los metadatos de cada video y la cantidad real de *keypoints* disponibles tras el procesamiento. Esta discrepancia surgía porque durante la primera etapa se descartaban *frames* con detecciones inválidas o donde el sujeto principal no era visible, pero el campo *total_frames* se calculaba basándose en el conteo original del video antes del filtrado. Por ejemplo, los metadatos podían indicar que un video tiene 50 *frames*, pero tras el filtrado por confianza y la selección del sujeto principal, solo quedaban 45 *frames* con

keypoints válidos. Durante el entrenamiento, *PoseConv3D* carga datos en *batches* (grupos de videos procesados simultáneamente para eficiencia computacional). Si los metadatos indican 50 *frames* pero el archivo contiene solo 45, la operación de carga del *batch* falla causando errores de tiempo de ejecución que interrumpen el entrenamiento completo. Para resolver esto, se implementó un *script* de validación que recorre el *dataset* unificado contando los *frames* reales disponibles por cada video y corrigiendo el campo *total_frames* para reflejar la cantidad exacta de *keypoints* presentes. Este *script* realiza adicionalmente validaciones de consistencia sobre las etiquetas de clase, las divisiones de conjunto (entrenamiento/validación) y las rutas de archivos referenciadas, asegurando que toda la información esté sincronizada correctamente.

La salida final del *pipeline* completo es un *dataset* validado en formato *pickle* de 2,135 videos etiquetados, donde cada video está representado por una secuencia temporal de 17 *keypoints COCO* a lo largo de sus *frames*. Cada *keypoint* se almacena como un triplete de coordenada x, coordenada y, y valor de confianza. Las secuencias mantienen consistencia temporal gracias al algoritmo de corrección de *tracking*, los metadatos están validados y sincronizados con el contenido real de *keypoints*, y las divisiones de entrenamiento/validación están preservadas. Esta estructura es completamente compatible con el formato de entrada esperado por *PoseConv3D*, que durante el entrenamiento se encarga internamente de transformar estas coordenadas en representaciones de *heatmaps* tridimensionales para alimentar su arquitectura de red convolucional 3D.

La Figura 11 resume el *pipeline* completo desarrollado, integrando los componentes documentados en las secciones anteriores.

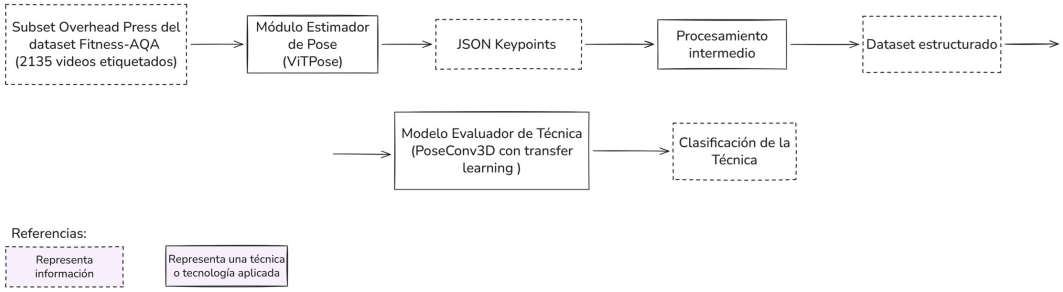


Figura 11. Arquitectura detallada del *pipeline* implementado.

3.5 Proceso Iterativo de Entrenamiento del Modelo

Una vez completado el *pipeline* de preparación de datos documentado en las secciones anteriores, es posible iniciar el proceso de entrenamiento del modelo clasificador *PoseConv3D*.

Este proceso no siguió un camino lineal exitoso, sino que requirió múltiples iteraciones experimentales para identificar y resolver problemas de generalización, muchas de estas mejoras son debidas a los cambios en los pasos de preprocesamiento. Esta sección documenta incrementalmente las configuraciones probadas, los problemas identificados mediante análisis de curvas de entrenamiento, y las soluciones implementadas hasta alcanzar el modelo final.

3.5.1 Experimento Base: Identificación del Problema de Overfitting

La configuración inicial del modelo se diseñó siguiendo parámetros estándar reportados en la literatura de reconocimiento de acciones, sin aplicar técnicas de regularización específicas para *datasets* pequeños.

Arquitectura del modelo:

La arquitectura *ResNet3D SlowOnly* combina dos conceptos fundamentales: *ResNet (Residual Network)*, una familia de redes neuronales que utiliza conexiones residuales para facilitar el entrenamiento de redes muy profundas, y 3D refiriéndose a que las operaciones convolucionales procesan información tridimensional (ancho, alto, y tiempo).

El término *SlowOnly* hace referencia al esquema de procesamiento temporal: esta variante toma clips de video y procesa todos los *frames* con la misma resolución temporal, aplicando convoluciones 3D uniformemente a lo largo de la secuencia. Esto contrasta con arquitecturas alternativas como *SlowFast* que emplean vías separadas procesando *frames* a diferentes tasas temporales para capturar simultáneamente información semántica (qué está pasando) de movimiento lento e información de movimiento rápido (cómo es el movimiento).

Por ejemplo, para un video de ejercicio con 150 *frames* totales, *SlowOnly* toma un clip de 48 *frames* uniformemente espaciados (por ejemplo, *frames* 1, 4, 7, 10... hasta el 148) y procesa esos 48 *frames* con la misma resolución temporal. En contraste, *SlowFast* dividiría el procesamiento en dos vías: una vía "lenta" que procesaría solo 8 *frames* del clip (*frames* 1, 22, 43, 64...) con muchos canales para capturar semántica, y una vía "rápida" que procesaría 32 *frames* (*frames* más frecuentes) con pocos canales para capturar movimiento. *SlowOnly* simplifica esta arquitectura usando una sola vía que procesa todos los 48 *frames* uniformemente.

Para la tarea de clasificación de errores de forma en ejercicios, *SlowOnly* resulta particularmente apropiada por dos razones. Primero, la entrada al modelo son secuencias de *keypoints* corporales, no *frames* RGB completos; esta representación compacta elimina la redundancia espacial masiva del video original, reduciendo significativamente el costo computacional y haciendo viable procesar todos los *frames* uniformemente. Segundo, la naturaleza de los errores a detectar es principalmente postural/geométrica: determinar si la configuración articular (posición de codos, alineación de rodillas) es correcta o incorrecta. Estos errores se manifiestan en la configuración espacial de las articulaciones más que en dinámicas de movimiento ultra-rápidas, por lo que procesar 48 *frames* uniformemente espaciados del ejercicio proporciona suficiente contexto temporal.

El modelo se compone de los siguientes elementos:

- **Entrada:** 17 canales correspondientes a los 17 *keypoints COCO*. Durante el entrenamiento, *PoseConv3D* transforma internamente las coordenadas de cada *keypoint* en mapas de calor (*heatmaps*) espaciales que codifican la posición y confianza del *keypoint*, generando representaciones de 56×56 píxeles por *keypoint* y por *frame*. Un *heatmap* es una representación bidimensional donde cada píxel contiene un valor de probabilidad indicando qué tan probable es que el *keypoint* esté ubicado en esa posición, típicamente modelado como una distribución gaussiana centrada en la coordenada detectada.
- **Backbone (red troncal):** *ResNet3D SlowOnly* con 32 canales base. El *backbone* es la red neuronal principal que extrae características progresivamente más abstractas de los datos de entrada. Los canales representan las diferentes características aprendidas en cada capa; 32 canales base significa que la primera capa convolucional genera 32 mapas de características distintos, cada uno capturando diferentes aspectos de los datos como bordes, texturas o patrones de movimiento.
- **Etapas convolucionales:** 3 *stages* con bloques (4, 6, 3). Un *stage* es una sección de la red donde las operaciones convolucionales operan a la misma resolución espacial y temporal. Los números (4, 6, 3) indican que el primer *stage* contiene 4 bloques residuales secuenciales, el segundo contiene 6 bloques, y el tercero contiene 3 bloques. Cada bloque residual es una unidad que aprende transformaciones de las características mientras mantiene conexiones directas (residuales) que facilitan el flujo de gradientes durante el

entrenamiento.
Esta configuración específica de bloques por *stage* no es arbitraria, sino que deriva de arquitecturas *ResNet* estándar adaptadas para procesamiento 3D de video. La distribución refleja un principio de diseño jerárquico: el primer *stage* con 4 bloques procesa características de bajo nivel (bordes, texturas, patrones básicos de movimiento); el segundo *stage* con 6 bloques procesa características de nivel medio como configuraciones articulares y partes del cuerpo, siendo donde ocurre el aprendizaje de representaciones más críticas; el tercer *stage* con 3 bloques procesa características de alto nivel (poses completas, patrones biomecánicos globales) requiriendo menos bloques porque las características ya están bien refinadas de etapas anteriores. Esta configuración representa un balance entre capacidad del modelo (suficiente para aprender patrones complejos) y eficiencia computacional (no excesivamente profundo para evitar *overfitting* en *datasets* pequeños), resultando apropiada para *datasets* de tamaño moderado al balancear capacidad de aprendizaje con riesgo de sobreajuste.

- **Clasificador:** *I3DHead* con *dropout* 0.5 para 3 clases. El *I3DHead (Inflated 3D Head)* es la capa final que toma las características extraídas por el *backbone* y las transforma en predicciones de clase. *Dropout* es una técnica de regularización que durante el entrenamiento desactiva aleatoriamente un porcentaje de las neuronas (en este caso 50 %) para prevenir que el modelo dependa excesivamente de neuronas específicas, forzándolo a aprender representaciones más robustas y distribuidas.

Configuración de entrenamiento:

El entrenamiento se configuró con los siguientes parámetros:

- **Optimizador:** SGD (*Stochastic Gradient Descent*, Descenso de Gradiente Estocástico) con *learning rate* 0.05 y *momentum* 0.9. SGD es el algoritmo fundamental que actualiza los pesos de la red neuronal iterativamente para minimizar el error. En cada iteración, calcula el gradiente (dirección de mayor aumento) de la función de pérdida respecto a los parámetros del modelo y los ajusta en dirección opuesta para reducir el error. El *learning rate* (tasa de aprendizaje) de 0.05 controla el tamaño de cada paso de actualización: valores altos permiten convergencia rápida pero arriesgan inestabilidad, mientras valores bajos garantizan estabilidad a costa de entrenamiento más lento. El *momentum* de 0.9 agrega inercia al proceso: acumula un 90 % de la dirección de actualización previa, ayudando a acelerar el entrenamiento y superar mínimos locales al mantener velocidad en direcciones consistentes.
- **Weight decay:** 0.0003. Esta técnica de regularización añade una penalización proporcional a la magnitud de los pesos del modelo en la función de pérdida (regularización L2), incentivando que los pesos permanezcan pequeños.
  La regularización L2 previene *overfitting* porque los modelos sobreajustados típicamente desarrollan pesos extremadamente grandes para memorizar ruido y detalles específicos del conjunto de entrenamiento. Al penalizar la magnitud de los pesos, L2 fuerza al modelo a aprender solo patrones robustos y generales, distribuyendo el aprendizaje entre muchos parámetros en lugar de confiar en pocos pesos extremos para ajustarse a particularidades de los datos de entrenamiento.
- **Epochs:** 24. Una época representa un ciclo completo de entrenamiento donde el modelo ha procesado todos los ejemplos del conjunto de entrenamiento exactamente una vez. Durante 24 épocas, el modelo verá el *dataset* completo 24 veces, permitiendo refinamiento progresivo de los pesos.
- **Dataset:** *fitness-aqa.pkl* con 3 clases (correcto, errores de codos, errores de rodillas)
- **Data augmentation:** Las transformaciones de aumentación aplicadas incluyen *flip* horizontal con probabilidad 0.5 (reflejo especular izquierda-derecha aplicado a los *heatmaps*,

simulando perspectivas inversas), *RandomResizedCrop* que recorta aleatoriamente regiones con áreas entre 56 % y 100 % del tamaño original y las escala al tamaño de entrada (introduciendo variabilidad en escala y posicionamiento), y redimensionamiento final a 56×56 píxeles. Estas transformaciones se aplican aleatoriamente durante el entrenamiento a los *heatmaps* generados internamente por *PoseConv3D*, lo cual implica que cada época el modelo ve versiones ligeramente diferentes de los mismos videos, incrementando la diversidad efectiva de datos sin requerir videos adicionales.

El entrenamiento del modelo del experimento base reveló un patrón problemático característico de *overfitting* severo. El *overfitting* (sobreajuste) ocurre cuando el modelo aprende a memorizar los ejemplos de entrenamiento en lugar de extraer patrones generalizables, resultando en excelente desempeño en datos vistos pero pobre capacidad para clasificar datos nuevos.

Las curvas de pérdida mostraron divergencia pronunciada: la pérdida de entrenamiento descendió consistentemente, mientras que la pérdida de validación se mantuvo alta y errática sin mostrar tendencia descendente. Esta brecha creciente entre ambas curvas constituye la evidencia definitiva de *overfitting*: el modelo minimiza el error en los datos de entrenamiento pero no logra reducir el error en el conjunto de validación que contiene ejemplos no vistos previamente.

Las métricas de precisión confirmaron esta interpretación. La precisión mide el porcentaje de predicciones correctas sobre el total de ejemplos. La precisión de entrenamiento alcanzó valores muy altos, mientras que la precisión de validación fluctuó erráticamente sin convergencia. El modelo aprendió a clasificar perfectamente los ejemplos vistos durante entrenamiento, pero falló sistemáticamente en generalizar a nuevos ejemplos del conjunto de validación.

Este comportamiento se atribuye a la combinación de dos factores principales. El primer factor es el desbalance severo entre la complejidad del modelo y el tamaño del *dataset*: aproximadamente 1,708 videos de entrenamiento resultan insuficientes para entrenar desde cero una red convolucional 3D con millones de parámetros.

El modelo *ResNet3D SlowOnly* utilizado posee aproximadamente 12 millones de parámetros, una cantidad considerable que se explica por la naturaleza tridimensional de las convoluciones. A diferencia de las redes 2D que procesan solo información espacial, las convoluciones 3D operan sobre volúmenes espacio-temporales: donde una convolución 2D con *kernel* 3×3 contiene 9 pesos, su equivalente 3D con *kernel* $3 \times 3 \times 3$ contiene 27 pesos. Esta multiplicación por tres en cada *kernel* se amplifica a través de las múltiples capas y canales de la red, resultando en millones de parámetros totales. Con solo 1,281 ejemplos de entrenamiento, la proporción resulta en aproximadamente 9,400 parámetros por video, muy distante de heurísticas comunes en aprendizaje profundo que sugieren al menos 10-100 ejemplos por parámetro para entrenamiento robusto. Esta desproporción convierte el problema en matemáticamente indeterminado, donde existen infinitas combinaciones de pesos que pueden ajustarse perfectamente a los datos de entrenamiento, favoreciendo soluciones que memorizan en lugar de generalizar.

El segundo factor es que las técnicas de regularización básicas aplicadas (*dropout* de 0.5, *weight decay* de 0.0003, aumentación de datos mediante *flip* y *crop*) resultaron insuficientes para contrarrestar el severo desbalance entre complejidad del modelo y cantidad de datos. Estas técnicas, aunque efectivas en escenarios más equilibrados, no lograron prevenir que el modelo se ajustara excesivamente a las particularidades de los ejemplos de entrenamiento en lugar de aprender características biomecánicas generales del movimiento.

3.5.2 Iteración con Transfer Learning con Fine-Tuning

Identificado el problema de *overfitting*, se implementó una segunda configuración experimental incorporando dos estrategias complementarias: *transfer learning* con *fine-tuning* desde un modelo preentrenado en el *dataset NTU60* y regularización adicional mediante suavizado de etiquetas.

Transfer Learning con Fine-Tuning:

A diferencia del experimento base que entrenaba el modelo desde cero con pesos inicializados aleatoriamente, esta iteración adoptó *transfer learning* utilizando un *checkpoint* de *ResNet3D SlowOnly* preentrenado en *NTU60*. Este *checkpoint* proviene del *framework MMAction2/pyskl*, ampliamente utilizado en la comunidad de investigación en reconocimiento de acciones, y fue entrenado en *NTU60*, un *dataset* de referencia con 60 clases de acciones y aproximadamente 56,000 videos de movimientos humanos diversos (Shahroudy y cols., 2016). El modelo preentrenado ya aprendió representaciones genéricas de movimiento humano durante su entrenamiento previo: las capas iniciales detectan patrones de bajo nivel como bordes y texturas, las capas intermedias reconocen partes corporales y configuraciones articulares, y las capas superiores codifican patrones de movimiento complejos. Estas representaciones, aunque aprendidas para reconocer acciones generales, resultan transferibles al dominio específico de evaluación de ejercicios porque los movimientos fundamentales del cuerpo humano son consistentes entre dominios (Yosinski y cols., 2014).

El proceso de *fine-tuning* no parte del modelo del experimento base de la primera iteración, sino que utiliza un modelo diferente: el *checkpoint* preentrenado en *NTU60*. Ambos modelos poseen aproximadamente 12 millones de parámetros totales porque comparten la misma arquitectura *ResNet3D SlowOnly*, pero el origen de sus pesos difiere sustancialmente: el experimento base tenía pesos inicializados aleatoriamente que resultaron en *overfitting*, mientras que el modelo *NTU60* posee pesos ya optimizados mediante entrenamiento en 56,000 videos de movimientos humanos diversos.

La adaptación del modelo *NTU60* al problema de evaluación de ejercicios requiere dos modificaciones principales. Primero, se descarta completamente el clasificador original (diseñado para 60 clases de acciones) y se reemplaza con un clasificador nuevo de 3 salidas (correcto, error de codos, error de rodillas), cuyos pesos se inicializan aleatoriamente. Segundo, se cargan los pesos preentrenados del *backbone* mediante el parámetro de configuración *load_from*. De esta forma, el entrenamiento comienza con un *backbone* que ya sabe reconocer movimiento humano (aprendido de *NTU60*), pero con un clasificador nuevo que debe aprender a distinguir específicamente entre técnica correcta e incorrecta en *press* de hombro.

Adicionalmente, se experimentó con congelamiento parcial de capas (*frozen stages*) para preservar características de bajo nivel aprendidas en *NTU60* mientras se permite que las capas superiores se adapten al nuevo dominio. El congelamiento implica fijar los pesos de las primeras etapas convolucionales del modelo para que no se actualicen durante el entrenamiento, preservando intactas las representaciones de bajo nivel aprendidas en el *dataset* original. Se probaron configuraciones con *frozen_stages*=1 (congelar *stage* 1 que detecta patrones básicos como bordes y texturas) y *frozen_stages*=2 (congelar *stages* 1 y 2 que detectan patrones de bajo y medio nivel). Con *frozen_stages*=2, el modelo conserva sus 12 millones de parámetros totales, pero solo 7.5 millones se actualizan durante el entrenamiento: el *stage* 3 del *backbone* (aproximadamente 6 millones de parámetros) y el clasificador nuevo (aproximadamente 1.5 millones de parámetros), mientras que los 4.5 millones de parámetros de los *stages* 1 y 2 permanecen congelados con sus valores originales de *NTU60*. Esta estrategia balancea entre aprovechar conocimiento preentrenado robusto en las capas tempranas y permitir adaptación al dominio

específico de evaluación de ejercicios en las capas superiores.

Regularización mediante Suavizado de Etiquetas:

Se incorporó suavizado de etiquetas como técnica adicional de regularización en la función de pérdida (Szegedy y cols., 2016). Las etiquetas verdaderas son las etiquetas asignadas a cada video en el *dataset* (p. ej., un video etiquetado como "correcto" o como "error de codos"). Tradicionalmente, estas etiquetas se representan mediante vectores *one-hot* que expresan certeza absoluta: [1, 0, 0] indica 100 % de certeza en la primera clase y 0 % en las demás. El suavizado de etiquetas, implementado mediante la función *SmoothedCELoss* de la librería *pyskl*, suaviza estas distribuciones redistribuyendo una pequeña fracción de probabilidad hacia las otras clases. Con $\varepsilon = 0{,}1$ (el parámetro de suavizado configurado), la etiqueta [1, 0, 0] se transforma en [0.9, 0.05, 0.05], distribuyendo 10 % de probabilidad uniformemente entre las clases no verdaderas.

Esta técnica previene que el modelo genere predicciones extremadamente confiadas en una sola clase, comportamiento que frecuentemente conduce a *overfitting*. Cuando el modelo aprende a predecir con certeza absoluta (p. ej., [0.99, 0.005, 0.005]) en el conjunto de entrenamiento, tiende a memorizar patrones específicos de esos ejemplos en lugar de aprender características robustas y generalizables. Al forzar al modelo a mantener cierta incertidumbre mediante el suavizado de etiquetas, se incentiva que aprenda representaciones más robustas que generalizan mejor a datos nuevos, funcionando como una forma de regularización implícita que penaliza la sobreconfianza.

Evaluación de la iteración:

Tras incorporar *transfer learning* con *fine-tuning* y suavizado de etiquetas, las curvas de entrenamiento mostraron mejora sustancial. La brecha entre pérdida de entrenamiento y validación se redujo significativamente, indicando que el modelo generalizaba mejor. La precisión de validación alcanzó valores más estables, representando mejora considerable respecto al experimento base.

Sin embargo, el desempeño aún no alcanzaba niveles competitivos para aplicación práctica. Se identificó que el modelo requería optimizaciones adicionales en la estrategia de entrenamiento y aumentación de datos.

3.5.3 Optimización Avanzada: Aumentación de Datos y AdamW

La tercera iteración experimental introdujo dos optimizaciones adicionales: aumentación de datos mediante perturbaciones controladas de *keypoints* y cambio del optimizador SGD a *AdamW* con ajuste de hiperparámetros.

Aumentación de Datos con Pose Jittering:

Se implementó aumentación de datos mediante *PoseJitter*, una transformación que añade ruido gaussiano a las coordenadas de los *keypoints* durante el entrenamiento. Esta técnica simula variabilidad natural en la detección de poses que ocurre en condiciones reales debido a imprecisiones del estimador *ViTPose*, variaciones en iluminación, o ambigüedad en la posición exacta de articulaciones. Al exponer el modelo a versiones ligeramente perturbadas de cada pose durante el entrenamiento, se incrementa la robustez del modelo ante estas imprecisiones, mejorando su capacidad de generalización.

Los parámetros configurados fueron:

- $\sigma = 2{,}0$: desviación estándar del ruido gaussiano en píxeles, controlando la magnitud típica de las perturbaciones

- $clip = 5{,}0$: valor máximo de perturbación para evitar desplazamientos excesivos que distorsionen de forma poco realista la pose
- *same_on_sequence*=True: aplica la misma perturbación a todos los *frames* de una secuencia, preservando coherencia temporal del movimiento

Matemáticamente, para cada *keypoint* con coordenadas (x, y), se genera una perturbación $\delta \sim \mathcal{N}(0, \sigma^2)$ recortada en $[-clip, +clip]$, resultando en coordenadas aumentadas $(x + \delta_x, y + \delta_y)$. Esta transformación se aplica solo durante entrenamiento, no durante validación o evaluación.

Optimizador AdamW y Ajuste de Hiperparámetros:

Se reemplazó el optimizador SGD con momento por *AdamW (Adam with decoupled Weight decay)* (Loshchilov y Hutter, 2019). A diferencia de SGD que aplica la misma tasa de aprendizaje a todos los parámetros del modelo, *AdamW* ajusta automáticamente cuánto modificar cada parámetro según su comportamiento histórico durante el entrenamiento (Kingma y Ba, 2015). Los parámetros que muestran gradientes consistentes se actualizan más agresivamente, mientras que aquellos con gradientes inestables se modifican de forma más conservadora. Esta adaptación individual permite al modelo aprender más eficientemente, especialmente en arquitecturas profundas donde distintas capas requieren estrategias de actualización diferentes. La variante *AdamW* mejora sobre *Adam* estándar aplicando la regularización de *weight decay* de forma separada, lo que resulta en mejor control del *overfitting*.

La configuración final fue:

- Tasa de aprendizaje: 0.001 (reducción de 50x respecto a SGD 0.05). La tasa de aprendizaje más baja aprovecha la naturaleza adaptativa de *Adam*: como *Adam* ajusta tasas individuales, una tasa de aprendizaje global baja previene actualizaciones excesivamente agresivas mientras permite que parámetros individuales converjan a su ritmo óptimo.
- *Weight decay*: 0.001 (aumento de 3x respecto a 0.0003). El *weight decay* incrementado añade regularización L2 más agresiva, penalizando más fuertemente pesos grandes y forzando al modelo a aprender representaciones más simples.
- Política de tasa de aprendizaje: *CosineAnnealing* con *min_lr*=0. Esta política reduce gradualmente la tasa de aprendizaje siguiendo una función coseno desde el valor inicial hasta un mínimo cercano a cero. En épocas tempranas, la tasa de aprendizaje alta permite exploración amplia del espacio de parámetros; hacia el final del entrenamiento, la tasa de aprendizaje baja permite refinamiento fino para convergencia precisa.
- Recorte de gradiente: *max_norm*=40. Esta técnica limita la magnitud del gradiente durante el entrenamiento: si la norma del gradiente excede 40, se escala proporcionalmente para que su norma sea exactamente 40. Esto previene actualizaciones explosivas que pueden ocurrir cuando el gradiente es extremadamente grande, estabilizando el proceso de entrenamiento especialmente en las primeras épocas.

Configuración de dataset y entrenamiento:

Esta iteración mantuvo el enfoque de clasificación multi-clase (3 categorías: técnica correcta, errores de codos, errores de rodillas) sobre el *dataset* completo *Fitness-AQA*. Paralelamente, se completó la implementación del algoritmo de corrección de *tracking* documentado en Sección 3.4.1, aplicándose al *dataset* para asegurar coherencia temporal en las secuencias. Esta corrección resultó diferencial: eliminó las secuencias fragmentadas de identificadores inconsistentes que previamente impedían al modelo aprender patrones temporales coherentes del ejercicio completo, permitiendo ahora observar la progresión temporal completa del movimiento de un mismo individuo a lo largo de los 48 *frames*.

Se incrementó *videos_per_gpu*=8, parámetro que controla cuántos videos se procesan

simultáneamente en cada iteración de entrenamiento (*batch size*). El *batch size* más grande proporciona estimaciones de gradiente más estables basadas en mayor cantidad de ejemplos, acelerando la convergencia y reduciendo la variabilidad del entrenamiento. Sin embargo, está limitado por la memoria disponible de la GPU.

Se configuró *RepeatDataset* con *times*=2, una técnica que repite artificialmente el *dataset* dos veces por época. Esto expone el modelo a más ejemplos por época sin requerir datos adicionales: aunque los ejemplos son repetidos, las transformaciones aleatorias de *data augmentation* (*PoseJitter*, *RandomResizedCrop*, *Flip*) generan versiones distintas en cada repetición, incrementando efectivamente la diversidad de datos vistos por el modelo.

Configuración final alcanzada:

La combinación de *transfer learning* con *fine-tuning* desde *NTU60*, suavizado de etiquetas, *pose jittering*, optimizador *AdamW* con ajuste de hiperparámetros, y crucialmente el *dataset* con *tracking* corregido, produjo la configuración final del modelo. Las curvas de entrenamiento mostraron convergencia estable con brecha mínima entre entrenamiento y pérdida de validación, indicando capacidad de generalización satisfactoria.

Los resultados cuantitativos detallados del proceso iterativo se presentan en el Capítulo 4.

3.6 Análisis de Costos y Factibilidad

Esta sección presenta el análisis de costos incurridos durante el desarrollo del proyecto en contexto académico, las especificaciones de hardware necesarias para su replicación, una comparación con soluciones comerciales existentes, y el estudio de factibilidad para una potencial implementación profesional.

3.6.1 Costos del Proyecto Académico

El desarrollo del modelo de clasificación de técnica se llevó a cabo utilizando exclusivamente recursos disponibles en el ámbito académico y herramientas de código abierto. La Tabla 1 detalla los costos reales incurridos.

Concepto	Descripción	Costo
Recursos Humanos	Trabajo realizado por el estudiante en contexto de proyecto de grado	\$0
Hardware	Computadora de la universidad equipada con GPU para entrenamiento del modelo; Google Colab en modalidad gratuita para extracción de *keypoints*	\$0
Software	*ViTPose*, *PoseConv3D*, *MMAction2*, *pyskl*, *YOLOv8*, Python, PyTorch y librerías asociadas (todo código abierto)	\$0
Datos	*Dataset Fitness-AQA* disponible para uso académico con cita apropiada (Parmar y cols., 2022)	\$0
Total		**\$0**

Tabla 1. Costos del proyecto académico.

El uso de Google Colab en su modalidad gratuita implicó restricciones en tiempo de procesamiento continuo y disponibilidad de GPU. Esto requirió procesar los 2,135 videos del *dataset* en lotes distribuidos en múltiples sesiones, verificando la existencia del último archivo procesado antes de cada ejecución para evitar trabajo duplicado. Estas limitaciones ocasionaron atrasos en el desarrollo que podrían evitarse mediante una suscripción a Google Colab Pro (aproximadamente USD 10 mensuales) o el uso de servicios de computación en la nube dedicados como AWS, Google Cloud Platform o Azure.

3.6.2 Especificaciones de Hardware para Replicación

Para entrenar modelos de reconocimiento de acciones basados en poses similares al desarrollado en este proyecto, se requiere hardware con capacidad de procesamiento gráfico significativa. La Tabla 2 presenta las especificaciones recomendadas.

Componente	Especificación Recomendada
GPU	NVIDIA RTX 3090 (24 GB VRAM)
RAM	32 GB DDR4
Almacenamiento	500 GB SSD NVMe
CPU	Procesador de 8 o más núcleos (Intel i7/i9 o AMD Ryzen 7/9)

Tabla 2. Especificaciones de hardware recomendadas.

La selección de una GPU con 24 GB de VRAM se justifica por la naturaleza del procesamiento que realiza *PoseConv3D* (Duan y cols., 2022). Durante el entrenamiento, el modelo genera representaciones de *heatmaps* tridimensionales con dimensiones (C×T×H×W) correspondientes a canales, tiempo, alto y ancho. Estas estructuras, combinadas con el procesamiento de *batches* de múltiples videos simultáneamente, demandan memoria de video significativa. Una GPU con menor VRAM limitaría el tamaño del *batch* y extendería considerablemente los

tiempos de entrenamiento, aunque sigue siendo posible entrenar con GPUs de menor capacidad ajustando los hiperparámetros correspondientes.

3.6.3 Comparación con Soluciones Comerciales

El mercado actual ofrece diversas soluciones para análisis biomecánico y evaluación de forma en ejercicios. La Tabla 3 presenta una comparación con sistemas comerciales representativos.

Sistema	Descripción	Modelo de Costo
VICON / OptiTrack	Sistemas de captura de movimiento 3D con marcadores reflectivos	Hardware especializado: USD 10,000 - 100,000+
Physimax / Kaia Health	Plataformas de análisis de movimiento basadas en inteligencia artificial	Suscripción mensual o anual (USD 50-200/mes)
Tempo / Mirror	Dispositivos de entrenamiento con análisis de forma integrado	Hardware USD 1,500+ más suscripción mensual (USD 39/mes)

Tabla 3. Comparación con soluciones comerciales.

Es importante destacar que cada una de estas soluciones está diseñada para atender casos de uso específicos y diferentes segmentos del mercado. Los sistemas de captura de movimiento profesionales como VICON u OptiTrack están orientados a aplicaciones que requieren máxima precisión, como investigación biomecánica, animación cinematográfica y análisis deportivo de élite (Windolf, Götzen, y Morlock, 2008). Estos sistemas utilizan cámaras infrarrojas y marcadores reflectivos para lograr precisión submilimétrica, pero requieren instalaciones dedicadas, calibración especializada y personal técnico capacitado, lo que los posiciona fuera del alcance de gimnasios o centros de entrenamiento de menor escala.

Las plataformas basadas en inteligencia artificial como Physimax o Kaia Health ofrecen análisis de movimiento mediante video estándar, similar al enfoque de este proyecto. Están orientadas al mercado de rehabilitación física y entrenamiento remoto, ofreciendo interfaces amigables para usuarios finales pero operando como servicios propietarios con costos de suscripción recurrentes. Los dispositivos como Tempo o Mirror combinan hardware dedicado (pantallas interactivas con cámaras integradas) con software de análisis, apuntando al mercado de *fitness* doméstico premium.

La solución desarrollada en este trabajo se posiciona como una alternativa de bajo costo basada en tecnologías de código abierto. Procesa video RGB estándar sin requerir hardware especializado ni marcadores corporales, democratizando el acceso a herramientas de análisis biomecánico para instituciones educativas, centros de rehabilitación y gimnasios que de otra forma no podrían acceder a estas tecnologías. Una ventaja significativa de desarrollar una herramienta propia radica en la flexibilidad que ofrece: si una organización requiere funcionalidades específicas que las soluciones comerciales no proporcionan, o si los costos de licenciamiento resultan prohibitivos para el volumen de uso esperado, contar con una implementación propia permite adaptar el *pipeline* a necesidades puntuales sin depender de proveedores externos.

3.6.4 Estimación para Implementación Comercial

Se presenta un análisis hipotético de los costos que implicaría desarrollar una solución de este tipo para uso comercial, asumiendo que se cuenta con la metodología documentada en este trabajo. El modelo de negocio propuesto consiste en un servicio de API para procesamiento de videos de ejercicio, donde los clientes (gimnasios, aplicaciones de *fitness*, centros de rehabilitación) envían videos y reciben análisis de técnica.

Inversión en Hardware

La implementación de un servicio de procesamiento de videos requiere una estación de trabajo equipada con GPU de alto rendimiento y un servidor para el despliegue de la API. La Tabla 4 presenta los costos estimados.

Concepto	**Costo Est. (USD)**
GPU NVIDIA RTX 3090 (24 GB VRAM)	1,200 – 1,500
Componentes de estación de trabajo (CPU, RAM, almacenamiento, gabinete, fuente)	1,000 – 1,500
Servidor para despliegue de API	500 – 1,000
Total Hardware	**2,700 – 4,000**

Tabla 4. Inversión en hardware para implementación comercial.

Como alternativa al hardware físico, pueden considerarse servicios de computación en la nube. Instancias con GPU comparable (como AWS p3.2xlarge o Google Cloud con NVIDIA T4/V100) tienen costos aproximados de USD 3-4 por hora de uso, lo que resulta conveniente para cargas de trabajo variables pero puede ser costoso para procesamiento continuo.

Recursos Humanos

La estimación de recursos humanos considera que el equipo de desarrollo cuenta con el conocimiento metodológico documentado en este proyecto, incluyendo la arquitectura del *pipeline*, el procesamiento de datos, las configuraciones de entrenamiento optimizadas y las soluciones a problemas técnicos como la corrección de *tracking*. Sin este conocimiento previo, el tiempo de desarrollo se incrementaría considerablemente.

Rol	**Horas Est.**	**Costo/Hora (USD)**	**Subtotal (USD)**
Especialista en ML / Visión por Computadora	150 – 200	40 – 42	6,000 – 8,400
Desarrollador *backend* para integración de API	80 – 100	40 – 42	3,200 – 4,200
Total	**230 – 300**		**9,200 – 12,600**

Tabla 5. Estimación de recursos humanos para implementación comercial.

Los costos por hora reflejan estimaciones basadas en tarifas del mercado argentino para profesionales senior especializados en desarrollo de software e inteligencia artificial. Los

valores se obtuvieron de la plataforma Salancy (https://salarios.gonzalopozzo.com/), que recopila salarios reales del sector tecnológico argentino. Según datos de enero de 2026, los salarios mensuales promedio para perfiles senior con más de 5 años de experiencia son aproximadamente USD 6,800 para Data Scientist y USD 6,800 para Backend Developer, lo que equivale a tarifas horarias de USD 40-42 considerando jornadas de 160 horas mensuales. Estas cifras son consistentes con rangos reportados en plataformas internacionales como Glassdoor, Levels.fyi y Stack Overflow Developer Survey para mercados latinoamericanos. Un especialista en aprendizaje automático y visión por computadora con conocimientos de PyTorch, procesamiento de video y arquitecturas de redes neuronales convolucionales representa un perfil de alta demanda. El desarrollador *backend* debe tener experiencia en despliegue de modelos de aprendizaje automático y construcción de APIs escalables.

Consideraciones sobre el Dataset

El *dataset Fitness-AQA* utilizado en este proyecto fue publicado por Parmar y cols. (2022) y está disponible para uso académico con cita apropiada de los autores. Para una implementación comercial serían necesarias alternativas: obtener licencia comercial directamente de los autores del *dataset*, o desarrollar un *dataset* propio mediante grabación y etiquetado de videos de ejercicio.

El desarrollo de un *dataset* propio implicaría costos adicionales significativos que incluyen la grabación de videos con sujetos diversos, el etiquetado por expertos en biomecánica o entrenamiento físico, y la validación de las anotaciones. Se estima que para un ejercicio adicional (por ejemplo, sentadillas o peso muerto) con un volumen de datos comparable al utilizado en este proyecto, los costos oscilarían entre USD 3,000 y USD 5,000, desglosados aproximadamente en USD 1,000-1,500 para compensación de participantes y logística de grabación, y USD 2,000-3,500 para etiquetado experto con múltiples revisiones de calidad.

Expansión a Otros Ejercicios

El modelo actual está entrenado específicamente para el ejercicio de *press* de hombro. Extender la funcionalidad a otros ejercicios (sentadillas, peso muerto, press de banca) requeriría la obtención o grabación de videos etiquetados para cada ejercicio adicional, la definición de categorías de error específicas por ejercicio en consulta con profesionales de educación física o fisioterapia, y el re-entrenamiento del modelo o entrenamiento de modelos especializados. Cada ejercicio adicional representaría un costo incremental de desarrollo de aproximadamente USD 5,000-8,000 considerando datos y horas de especialista.

Resumen de Inversión Total

Consolidando las estimaciones anteriores, una implementación comercial básica para un único ejercicio requeriría una inversión inicial aproximada de USD 12,000-17,000 (hardware más recursos humanos más obtención de datos), con costos incrementales de USD 5,000-8,000 por cada ejercicio adicional que se desee incorporar al *pipeline*.

3.6.5 Análisis de Factibilidad

Factibilidad Económica

El proyecto es económicamente viable debido al uso exclusivo de tecnologías de código abierto que eliminan costos de licenciamiento de software. Las herramientas utilizadas, incluyendo *ViTPose*, *PoseConv3D*, *MMAction2*, PyTorch y las librerías de procesamiento de datos, están disponibles bajo licencias que permiten tanto uso académico como comercial sin costos asociados.

La principal inversión requerida corresponde a recursos humanos capacitados en aprendizaje profundo y visión por computadora, junto con hardware equipado con GPU para entrenamiento. Esta combinación de costos representa una barrera de entrada significativamente menor comparada con sistemas comerciales de análisis biomecánico. Mientras sistemas como VICON requieren inversiones de decenas de miles de dólares solo en hardware especializado (sin considerar instalación, calibración y mantenimiento), la solución propuesta puede implementarse con una inversión total inferior a USD 20,000 incluyendo hardware, recursos humanos y datos.

Desde la perspectiva de retorno de inversión, un servicio de API para análisis de técnica podría comercializarse mediante modelos de suscripción o pago por uso. Mediciones realizadas durante el desarrollo del proyecto, documentadas en los registros de ejecución del *notebook* de *Easy ViTPose*[3], indican que el *pipeline* de estimación de poses procesa aproximadamente 290 videos por hora utilizando una GPU NVIDIA T4 en *Google Colab*. Considerando tarifas de servicios de nube con GPU T4 de aproximadamente USD 0.50-0.80 por hora, el costo operativo de inferencia resulta en aproximadamente USD 0.003-0.005 por video analizado. Este costo marginal reducido, combinado con la inversión inicial documentada en las secciones anteriores, permite estructurar modelos de negocio con márgenes comerciales viables. Gimnasios, aplicaciones de *fitness* y centros de rehabilitación representan potenciales clientes que actualmente no tienen acceso a tecnologías de análisis biomecánico automatizado debido a los costos de las alternativas comerciales existentes.

Factibilidad Técnica

La implementación es técnicamente factible dado que las tecnologías utilizadas son maduras y cuentan con trayectoria comprobada tanto en investigación académica como en aplicaciones productivas. *ViTPose* alcanzó resultados estado del arte en *benchmarks* públicos de estimación de pose humana al momento de su publicación, mientras que *PoseConv3D* demostró rendimiento superior en tareas de reconocimiento de acciones basadas en esqueleto comparado con aproximaciones anteriores.

Los *frameworks* seleccionados cuentan con documentación extensa, tutoriales detallados y comunidades activas que proporcionan soporte. El ecosistema de PyTorch en particular se ha consolidado como estándar de facto para investigación y desarrollo en aprendizaje profundo, lo que facilita la contratación de personal capacitado y el acceso a recursos de aprendizaje.

Los requisitos de hardware son accesibles en el mercado actual. Las GPU de la serie NVIDIA RTX están disponibles comercialmente a través de múltiples distribuidores, y los servicios de computación en la nube ofrecen alternativas flexibles para organizaciones que prefieren evitar inversiones de capital en infraestructura física. La inferencia (procesamiento de nuevos videos una vez entrenado el modelo) puede ejecutarse en hardware menos demandante

[3] El log de ejecución registra tiempos de procesamiento por video y velocidad de inferencia en frames por segundo para cada video del *dataset*.

que el entrenamiento, lo que reduce los costos operativos del servicio en producción.

La arquitectura modular del *pipeline* permite actualizaciones incrementales. Si surgen modelos de estimación de pose más precisos en el futuro, pueden integrarse sin modificar el resto de los componentes. Igualmente, mejoras en el modelo de clasificación pueden implementarse de manera independiente, facilitando el mantenimiento y evolución del *pipeline* a largo plazo.

Factibilidad Operativa

El *pipeline* de procesamiento desarrollado está documentado y es reproducible. Los *scripts* de preprocesamiento, las configuraciones de entrenamiento y los procedimientos de evaluación están versionados y organizados de manera que permiten a otros desarrolladores comprender y replicar el proceso completo. El proceso de entrenamiento iterativo establecido permite ajustar y mejorar el modelo según se disponga de datos adicionales o se identifiquen patrones de error en el uso real del modelo.

La metodología es transferible a otros ejercicios siguiendo los pasos documentados de preparación de datos, configuración de entrenamiento y optimización. Una vez establecido el procedimiento para un ejercicio, la curva de aprendizaje para agregar ejercicios adicionales se reduce significativamente, ya que las decisiones arquitecturales fundamentales (formato de datos, hiperparámetros base, estrategia de *data augmentation*) pueden reutilizarse con ajustes menores.

El *pipeline* puede operarse como servicio sin requerir intervención continua una vez desplegado. La inferencia sobre nuevos videos es automática y determinística, sin requerir conocimientos especializados por parte del usuario final. La integración mediante API permite que aplicaciones cliente (móviles, web o de escritorio) incorporen la funcionalidad de análisis de técnica sin necesidad de ejecutar modelos de aprendizaje automático localmente.

Desde la perspectiva de escalabilidad, el procesamiento de videos puede paralelizarse mediante múltiples instancias del servicio o mediante procesamiento por lotes durante horarios de baja demanda. El cuello de botella principal es el tiempo de GPU requerido para la estimación de pose, pero este proceso es inherentemente paralelizable a nivel de video.

Factibilidad Legal

Las licencias de código abierto de las herramientas utilizadas permiten tanto uso académico como comercial. *MMAction2* se distribuye bajo licencia Apache 2.0, que permite uso, modificación y redistribución incluso con fines comerciales, requiriendo únicamente la preservación de avisos de *copyright* y atribución. PyTorch utiliza licencia BSD, igualmente permisiva. Las librerías complementarias (*NumPy*, *OpenCV*, *scikit-learn*) operan bajo licencias similares que no imponen restricciones significativas al uso comercial.

El *dataset Fitness-AQA* está disponible para uso académico mediante solicitud a los autores. Para uso comercial se requeriría negociar términos de licencia específicos con los autores originales, o desarrollar un *dataset* alternativo con videos grabados con consentimiento explícito de los participantes para uso comercial.

El *pipeline* desarrollado no almacena ni procesa datos personales sensibles más allá del contenido de los videos de ejercicio proporcionados para análisis. En caso de implementación comercial, se deberían establecer políticas de privacidad y manejo de datos acordes a la legislación vigente, incluyendo la Ley de Protección de Datos Personales de Argentina (Ley 25.326)

y regulaciones internacionales aplicables si se ofreciera el servicio a clientes en otras jurisdicciones. Los videos procesados contienen imágenes de personas realizando ejercicio, lo que constituye dato personal según las definiciones legales vigentes. Una implementación comercial debería considerar el almacenamiento seguro, la limitación de propósito, y los derechos de acceso y eliminación por parte de los titulares de los datos.

Capítulo 4

Resultados

Este capítulo presenta los resultados cuantitativos del modelo desarrollado, evaluando el desempeño del clasificador *PoseConv3D* tras el proceso iterativo de optimización documentado en el Capítulo 3. Se analizan las métricas de performance alcanzadas, se comparan las distintas configuraciones experimentales, y se discute la efectividad del enfoque propuesto para evaluación automatizada de técnica en ejercicios.

4.1 Progreso Iterativo del Modelo y Métricas

El proceso de desarrollo iterativo documentado en la sección 3.5 exploró tres configuraciones principales:

1. **Experimento Base:** Arquitectura *ResNet3D SlowOnly* entrenada desde cero sin regularización, evidenciando *overfitting* severo por tamaño limitado del *dataset*.
2. **Iteración con Transfer Learning con Fine-Tuning:** Incorporación de *transfer learning* con *fine-tuning* desde *NTU60* y suavizado de etiquetas, logrando reducción significativa de *overfitting* y mejora en generalización.
3. **Configuración Final:** Integración de *pose jittering*, optimizador *AdamW*, y crucialmente el *dataset* con *tracking* corregido, alcanzando convergencia estable del modelo.

Cada iteración abordó limitaciones específicas identificadas mediante análisis de curvas de entrenamiento. La corrección de *tracking* representó el componente diferencial que permitió al modelo aprender patrones temporales coherentes del movimiento completo. La Tabla 6 resume las configuraciones experimentales y sus métricas de performance correspondientes.

Aspecto	**Exp. Base**	**Transfer Learning**	**Config. Final**
Inicialización	Desde cero	Preentren. *NTU60*	Preentren. *NTU60*
Etapas congeladas	N/A	2	0
Función de pérdida	*CrossEntropy*	*SmoothedCE* ($\varepsilon = 0{,}1$)	*SmoothedCE* ($\varepsilon = 0{,}1$)
Optimizador	SGD (lr=0.05)	SGD (lr=0.05)	*AdamW* (lr=0.001)
Weight decay	0.0003	0.0003	0.001
Aumentación	*Flip*, *Crop*	*Flip*, *Crop*	*Flip*, *Crop*, *Jitter*
Dataset	3 clases	3 clases	3 clases + *tracking*
Precisión	**51.2 %**	**70.5 %**	**91.6 %**
F1-Score	**0.48**	**0.69**	**0.91**

Tabla 6. Comparación de configuraciones experimentales y métricas de performance.

El incremento progresivo en precisión de validación evidencia que cada optimización contribuyó acumulativamente al desempeño final. La configuración del experimento base alcanzó apenas 51.2 % de precisión, indicando que el modelo no logró aprender patrones útiles más allá del azar en un problema multiclase de 3 categorías. La iteración con *transfer learning*

con *fine-tuning* elevó la precisión a 70.5 %, demostrando que el conocimiento preentrenado en *NTU60* resultó transferible al dominio de evaluación de ejercicios. La configuración final, integrando *pose jittering*, optimizador *AdamW*, y crucialmente el *dataset* con *tracking* corregido, alcanzó 91.6 % de precisión y *F1-score* macro de 0.91.

Análisis de Curvas de Entrenamiento

Las curvas de pérdida y precisión durante el entrenamiento proveen información sobre la capacidad de generalización del modelo y la presencia de *overfitting*. Las figuras siguientes muestran la evolución del modelo a través de las iteraciones experimentales, con las líneas azules representando el conjunto de entrenamiento (*train*) y las líneas naranjas el conjunto de validación (*val*).

4.1.1 Experimento Base

La primera configuración experimental consistió en entrenar el modelo *ResNet3D SlowOnly* desde cero sin *transfer learning* ni técnicas de regularización, para establecer un punto de referencia.

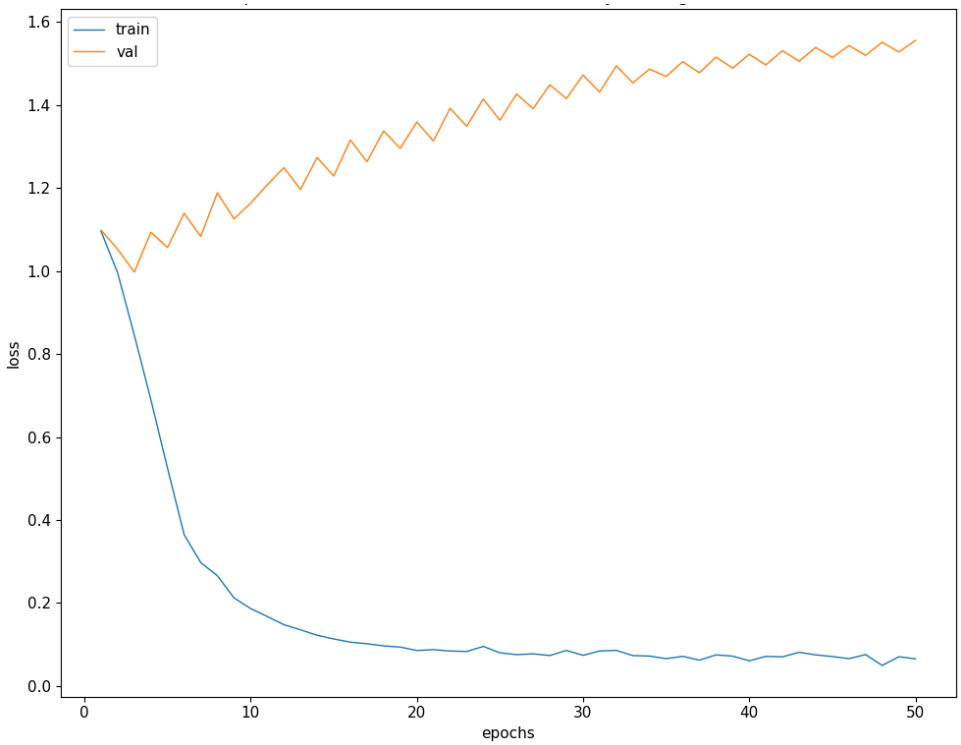


Figura 12. Curva de pérdida del experimento base.

La curva de pérdida (Figura 12) evidencia *overfitting* extremo desde épocas tempranas. La pérdida de entrenamiento (azul) desciende consistentemente hasta valores muy bajos (0.07), mientras la pérdida de validación (naranja) presenta tendencia ascendente con oscilaciones pronunciadas, alcanzando valores superiores a 1.5. Esta divergencia indica que el modelo memorizó los datos de entrenamiento sin desarrollar capacidad de generalización.

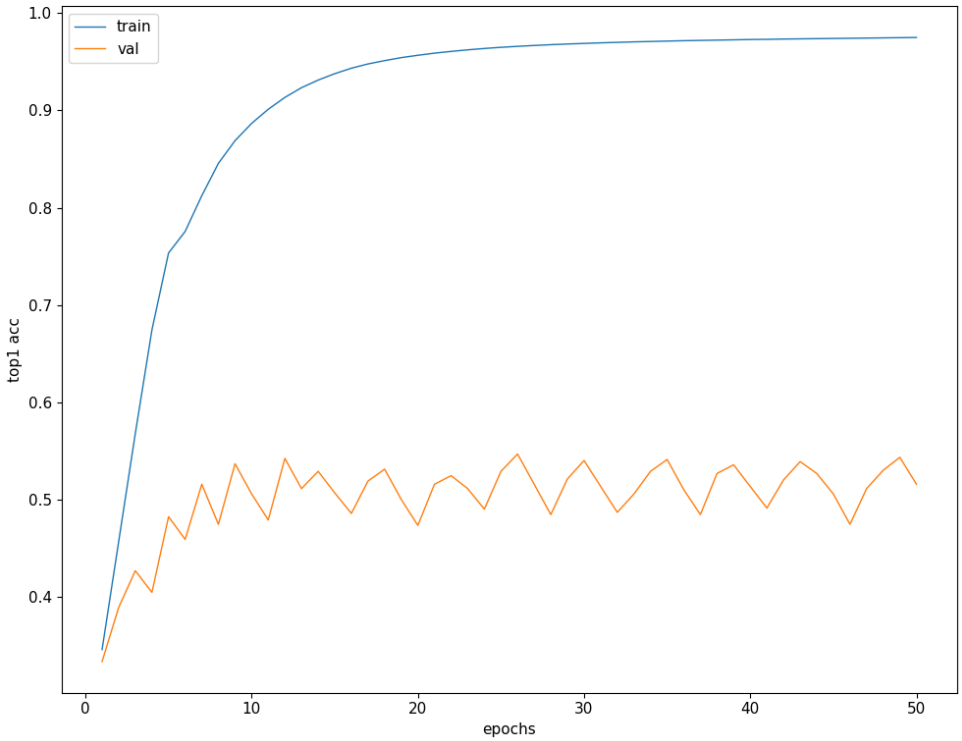


Figura 13. Curva de precisión del experimento base.

La curva de precisión (Figura 13) confirma el problema de *overfitting*. El modelo alcanza 97.5 % de precisión en entrenamiento (azul), pero la precisión de validación (naranja) oscila erráticamente alrededor de 51 %, apenas superior al rendimiento aleatorio esperado en un problema de 3 clases (33.3 %). Las oscilaciones caóticas en validación indican que el modelo no aprendió patrones robustos del ejercicio, sino que ajustó a peculiaridades específicas del conjunto de entrenamiento.

Este resultado confirmó que el tamaño limitado del *dataset* (2,135 videos totales) es insuficiente para entrenar arquitecturas profundas desde cero, motivando la adopción de *transfer learning* con *fine-tuning* en la siguiente iteración.

4.1.2 Iteración con Transfer Learning con Fine-Tuning

La segunda configuración incorporó *transfer learning* desde modelos preentrenados en *NTU60* con *fine-tuning* de capas superiores, junto con regularización mediante suavizado de etiquetas ($\varepsilon = 0{,}1$).

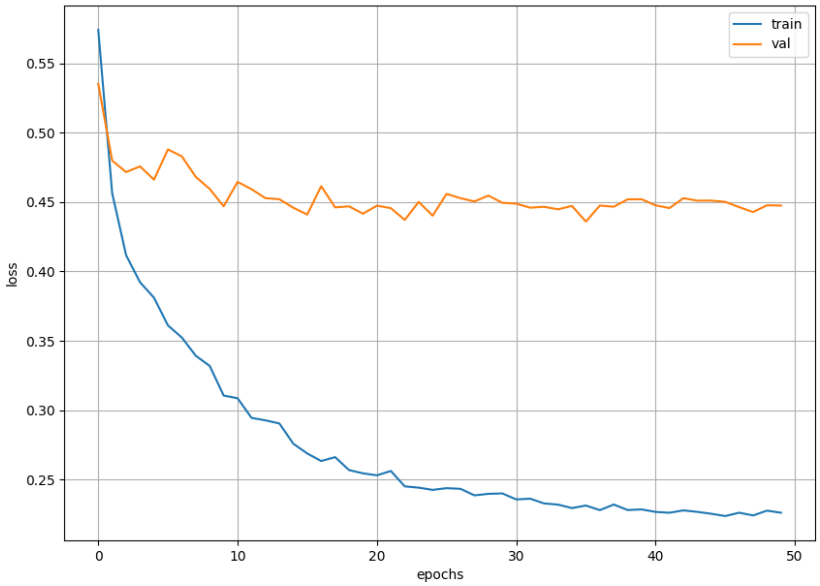


Figura 14. Curva de pérdida con *transfer learning* y *fine-tuning*.

La curva de pérdida (Figura 14) muestra mejora sustancial respecto al experimento base. La pérdida de entrenamiento (azul) desciende de forma controlada, mientras la pérdida de validación (naranja) mantiene tendencia descendente general con oscilaciones moderadas. La brecha entre ambas curvas se redujo significativamente, indicando que el *transfer learning* con *fine-tuning* permitió al modelo aprovechar conocimiento preentrenado sobre movimiento humano para mejorar generalización.

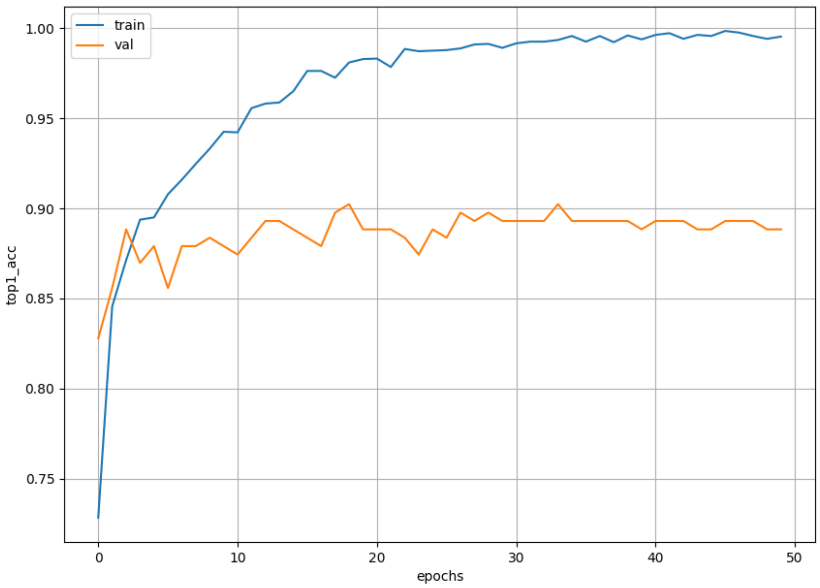


Figura 15. Curva de precisión con *transfer learning* y *fine-tuning*.

La curva de precisión (Figura 15) refleja el impacto positivo del *transfer learning* con *fine-tuning*. El modelo alcanza aproximadamente 95 % de precisión en entrenamiento (azul) mientras que validación (naranja) se estabiliza alrededor de 70 %. Esta brecha del 25 % indica *overfitting* moderado: el modelo logró aprender representaciones útiles del *dataset* de entrenamiento pero con capacidad de generalización limitada.

El salto de 51.2 % (experimento base) a 70.5 % (*transfer learning*) confirma que el conocimiento preentrenado en *NTU60* resultó fundamental. Sin embargo, la brecha del 25 % entre entrenamiento y validación indicaba que se requería optimización adicional.

4.1.3 Configuración Final

La configuración final integró todas las optimizaciones identificadas: *transfer learning* con *fine-tuning* desde *NTU60*, suavizado de etiquetas, *pose jittering*, optimizador *AdamW*, y crucialmente, el *dataset* con *tracking* corregido.

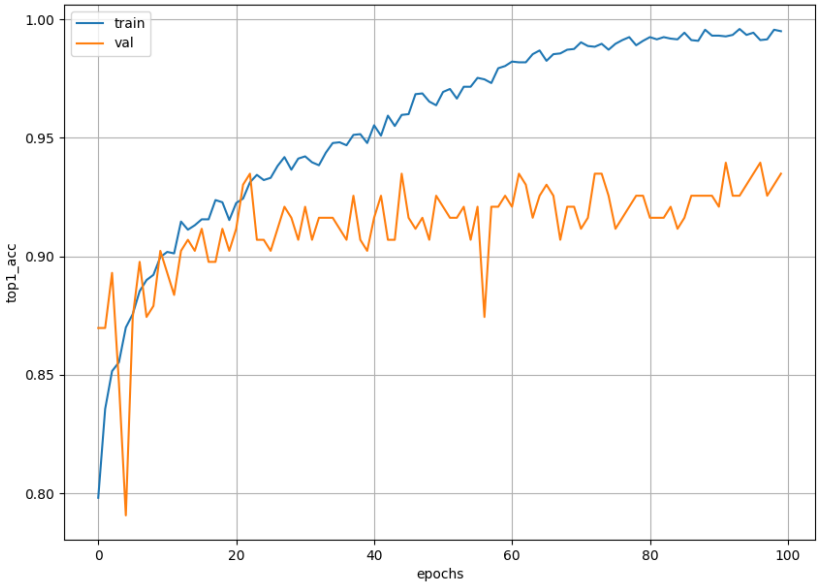


Figura 16. Curva de precisión de la configuración final.

El gráfico de precisión de la configuración final (Figura 16) revela un cambio cualitativo respecto a iteraciones previas. Las curvas de precisión de entrenamiento y validación convergen establemente y mantienen proximidad notable a lo largo de todo el entrenamiento. Características destacables:

- **Convergencia rápida y estable:** Ambas curvas alcanzan valores altos (>85 %) en épocas tempranas y continúan mejorando gradualmente sin oscilaciones pronunciadas.
- **Brecha mínima:** La precisión de entrenamiento alcanza 93-95 % mientras validación se estabiliza en 91.6 %, resultando en una brecha de apenas 2-4 %. Esto contrasta drásticamente con el 20-25 % de brecha observado en iteraciones previas.

El salto cualitativo de 70 % a 91.6 % en precisión de validación (*F1-score* macro: 0.91) se atribuye principalmente a la incorporación del *dataset* con *tracking* corregido. En el contexto de evaluación de técnica en ejercicios, una precisión de 91.6 % significa que el modelo clasifica correctamente aproximadamente 9 de cada 10 ejecuciones.

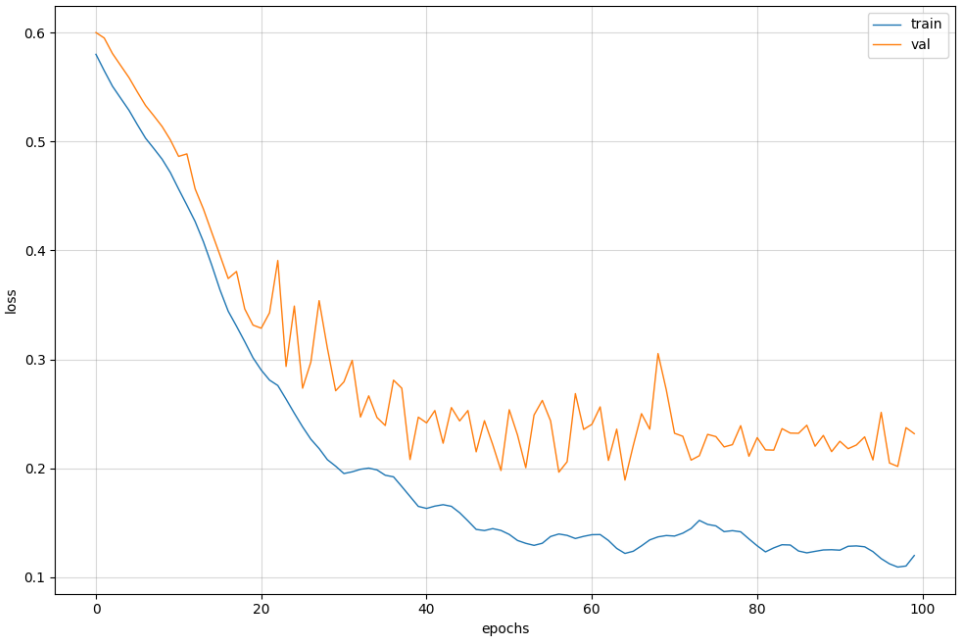


Figura 17. Curva de pérdida de la configuración final.

La curva de pérdida (Figura 17) complementa el análisis de la precisión, mostrando el comportamiento de la función de costo durante el entrenamiento. Ambas curvas (entrenamiento y validación) descienden de manera consistente y convergen hacia valores bajos, con una brecha reducida que se mantiene estable a lo largo del entrenamiento.

4.1.4 Análisis Detallado de Clasificación

Para comprender mejor el desempeño del modelo en cada categoría, se presenta la matriz de confusión y las métricas de clasificación por clase.

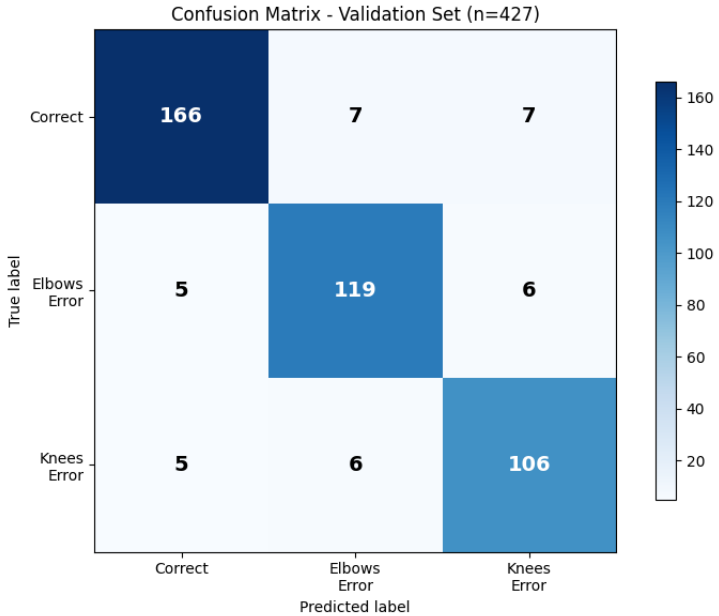


Figura 18. Matriz de confusión del modelo final en conjunto de validación.

La matriz de confusión muestra la distribución de predicciones del modelo para las 427

muestras del conjunto de validación. Los valores en la diagonal representan las clasificaciones correctas, mientras que los valores fuera de la diagonal indican errores de clasificación. Se observa que el modelo presenta desempeño consistente en las tres categorías, sin sesgos evidentes hacia ninguna clase particular.

Clase	Precision	Recall	F1-Score	Support
Correct	0.94	0.92	0.93	180
Elbows Error	0.90	0.92	0.91	130
Knees Error	0.89	0.91	0.90	117
Macro avg	**0.91**	**0.92**	**0.91**	**427**

Tabla 7. Métricas de clasificación por clase en conjunto de validación.

Las métricas por clase revelan un desempeño balanceado del modelo. La clase *Correct* obtiene las métricas más altas (F1=0.93), lo cual es esperable dado que representa la categoría con mayor cantidad de ejemplos en el *dataset*. Las clases de error (*Elbows Error* y *Knees Error*) presentan métricas ligeramente inferiores pero consistentes entre sí (F1=0.91 y F1=0.90 respectivamente), indicando que el modelo aprendió a distinguir ambos tipos de errores con capacidad similar.

4.2 Discusión

Los resultados obtenidos validan la viabilidad del enfoque propuesto para evaluación automatizada de técnica en ejercicios mediante análisis de secuencias de poses. El modelo desarrollado alcanza desempeño comparable a sistemas de análisis biomecánico en tareas de clasificación de movimiento, utilizando únicamente video RGB estándar sin requerir equipamiento especializado como sistemas de captura de movimiento 3D.

La arquitectura basada en la combinación de *ViTPose* para extracción de poses y *PoseConv3D* para clasificación temporal demostró ser efectiva para capturar tanto la configuración espacial de las articulaciones en cada *frame* como los patrones dinámicos del movimiento a lo largo de la secuencia completa del ejercicio.

En relación al trabajo de referencia sobre el *dataset* utilizado, Parmar y cols. (2022) abordaron el mismo problema con un enfoque de aprendizaje auto-supervisado que entrena dos clasificadores binarios independientes: uno para errores de codos (F1 = 0.4552) y otro para errores de rodillas (F1 = 0.8452). El presente trabajo, en cambio, entrena un único clasificador multi-clase que distingue simultáneamente entre técnica correcta, errores de codos y errores de rodillas, alcanzando F1 macro de 0.91. La diferencia metodológica y tecnológica entre ambos trabajos hace que la comparación sea referencial antes que directa; sin embargo, los resultados obtenidos son mejores en ambas categorías de error.

Los aportes concretos de este trabajo comprenden: el *pipeline* completo de preprocesamiento desde video crudo hasta formato compatible con *PoseConv3D*, incluyendo el algoritmo de corrección de *tracking* desarrollado específicamente para este proyecto; la metodología de entrenamiento iterativo documentada con experimentos reproducibles; y la configuración final optimizada que combina *transfer learning* desde *NTU60*, *pose jittering* y *AdamW*. Estos artefactos constituyen una base técnica replicable para trabajos futuros en análisis automatizado de movimiento humano.

El modelo aprende a aproximar la evaluación humana a partir de patrones temporales de pose, tomando como referencia el criterio del profesional que etiquetó los videos del *dataset*. Esta característica implica que la calidad de las predicciones depende de la consistencia y conocimiento experto del etiquetado original; al mismo tiempo, permite capturar aspectos sutiles de la evaluación profesional que serían difíciles de formalizar mediante reglas explícitas. El proceso de etiquetado del *dataset Fitness-AQA* fue realizado por dos entrenadores profesionales de gimnasio contratados por Parmar y cols. (2022), lo que garantiza la calidad del criterio de referencia utilizado para el entrenamiento.

El desempeño final de 91.6% de precisión y 0.91 de *F1-score* macro en la tarea de clasificación multiclase (3 categorías: correcto, errores de codos, errores de rodillas) establece un resultado sólido considerando el tamaño limitado del *dataset* (2,135 videos totales, 1,708 videos de entrenamiento tras división 80/20). Este resultado es particularmente relevante dado que el *dataset* contiene variabilidad real en condiciones de captura (ángulos de cámara, iluminación, características físicas de los sujetos) que el modelo debe generalizar.

Sin embargo, el modelo presenta limitaciones que contextualizan el alcance de los resultados. El tamaño del *dataset*, si bien suficiente para entrenamiento con *transfer learning*, representa una limitación para alcanzar un mejor desempeño de las arquitecturas de aprendizaje profundo. La necesidad de técnicas intensivas de regularización y aumentación de datos evidencia que un *dataset* más extenso mejoraría el desempeño y robustez del modelo.

Respecto a las categorías de error, el modelo clasifica tres categorías (correcto, errores de codos, errores de rodillas), pero existen otros aspectos técnicos del *press* de hombro no cubiertos, como la trayectoria de barra, activación del *core* abdominal y posición de pies. Adicionalmente, el modelo entrenado es específico para este ejercicio, requiriendo reentrenamiento completo para aplicarse a otros movimientos.

Finalmente, el modelo clasifica la ejecución en una de las tres categorías establecidas, sin proporcionar información específica sobre qué aspecto de la técnica requiere ajuste ni instrucciones de corrección. Para una aplicación práctica en entrenamiento, resultaría valioso identificar el momento temporal específico donde ocurre el error y sugerir correcciones.

Los resultados demuestran que el enfoque propuesto constituye una base técnica viable para el análisis automatizado de técnica en ejercicios, cuyas limitaciones identificadas representan direcciones claras para trabajo futuro.

Capítulo 5

Conclusiones y Trabajo Futuro

Este capítulo sintetiza los logros del proyecto en relación a los objetivos planteados y propone direcciones para trabajo futuro que extiendan las capacidades del modelo desarrollado.

5.1 Cumplimiento de Objetivos y Líneas Futuras

El objetivo general del proyecto consistió en desarrollar un modelo de clasificación de técnica en el ejercicio de *press* de hombro mediante técnicas de visión por computadora y aprendizaje profundo. Este objetivo se alcanzó mediante el desarrollo de un *pipeline* completo que procesa videos de ejercicio y clasifica la técnica de ejecución con precisión de 91.6 %.

Respecto a los objetivos específicos planteados en la Sección 1.1:

1. Conformar un conjunto de videos de entrenamiento útil para el ejercicio seleccionado.

Se cumplió mediante la adopción del *dataset Fitness-AQA* (Parmar y cols., 2022), específicamente el subconjunto de *press* de hombro con 2,135 videos etiquetados en tres categorías (correcto, errores de codos, errores de rodillas). La evaluación del *dataset* reveló que resulta suficiente para entrenar modelos de clasificación cuando se combina con técnicas de *transfer learning* y aumentación de datos, aunque su tamaño limitado requirió estrategias específicas de regularización para evitar *overfitting*.

2. Construir un *pipeline* de extracción y transformación de información biomecánica que convierta videos crudos en datos estructurados compatibles con modelos de clasificación.

Se cumplió mediante el desarrollo de un *pipeline* de procesamiento que integra: (a) extracción de *keypoints* usando *ViTPose* implementado en *Google Colab*, generando archivos *JSON* con 133 puntos anatómicos por *frame*; (b) transformación de datos mediante *notebooks* en Python que reducen a los 17 *keypoints COCO* relevantes, filtran detecciones de baja confianza, y estructuran los datos en formato *pickle* compatible con *PoseConv3D*; (c) algoritmo de corrección de *tracking* que mantiene consistencia de IDs de personas entre *frames* mediante cálculo de distancia euclidiana entre poses, solucionando las fragmentaciones causadas por la detección *frame*-por-*frame* independiente de *ViTPose*.

Este *pipeline* representa una contribución técnica del proyecto, particularmente el algoritmo de corrección de *tracking* que agrega consistencia temporal a la detección de poses sin requerir modelos de *tracking* especializados.

3. Entrenar y optimizar iterativamente modelos de clasificación hasta alcanzar desempeño robusto.

Se cumplió mediante un proceso de desarrollo iterativo documentado que progresó desde un experimento base con *overfitting* severo (50 % de precisión) hasta la configuración final optimizada (91.6 % de precisión, *F1*=0.91). El proceso exploró sistemáticamente estrategias de *transfer learning* desde *NTU60*, técnicas de regularización (suavizado de etiquetas, weight

decay, abandono), aumentación de datos (*Pose Jittering*, *flip*, *crop*), y optimizadores (SGD vs. *AdamW*). Cada iteración se evaluó mediante análisis de curvas de entrenamiento para identificar limitaciones específicas y orientar las optimizaciones subsiguientes.

4. Evaluar el desempeño del modelo desarrollado utilizando métricas estándar y protocolos de validación.

Se cumplió mediante evaluación del modelo en conjunto de validación separado (20 % del *dataset*) utilizando métricas estándar de clasificación: precisión, *F1-score*, y análisis de curvas de pérdida durante entrenamiento. Las métricas alcanzadas (91.6 % de precisión, 0.91 de *F1-score*) demuestran capacidad de generalización satisfactoria a ejemplos no vistos durante entrenamiento. La evaluación final se reporta sobre el conjunto de validación (20 % del *dataset*), que no fue utilizado durante el entrenamiento del modelo. Esta decisión responde al tamaño limitado del *dataset*: maximizar los datos disponibles para entrenamiento priorizó el desempeño del modelo frente a reservar un conjunto de prueba adicional.

A partir de los resultados obtenidos y las limitaciones identificadas en la sección de Discusión, se identifican las siguientes direcciones para extender las capacidades del modelo desarrollado:

Expansión de categorías de error: Ampliar el conjunto de tipos de errores detectables más allá de codos y rodillas, incluyendo aspectos técnicos adicionales del *press* de hombro (trayectoria de barra, activación de core, posición de pies). Esto requeriría expandir el *dataset* con anotaciones de estos errores adicionales y potencialmente explorar arquitecturas de clasificación multi-etiqueta para detectar múltiples errores simultáneos en una misma ejecución.

Localización temporal de errores: Extender el modelo para identificar no solo si existe un error sino en qué *frames* específicos de la secuencia ocurre. Esto permitiría proporcionar *feedback* específico sobre el momento del ejercicio donde la técnica se deteriora, facilitando corrección dirigida.

Ampliación a múltiples ejercicios: Desarrollar modelos para evaluación de técnica en otros ejercicios comunes (sentadillas, peso muerto, press de banca) y explorar estrategias de *transfer learning* con *fine-tuning* entre ejercicios para aprovechar patrones biomecánicos compartidos. Adicionalmente, investigar la viabilidad de modelos generalizados que aprendan representaciones transferibles entre diferentes tipos de movimientos.

Validación con usuarios reales: Implementar estudios de usuario con entrenadores físicos y practicantes para evaluar la utilidad práctica del modelo, recopilar *feedback* sobre falsos positivos/negativos, y comparar la clasificación automatizada con criterio experto humano en condiciones realistas de uso.

Incorporación de feedback correctivo: Desarrollar un módulo que traduzca las predicciones del modelo en instrucciones específicas de corrección, potencialmente utilizando modelos de lenguaje para generar explicaciones comprensibles sobre qué aspecto de la técnica requiere ajuste.

Optimización de eficiencia computacional: Investigar modelos más livianos o técnicas de compresión para permitir inferencia en tiempo real en dispositivos con recursos limitados, apuntando a aplicaciones móviles para usuarios finales.

5.2 Reflexión Final

Este proyecto demostró que la combinación de técnicas modernas de visión por computadora y aprendizaje profundo permite desarrollar modelos de clasificación de técnica en ejercicios que alcanzan un desempeño competitivo utilizando únicamente video RGB estándar. Los resultados validan la viabilidad de soluciones accesibles basadas en código abierto como alternativa a sistemas comerciales especializados.

El proceso de desarrollo iterativo, documentado exhaustivamente en este trabajo, ilustra que el éxito en proyectos de aprendizaje automático depende tanto de la arquitectura de modelos como de la calidad y estructura de los datos. Particularmente, el algoritmo de corrección de *tracking* que apliqué resultó diferencial para permitir que el modelo aprenda patrones temporales coherentes, evidenciando la importancia de preprocesamiento cuidadoso de datos y de intentar entender qué está viendo el modelo cuando entrena.

Este proyecto representa la convergencia de un proceso formativo especializado, extendido durante aproximadamente dos años, a los que se añaden los cinco años previos de conocimiento general en el campo de la Ingeniería. Su punto de partida fue una conversación con el Instituto de Rehabilitación de la UCASAL, donde se planteó la necesidad de aplicar tecnologías de análisis de movimiento a la evaluación de ejercicios. En aquel momento la propuesta era utilizar *OpenPose*; lo que tenía claro era el objetivo: recibir un video de alguien realizando un ejercicio y devolver una evaluación de la técnica. El camino recorrido desde esa idea inicial hasta los resultados de este trabajo implicó incorporar nociones progresivas sobre las complejidades matemáticas de estas tecnologías, los principios detrás de cada técnica y las perspectivas que fundamentan estos enfoques, para entender de a poco cómo aplicarlos.

Los intercambios académicos que realicé en España (Universidad de Cádiz) y Corea del Sur (POSTECH) me aportaron los fundamentos en visión por computadora, aprendizaje profundo y materias relacionadas que resultaron esenciales para abordar los desafíos técnicos del trabajo. En Corea, en particular, me crucé con personas con conocimiento profundo en estas áreas que me ayudaron, aportaron y orientaron cuando me trababa. Quiero agradecer especialmente al equipo con el que trabajé en el curso *Deep Learning* de POSTECH: Jakob, Stamatia, Thao y Darrel, con quienes presenté el estado de avance de esta investigación para el proyecto final de esa materia, con quienes trabajamos y debatimos muchas de las soluciones que finalmente se implementaron, y junto a quienes logramos aprobar el curso.

Cuando empecé a andar este camino tenía apenas conocimientos básicos en IA, tras el *boom* de *ChatGPT* y lo visto en materias como *Sistemas Inteligentes* o *Minería de Datos*. El desarrollo iterativo, desde primeras versiones fallidas hasta resultados satisfactorios, refleja la aplicación práctica de esos conocimientos adquiridos. Y en el proceso descubrí otras culturas, aprendí otros idiomas, conocí amigos que ahora viven en otra parte del mundo y viví situaciones que solo pude haber soñado, lo cual merece ser mencionado como parte del proceso.

El valor de este trabajo, a mi entender, radica en la investigación realizada y documentada: un *pipeline* completo que permite entender cómo funcionaría una solución de extremo a extremo para este tipo de problema, y que puede servir de base para desarrollos más complejos si se cuenta con inversión y un equipo especializado. Fue un trabajo casi unipersonal, sostenido durante dos años sin conocimiento previo en el área; el recurso que lo habilitó fueron las becas que me permitieron formarme en los contextos donde pude adquirir y aplicar los conocimientos que este trabajo requería.

Aunque el sistema presenta limitaciones en su estado actual, el desarrollo generó artefactos reutilizables: *scripts* de preprocesamiento, el algoritmo de corrección de *tracking*, y con-

figuraciones de entrenamiento documentadas que facilitan la replicación y extensión del trabajo. Esta base metodológica puede extenderse para crear herramientas de asistencia en entrenamiento físico, rehabilitación y educación deportiva. La dirección de trabajo futuro propuesta provee un camino hacia sistemas que combinen detección de errores con *feedback* correctivo específico.

Referencias

Cao, Z., Simon, T., Wei, S.-E., y Sheikh, Y. (2017). Realtime multi-person 2D pose estimation using part affinity fields. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 7291–7299).

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. En *International conference on learning representations (iclr)*.

Duan, H., Zhao, Y., Chen, K., Lin, D., y Dai, B. (2022). Revisiting skeleton-based action recognition. En *Proceedings of the ieee/cvf conference on computer vision and pattern recognition (cvpr)* (pp. 2969–2978).

Feichtenhofer, C., Fan, H., Malik, J., y He, K. (2019). SlowFast networks for video recognition. En *Proceedings of the ieee/cvf international conference on computer vision (iccv)* (pp. 6202–6211).

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. Descargado de https://www.deeplearningbook.org/

He, K., Zhang, X., Ren, S., y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 770–778).

Kingma, D. P., y Ba, J. (2015). Adam: A method for stochastic optimization. En *International conference on learning representations (iclr)*.

LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. doi: 10.1038/nature14539

Loshchilov, I., y Hutter, F. (2019). Decoupled weight decay regularization. En *International conference on learning representations (iclr)*.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Parmar, P., Gharat, A., y Rhodin, H. (2022). Domain knowledge-informed self-supervised representations for workout form assessment. En *Proceedings of the european conference on computer vision (eccv)* (pp. 105–123).

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016). You only look once: Unified, real-time object detection. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 779–788).

Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Shahroudy, A., Liu, J., Ng, T.-T., y Wang, G. (2016). NTU RGB+D: A large scale dataset for action recognition. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 1010–1019).

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., y Wojna, Z. (2016). Rethinking the Inception architecture for computer vision. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 2818–2826).

Tran, D., Bourdev, L., Fergus, R., Torresani, L., y Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. En *Proceedings of the ieee international conference on computer vision (iccv)* (pp. 4489–4497).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En *Advances in neural information processing systems* (Vol. 30, pp. 5998–6008).

Windolf, M., Götzen, N., y Morlock, M. (2008). Systematic accuracy and precision analysis of video motion capturing systems—exemplified on the Vicon-460 system. *Journal of*

Biomechanics, *41*(12), 2776–2780.

Xu, Y., Zhang, J., Zhang, Q., y Tao, D. (2022). ViTPose: Simple vision transformer baselines for human pose estimation. En *Advances in neural information processing systems* (Vol. 35, pp. 38571–38584).

Yan, S., Xiong, Y., y Lin, D. (2018). Spatial temporal graph convolutional networks for skeleton-based action recognition. En *Proceedings of the aaai conference on artificial intelligence* (Vol. 32, pp. 7444–7452).

Yosinski, J., Clune, J., Bengio, Y., y Lipson, H. (2014). How transferable are features in deep neural networks? En *Advances in neural information processing systems* (Vol. 27, pp. 3320–3328).