Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia artificial

Optimización de Hiperparámetros OCR con Ray Tune para Documentos Académicos en Español

Trabajo fin de estudio presentado por:	Sergio Jiménez Jiménez
Tipo de trabajo:	Desarrollo Software
Director/a:	Javier Rodrigo Villazón Terrazas
Fecha:	06.10.2025

Resumen

El presente Trabajo Fin de Máster aborda la optimización de sistemas de Reconocimiento Óptico de Caracteres (OCR) basados en inteligencia artificial para documentos en español. El objetivo principal es identificar la configuración óptima de hiperparámetros que maximice la precisión del reconocimiento de texto sin requerir fine-tuning de los modelos base. Se realizó un estudio comparativo de tres soluciones OCR de código abierto: EasyOCR, PaddleOCR (PP-OCRv5) y DocTR, evaluando su rendimiento mediante las métricas estándar CER (Character Error Rate) y WER (Word Error Rate) sobre un corpus de 45 páginas de documentos académicos en español. Tras identificar PaddleOCR como la solución más prometedora, se procedió a una optimización sistemática de hiperparámetros utilizando Ray Tune con el algoritmo de búsqueda Optuna, ejecutando 64 configuraciones diferentes con aceleración GPU (NVIDIA RTX 3060). Los resultados demuestran que la optimización de hiperparámetros logró mejoras significativas: el mejor trial individual alcanzó un CER de 0.79% (precisión del 99.21%), cumpliendo el objetivo de CER < 2%. Al validar la configuración optimizada sobre el dataset completo de 45 páginas, se obtuvo una mejora del 12.8% en CER (de 8.85% a 7.72%). El hallazgo más relevante fue que el parámetro `textline_orientation` (clasificación de orientación de línea de texto) tiene un impacto crítico en el rendimiento. Adicionalmente, se identificó que el umbral de detección (`text_det_thresh`) presenta una correlación negativa moderada (-0.52) con el error. Este trabajo demuestra que la optimización de hiperparámetros es una alternativa viable al fine-tuning, especialmente útil cuando se dispone de modelos preentrenados para el idioma objetivo. La infraestructura dockerizada desarrollada permite reproducir los experimentos y facilita la evaluación sistemática de configuraciones OCR.

Palabras clave: OCR, Reconocimiento Óptico de Caracteres, PaddleOCR, Optimización de Hiperparámetros, Ray Tune, Procesamiento de Documentos, Inteligencia Artificial

Abstract

This Master's Thesis addresses the optimization of Artificial Intelligence-based Optical Character Recognition (OCR) systems for Spanish documents. The main objective is to identify the optimal hyperparameter configuration that maximizes text recognition accuracy without requiring fine-tuning of the base models. A comparative study of three open-source OCR solutions was conducted: EasyOCR, PaddleOCR (PP-OCRv5), and DocTR, evaluating their performance using standard CER (Character Error Rate) and WER (Word Error Rate) metrics on a corpus of 45 pages of academic documents in Spanish. After identifying PaddleOCR as the most promising solution, systematic hyperparameter optimization was performed using Ray Tune with the Optuna search algorithm, executing 64 different configurations with GPU acceleration (NVIDIA RTX 3060). Results demonstrate that hyperparameter optimization achieved significant improvements: the best individual trial reached a CER of 0.79% (99.21% accuracy), meeting the CER < 2% objective. When validating the optimized configuration on the full 45-page dataset, a 12.8% CER improvement was obtained (from 8.85% to 7.72%). The most relevant finding was that the `textline_orientation` parameter (text line orientation classification) has a critical impact on performance. Additionally, the detection threshold (`text_det_thresh`) was found to have a moderate negative correlation (-0.52) with error. This work demonstrates that hyperparameter optimization is a viable alternative to fine-tuning, especially useful when pre-trained models for the target language are available. The dockerized infrastructure developed enables experiment reproducibility and facilitates systematic evaluation of OCR configurations.

Keywords: OCR, Optical Character Recognition, PaddleOCR, Hyperparameter Optimization, Ray Tune, Document Processing, Artificial Intelligence

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

¿Es posible mejorar significativamente un sistema OCR sin reentrenarlo? Esta pregunta, aparentemente simple, encierra un desafío práctico que afecta a investigadores, instituciones educativas y empresas que necesitan digitalizar documentos pero carecen de los recursos para realizar fine-tuning de modelos neuronales. A lo largo de este capítulo se desarrolla la motivación del trabajo, se identifica el problema a resolver y se plantean las preguntas de investigación que guiarán el desarrollo experimental.

1.1. Motivación

El Reconocimiento Óptico de Caracteres (OCR) es una tecnología fundamental en la era de la digitalización documental. Su capacidad para convertir imágenes de texto en datos editables y procesables ha transformado sectores como la administración pública, el ámbito legal, la banca y la educación. Según estimaciones del sector, el mercado global de OCR alcanzó los 13.4 mil millones de dólares en 2023, con proyecciones de crecimiento continuo impulsado por la transformación digital empresarial (Grand View Research, 2023). Sin embargo, a pesar de los avances significativos impulsados por el aprendizaje profundo, la implementación práctica de sistemas OCR de alta precisión sigue presentando desafíos considerables.

1.1.1. El contexto de la digitalización documental

La digitalización de documentos ha pasado de ser una opción a una necesidad estratégica para organizaciones de todos los tamaños. Los beneficios son múltiples: reducción del espacio físico de almacenamiento, facilidad de búsqueda y recuperación, preservación del patrimonio documental, y habilitación de flujos de trabajo automatizados. Sin embargo, la mera conversión de papel a imagen digital no aprovecha plenamente estas ventajas; es necesario extraer el texto contenido en los documentos para permitir su indexación, análisis y procesamiento automatizado.

El OCR actúa como puente entre el mundo físico del documento impreso y el mundo digital del texto procesable. Su precisión determina directamente la calidad de los procesos downstream: un error de reconocimiento en un nombre propio puede invalidar una búsqueda; un dígito mal reconocido en una factura puede causar discrepancias contables; una palabra mal interpretada en un contrato puede alterar su significado legal.

1.1.2. Desafíos específicos del español

El procesamiento de documentos en español presenta particularidades que complican el reconocimiento automático de texto. Los caracteres especiales propios del idioma (la letra ñ, las vocales acentuadas á, é, í, ó, ú, la diéresis ü, y los signos de puntuación invertidos ¿ y ¡) no están presentes en muchos conjuntos de entrenamiento internacionales, lo que puede degradar el rendimiento de modelos preentrenados predominantemente en inglés.

La Tabla 1 resume los principales desafíos lingüísticos del OCR en español:

Tabla 1. ***Desafíos lingüísticos específicos del OCR en español.***

Desafío	Descripción	Impacto en OCR
Caracteres especiales	ñ, á, é, í, ó, ú, ü, ¿, ¡	Confusión con caracteres similares (n/ñ, a/á)
Palabras largas	Español permite compuestos largos	Mayor probabilidad de error por carácter
Abreviaturas	Dr., Sra., Ud., etc.	Puntos internos confunden segmentación
Nombres propios	Tildes en apellidos (García, Martínez)	Bases de datos sin soporte Unicode

Fuente: Elaboración propia.

Además de los aspectos lingüísticos, los documentos académicos y administrativos en español presentan características tipográficas que complican el reconocimiento: variaciones en fuentes entre encabezados, cuerpo y notas al pie; presencia de tablas con bordes y celdas; logotipos institucionales; marcas de agua; y elementos gráficos como firmas o sellos. Estos elementos generan ruido que puede propagarse en aplicaciones downstream como la extracción de entidades nombradas o el análisis semántico.

1.1.3. La brecha entre investigación y práctica

Los modelos OCR basados en redes neuronales profundas, como los empleados en PaddleOCR, EasyOCR o DocTR, ofrecen un rendimiento impresionante en benchmarks

estándar. PaddleOCR, por ejemplo, reporta tasas de precisión superiores al 97% en conjuntos de datos como ICDAR 2015 (Du et al., 2020). No obstante, estos resultados en condiciones controladas no siempre se trasladan a documentos del mundo real.

La adaptación de modelos preentrenados a dominios específicos típicamente requiere fine-tuning con datos etiquetados del dominio objetivo y recursos computacionales significativos. El fine-tuning de un modelo de reconocimiento de texto puede requerir decenas de miles de imágenes etiquetadas y días de entrenamiento en GPUs de alta capacidad. Esta barrera técnica y económica excluye a muchos investigadores y organizaciones de beneficiarse plenamente de estas tecnologías.

La Tabla 2 ilustra los requisitos típicos para diferentes estrategias de mejora de OCR:

Tabla 2. ***Comparación de estrategias de mejora de modelos OCR.***

Estrategia	Datos requeridos	Hardware	Tiempo	Expertise
Fine-tuning completo	>10,000 imágenes etiquetadas	GPU (≥16GB VRAM)	Días-Semanas	Alto
Fine-tuning parcial	>1,000 imágenes etiquetadas	GPU (≥8GB VRAM)	Horas-Días	Medio-Alto
Transfer learning	>500 imágenes etiquetadas	GPU (≥8GB VRAM)	Horas	Medio
Optimización de hiperparámetros	**<100 imágenes de validación**	**CPU suficiente**	**Horas**	**Bajo-Medio**

Fuente: Elaboración propia.

1.1.4. La oportunidad: optimización sin fine-tuning

La presente investigación surge de una necesidad práctica: optimizar un sistema OCR para documentos académicos en español sin disponer de recursos GPU para realizar fine-tuning. Esta restricción, lejos de ser una limitación excepcional, representa la realidad de muchos entornos académicos y empresariales donde el acceso a infraestructura de cómputo avanzada es limitado.

La hipótesis central de este trabajo es que los modelos OCR preentrenados contienen capacidades latentes que pueden activarse mediante la configuración adecuada de sus hiperparámetros de inferencia. Parámetros como los umbrales de detección de texto, las opciones de preprocesamiento de imagen, y los filtros de confianza de reconocimiento pueden tener un impacto significativo en el rendimiento final, y su optimización sistemática puede aproximarse a los beneficios del fine-tuning sin sus costes asociados.

Esta oportunidad se ve reforzada por la disponibilidad de frameworks modernos de optimización de hiperparámetros como Ray Tune (Liaw et al., 2018) y algoritmos de búsqueda eficientes como Optuna (Akiba et al., 2019), que permiten explorar espacios de configuración de manera sistemática y eficiente.

1.2. Planteamiento del trabajo

1.2.1. Formulación del problema

Las observaciones anteriores conducen a formular el problema central de este trabajo:

¿Es posible mejorar significativamente el rendimiento de modelos OCR preentrenados para documentos en español mediante la optimización sistemática de hiperparámetros, sin requerir fine-tuning ni recursos GPU?

Este planteamiento parte de una observación fundamental: los sistemas OCR modernos exponen múltiples parámetros configurables que afectan su comportamiento durante la inferencia. Estos parámetros incluyen umbrales de detección, opciones de preprocesamiento, y filtros de calidad. En la práctica habitual, estos parámetros se dejan en sus valores por defecto, asumiendo que fueron optimizados por los desarrolladores del modelo. Sin embargo, los valores por defecto representan compromisos generales que pueden no ser óptimos para dominios específicos.

1.2.2. Preguntas de investigación

Este planteamiento se descompone en las siguientes cuestiones específicas:

PI1. Selección de modelo base: ¿Cuál de las soluciones OCR de código abierto disponibles (EasyOCR, PaddleOCR, DocTR) ofrece el mejor rendimiento base para documentos en español?

Esta pregunta es fundamental porque la elección del modelo base determinará el punto de partida para la optimización. Un modelo con mejor rendimiento inicial puede ofrecer mayor margen de mejora o, alternativamente, estar ya cerca de su límite de optimización.

PI2. Impacto de hiperparámetros: ¿Qué hiperparámetros del pipeline OCR tienen mayor influencia en las métricas de error (CER, WER)?

Identificar los parámetros más influyentes permite focalizar el esfuerzo de optimización y proporciona insights sobre el funcionamiento interno del sistema. Parámetros con alta correlación con las métricas de error son candidatos prioritarios para ajuste.

PI3. Optimización automatizada: ¿Puede un proceso de búsqueda automatizada de hiperparámetros (mediante Ray Tune/Optuna) encontrar configuraciones que superen significativamente los valores por defecto?

Esta pregunta evalúa la viabilidad práctica de la metodología propuesta. "Significativamente" se define operacionalmente como una reducción del CER de al menos 50% respecto al baseline, un umbral que representaría una mejora sustancial en la calidad del texto reconocido.

PI4. Viabilidad práctica: ¿Son los tiempos de inferencia y los recursos requeridos compatibles con un despliegue en entornos con recursos limitados?

Una solución técnicamente superior pero impracticable tiene valor limitado. Esta pregunta ancla la investigación en consideraciones del mundo real.

1.2.3. Alcance y delimitación

Este trabajo se centra específicamente en:

Tabla 3. ***Delimitación del alcance del trabajo.***

Aspecto	Dentro del alcance	Fuera del alcance
Tipo de documento	Documentos académicos digitales (PDF)	Documentos escaneados, manuscritos
Idioma	Español	Otros idiomas

Modelos	EasyOCR, PaddleOCR, DocTR	Soluciones comerciales (Google Cloud Vision, AWS Textract)
Método de mejora	Optimización de hiperparámetros	Fine-tuning, aumento de datos
Hardware	Ejecución en CPU	Aceleración GPU

Fuente: Elaboración propia.

1.2.4. Relevancia y beneficiarios

La relevancia de este problema radica en su aplicabilidad inmediata. Una metodología reproducible para optimizar OCR sin fine-tuning beneficiaría a múltiples grupos:

Investigadores académicos: Quienes procesan grandes volúmenes de documentos para análisis de contenido, revisiones sistemáticas de literatura, o estudios bibliométricos. Un OCR más preciso reduce el tiempo de corrección manual y mejora la calidad de los análisis downstream.

Instituciones educativas: Universidades y centros de investigación que digitalizan archivos históricos, actas administrativas, o materiales docentes. La preservación del patrimonio documental requiere transcripciones precisas.

Pequeñas y medianas empresas: Organizaciones que automatizan flujos documentales (facturas, contratos, correspondencia) sin presupuesto para soluciones enterprise o infraestructura GPU.

Desarrolladores de software: Quienes integran OCR en aplicaciones con restricciones de recursos, como dispositivos móviles o servidores compartidos, y necesitan maximizar el rendimiento sin costes adicionales de hardware.

1.3. Estructura del trabajo

El documento sigue una estructura que refleja el proceso investigador. Tras esta introducción, el **Capítulo 2** sitúa el trabajo en su contexto técnico, revisando las tecnologías OCR basadas en aprendizaje profundo —desde las arquitecturas de detección hasta los modelos de reconocimiento— y los trabajos previos en optimización de estos sistemas.

El **Capítulo 3** traduce las preguntas de investigación en objetivos concretos siguiendo la metodología SMART, y describe con detalle el enfoque experimental: preparación del dataset, métricas de evaluación y configuración del proceso de optimización con Ray Tune y Optuna.

El núcleo del trabajo se desarrolla en el **Capítulo 4**, que presenta el estudio comparativo y la optimización de hiperparámetros estructurados en tres fases: planteamiento de la comparativa con evaluación de EasyOCR, PaddleOCR y DocTR; desarrollo de la optimización mediante 64 trials con Ray Tune; y análisis crítico de los resultados obtenidos.

Finalmente, el **Capítulo 5** sintetiza las contribuciones, evalúa el grado de cumplimiento de los objetivos y propone líneas de trabajo futuro. Los **Anexos** proporcionan acceso al repositorio de código fuente y datos, así como tablas detalladas de resultados experimentales.

2. Contexto y estado del arte

Para comprender el alcance y las decisiones tomadas en este trabajo, es necesario situarlo en su contexto tecnológico. El Reconocimiento Óptico de Caracteres ha recorrido un largo camino desde los primeros sistemas de plantillas de los años 50 hasta las sofisticadas arquitecturas de aprendizaje profundo actuales. A lo largo de este capítulo se revisan los fundamentos técnicos del OCR moderno, se analizan las principales soluciones de código abierto y se identifican los vacíos en la literatura que motivan la contribución de este trabajo.

2.1. Contexto del problema

2.1.1. Definición y Evolución Histórica del OCR

El Reconocimiento Óptico de Caracteres (OCR) es el proceso de conversión de imágenes de texto manuscrito, mecanografiado o impreso en texto codificado digitalmente. Esta tecnología permite la digitalización masiva de documentos, facilitando su búsqueda, edición y almacenamiento electrónico. La tecnología OCR ha evolucionado significativamente desde sus orígenes en la década de 1950, atravesando cuatro generaciones claramente diferenciadas:

2.1.1.1. Primera Generación (1950-1970): Sistemas basados en plantillas

Los primeros sistemas OCR surgieron en la década de 1950 con el objetivo de automatizar la lectura de documentos bancarios y postales. Estos sistemas utilizaban técnicas de correspondencia de plantillas (*template matching*), donde cada carácter de entrada se comparaba píxel a píxel con un conjunto predefinido de plantillas (Mori et al., 1992).

Las principales limitaciones de esta generación incluían:

- Dependencia de fuentes tipográficas específicas (OCR-A, OCR-B)
- Incapacidad para manejar variaciones en tamaño, rotación o estilo
- Alto coste computacional para la época
- Sensibilidad extrema al ruido y degradación de la imagen

A pesar de sus limitaciones, estos sistemas sentaron las bases para el desarrollo posterior del campo y demostraron la viabilidad comercial del reconocimiento automático de texto.

2.1.1.2. Segunda Generación (1970-1990): Extracción de características

La segunda generación introdujo técnicas más sofisticadas basadas en la extracción de características geométricas y estructurales de los caracteres. En lugar de comparar imágenes completas, estos sistemas extraían propiedades como:

- Número y posición de trazos
- Proporciones geométricas (altura, anchura, relación de aspecto)
- Momentos estadísticos de la distribución de píxeles
- Características topológicas (bucles, intersecciones, terminaciones)

Los clasificadores estadísticos, como el análisis discriminante lineal y los k-vecinos más cercanos (k-NN), se utilizaban para asignar cada vector de características a una clase de carácter (Trier et al., 1996). Esta aproximación permitió mayor robustez frente a variaciones tipográficas, aunque seguía requiriendo un diseño manual cuidadoso de las características a extraer.

2.1.1.3. Tercera Generación (1990-2010): Redes neuronales y modelos probabilísticos

La tercera generación marcó la introducción de técnicas de aprendizaje automático más avanzadas. Los Modelos Ocultos de Markov (HMM) se convirtieron en el estándar para el reconocimiento de secuencias de caracteres, especialmente en el reconocimiento de escritura manuscrita (Plamondon & Srihari, 2000).

Las Redes Neuronales Artificiales (ANN) también ganaron popularidad en esta época, con arquitecturas como el Perceptrón Multicapa (MLP) demostrando capacidades superiores de generalización. El trabajo seminal de LeCun et al. (1998) con las redes convolucionales (CNN)

para el reconocimiento de dígitos manuscritos (dataset MNIST) estableció los fundamentos para la siguiente revolución.

Las características de esta generación incluían:

- Aprendizaje automático de características discriminativas
- Modelado probabilístico de secuencias de caracteres
- Mayor robustez frente a ruido y degradación
- Capacidad de incorporar conocimiento lingüístico mediante modelos de lenguaje

2.1.1.4. Cuarta Generación (2010-presente): Aprendizaje profundo

La cuarta y actual generación está dominada por arquitecturas de aprendizaje profundo que han superado ampliamente el rendimiento de los métodos tradicionales. Los avances clave incluyen:

Redes Convolucionales Profundas (Deep CNNs): Arquitecturas como VGGNet, ResNet e Inception permiten la extracción automática de características jerárquicas a múltiples escalas, eliminando la necesidad de diseño manual de características (Krizhevsky et al., 2012).

Redes Recurrentes (RNN/LSTM): Las redes Long Short-Term Memory (LSTM) permiten modelar dependencias a largo plazo en secuencias de caracteres, siendo fundamentales para el reconocimiento de texto de longitud variable (Graves et al., 2009).

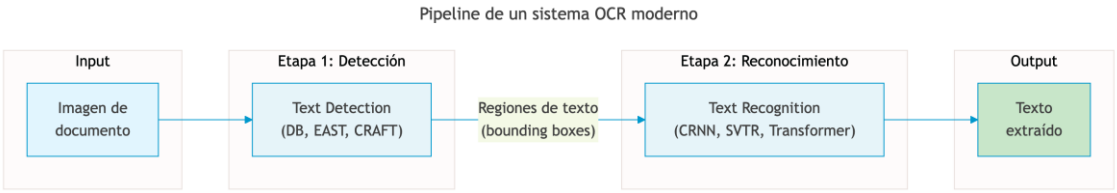
Mecanismos de Atención y Transformers: La arquitectura Transformer (Vaswani et al., 2017) y sus variantes han revolucionado el procesamiento de secuencias, permitiendo capturar relaciones globales sin las limitaciones de las RNN. Modelos como TrOCR (Li et al., 2023) representan el estado del arte actual.

Connectionist Temporal Classification (CTC): La función de pérdida CTC (Graves et al., 2006) permite entrenar modelos de reconocimiento de secuencias sin necesidad de alineamiento carácter por carácter, simplificando enormemente el proceso de entrenamiento.

2.1.2. Pipeline Moderno de OCR

Los sistemas OCR modernos siguen típicamente un pipeline de dos etapas principales, precedidas opcionalmente por una fase de preprocesamiento:

Figura 1. ***Pipeline de un sistema OCR moderno***



Fuente: Elaboración propia.

2.1.2.1. Etapa de Preprocesamiento

Antes de la detección, muchos sistemas aplican técnicas de preprocesamiento para mejorar la calidad de la imagen de entrada:

- **Binarización**: Conversión a imagen binaria (blanco/negro) mediante técnicas como Otsu o Sauvola
- **Corrección de inclinación (deskewing)**: Alineamiento horizontal del texto
- **Eliminación de ruido**: Filtros morfológicos y de suavizado
- **Normalización de contraste**: Mejora de la legibilidad mediante ecualización de histograma

2.1.2.2. Etapa 1: Detección de Texto (Text Detection)

La detección de texto tiene como objetivo localizar todas las regiones de una imagen que contienen texto. Esta tarea es particularmente desafiante debido a la variabilidad en:

- Tamaño y orientación del texto
- Fondos complejos y oclusiones parciales
- Texto curvo o deformado
- Múltiples idiomas y scripts en una misma imagen

Las arquitecturas más utilizadas para detección de texto incluyen:

EAST (Efficient and Accurate Scene Text Detector): Propuesto por Zhou et al. (2017), EAST es un detector de una sola etapa que predice directamente cuadriláteros rotados o polígonos que encierran el texto. Su arquitectura FCN (Fully Convolutional Network) permite procesamiento eficiente de imágenes de alta resolución.

CRAFT (Character Region Awareness for Text Detection): Desarrollado por Baek et al. (2019), CRAFT detecta regiones de caracteres individuales y las agrupa en palabras mediante el

análisis de mapas de afinidad. Esta aproximación bottom-up es especialmente efectiva para texto con espaciado irregular.

DB (Differentiable Binarization): Propuesto por Liao et al. (2020), DB introduce una operación de binarización diferenciable que permite entrenar end-to-end un detector de texto basado en segmentación. Esta arquitectura es la utilizada por PaddleOCR y destaca por su velocidad y precisión.

Tabla 4. ***Comparativa de arquitecturas de detección de texto.***

Arquitectura	Tipo	Salida	Fortalezas	Limitaciones
EAST	Single-shot	Cuadriláteros rotados	Rápido, simple	Dificultad con texto curvo
CRAFT	Bottom-up	Polígonos de palabra	Robusto a espaciado	Mayor coste computacional
DB	Segmentación	Polígonos arbitrarios	Rápido, preciso	Sensible a parámetros

Fuente: Elaboración propia.

2.1.2.3. Etapa 2: Reconocimiento de Texto (Text Recognition)

Una vez detectadas las regiones de texto, la etapa de reconocimiento transcribe el contenido visual a texto digital. Las arquitecturas predominantes son:

CRNN (Convolutional Recurrent Neural Network): Propuesta por Shi et al. (2016), CRNN combina una CNN para extracción de características visuales con una RNN bidireccional (típicamente LSTM) para modelado de secuencias, entrenada con pérdida CTC. Esta arquitectura estableció el paradigma encoder-decoder que domina el campo.

La arquitectura CRNN consta de tres componentes:

1. **Capas convolucionales**: Extraen características visuales de la imagen de entrada
2. **Capas recurrentes**: Modelan las dependencias secuenciales entre características
3. **Capa de transcripción**: Convierte las predicciones de la RNN en secuencias de caracteres mediante CTC

SVTR (Scene-Text Visual Transformer Recognition): Desarrollado por Du et al. (2022), SVTR aplica la arquitectura Transformer al reconocimiento de texto, utilizando parches de imagen como tokens de entrada. Esta aproximación elimina la necesidad de RNN y permite capturar dependencias globales de manera más eficiente.

Arquitecturas con Atención: Los modelos encoder-decoder con mecanismos de atención (Bahdanau et al., 2015) permiten al decodificador "enfocarse" en diferentes partes de la imagen mientras genera cada carácter. Esto es especialmente útil para texto largo o con layouts complejos.

TrOCR (Transformer-based OCR): Propuesto por Li et al. (2023), TrOCR utiliza un Vision Transformer (ViT) como encoder y un Transformer de lenguaje como decoder, logrando resultados estado del arte en múltiples benchmarks.

Tabla 5. ***Comparativa de arquitecturas de reconocimiento de texto.***

Arquitectura	Encoder	Decoder	Pérdida	Características
CRNN	CNN	BiLSTM	CTC	Rápido, robusto
SVTR	ViT	Linear	CTC	Sin recurrencia
Attention-based	CNN	LSTM+Attn	Cross-entropy	Flexible longitud
TrOCR	ViT	Transformer	Cross-entropy	Estado del arte

Fuente: Elaboración propia.

2.1.3. Métricas de Evaluación

La evaluación rigurosa de sistemas OCR requiere métricas estandarizadas que permitan comparaciones objetivas. Las métricas fundamentales se basan en la distancia de edición de Levenshtein.

2.1.3.1. Distancia de Levenshtein

La distancia de Levenshtein (Levenshtein, 1966) entre dos cadenas es el número mínimo de operaciones de edición (inserción, eliminación, sustitución) necesarias para transformar una cadena en otra. Formalmente, para dos cadenas *a* y *b*:

$$d(a, b) = min(\text{inserciones} + \text{eliminaciones} + \text{sustituciones})$$

Esta métrica es fundamental para calcular tanto CER como WER.

2.1.3.2. Character Error Rate (CER)

El CER mide el error a nivel de carácter y se calcula como:

$$CER = \frac{S + D + I}{N}$$

Donde:

- S = número de sustituciones de caracteres
- D = número de eliminaciones de caracteres
- I = número de inserciones de caracteres
- N = número total de caracteres en el texto de referencia

Un CER del 1% indica que, en promedio, 1 de cada 100 caracteres contiene un error. Para aplicaciones críticas como:

- **Documentos financieros**: Se requiere CER < 0.1%
- **Documentos médicos**: Se requiere CER < 0.5%
- **Documentos académicos**: CER < 2% es aceptable
- **Búsqueda y archivo**: CER < 5% puede ser suficiente

2.1.3.3. Word Error Rate (WER)

El WER mide el error a nivel de palabra, utilizando la misma fórmula pero considerando palabras como unidades:

$$WER = \frac{S_w + D_w + I_w}{N_w}$$

El WER es generalmente mayor que el CER, ya que un solo error de carácter puede invalidar una palabra completa. La relación típica es WER ≈ 2-3 × CER para texto en español.

2.1.3.4. Otras Métricas Complementarias

Precision y Recall a nivel de palabra: Útiles cuando se evalúa la capacidad del sistema para detectar palabras específicas.

Bag-of-Words Accuracy: Mide la proporción de palabras correctamente reconocidas independientemente de su orden.

BLEU Score: Adaptado de traducción automática, mide la similitud entre el texto predicho y la referencia considerando n-gramas.

2.1.4. Particularidades del OCR para el Idioma Español

El español, como lengua romance, presenta características específicas que impactan el rendimiento de los sistemas OCR:

2.1.4.1. Características Ortográficas

Caracteres especiales: El español incluye caracteres no presentes en el alfabeto inglés básico:

- La letra eñe (ñ, Ñ)
- Vocales acentuadas (á, é, í, ó, ú, Á, É, Í, Ó, Ú)
- Diéresis sobre u (ü, Ü)
- Signos de puntuación invertidos (¿, ¡)

Estos caracteres requieren que los modelos OCR incluyan dichos símbolos en su vocabulario de salida y que el entrenamiento incluya suficientes ejemplos de cada uno.

Diacríticos y acentos: Los acentos gráficos del español son elementos pequeños que pueden confundirse fácilmente con ruido, artefactos de imagen o signos de puntuación. La distinción entre vocales acentuadas y no acentuadas es crucial para el significado (e.g., "él" vs "el", "más" vs "mas").

2.1.4.2. Características Lingüísticas

Longitud de palabras: Las palabras en español tienden a ser más largas que en inglés debido a la morfología flexiva rica (conjugaciones verbales, géneros, plurales). Esto puede aumentar la probabilidad de error acumulativo.

Vocabulario: El español tiene un vocabulario amplio con muchas variantes morfológicas de cada raíz. Los modelos de lenguaje utilizados para post-corrección deben contemplar esta diversidad.

2.1.4.3. Recursos y Datasets

Los recursos disponibles para OCR en español son significativamente menores que para inglés o chino:

- Menor cantidad de datasets etiquetados de gran escala
- Menos modelos preentrenados específicos para español
- Documentación y tutoriales predominantemente en inglés

Esta escasez de recursos específicos para español motiva la necesidad de técnicas de adaptación como la optimización de hiperparámetros explorada en este trabajo.

2.2. Estado del arte

2.2.1. Soluciones OCR de Código Abierto

En los últimos años han surgido varias soluciones OCR de código abierto que democratizan el acceso a esta tecnología. A continuación se analizan en detalle las tres principales alternativas evaluadas en este trabajo.

2.2.1.1. EasyOCR

EasyOCR es una biblioteca de OCR desarrollada por Jaided AI (2020) con el objetivo de proporcionar una solución de fácil uso que soporte múltiples idiomas. Actualmente soporta más de 80 idiomas, incluyendo español.

Arquitectura técnica:

- **Detector**: CRAFT (Character Region Awareness for Text Detection)
- **Reconocedor**: CRNN con backbone ResNet/VGG + BiLSTM + CTC
- **Modelos preentrenados**: Disponibles para descarga automática

Características principales:

- API simple de una línea para casos de uso básicos
- Soporte para GPU (CUDA) y CPU
- Reconocimiento de múltiples idiomas en una misma imagen
- Bajo consumo de memoria comparado con otras soluciones

Limitaciones identificadas:

- Opciones de configuración limitadas (pocos hiperparámetros ajustables)
- Menor precisión en documentos con layouts complejos

- Actualizaciones menos frecuentes que otras alternativas
- Documentación menos exhaustiva

Caso de uso ideal: Prototipado rápido, aplicaciones con restricciones de memoria, proyectos que requieren soporte multilingüe inmediato.

2.2.1.2. PaddleOCR

PaddleOCR es el sistema OCR desarrollado por Baidu como parte del ecosistema PaddlePaddle (2024). Representa una de las soluciones más completas y activamente mantenidas en el ecosistema de código abierto. La versión PP-OCRv5, utilizada en este trabajo, incorpora los últimos avances en el campo.

Arquitectura técnica:

El pipeline de PaddleOCR consta de tres módulos principales:

1. **Detector de texto (DB - Differentiable Binarization)**:

- Backbone: ResNet18/ResNet50 - Neck: FPN (Feature Pyramid Network) - Head: Segmentación con binarización diferenciable - Salida: Polígonos que encierran regiones de texto

1. **Clasificador de orientación**:

- Determina si el texto está rotado 0° o 180° - Permite corrección automática de texto invertido - Opcional pero recomendado para documentos escaneados

1. **Reconocedor de texto (SVTR)**:

- Encoder: Vision Transformer modificado - Decoder: CTC o Attention-based - Vocabulario: Configurable por idioma

Hiperparámetros configurables:

PaddleOCR expone numerosos hiperparámetros que permiten ajustar el comportamiento del sistema. Los más relevantes para este trabajo son:

Tabla 6. ***Hiperparámetros de detección de PaddleOCR.***

Parámetro	Descripción	Rango	Defecto

`text_det_thresh`	Umbral de probabilidad para píxeles de texto	[0.0, 1.0]	0.3
`text_det_box_thresh`	Umbral de confianza para cajas detectadas	[0.0, 1.0]	0.6
`text_det_unclip_ratio`	Factor de expansión de cajas detectadas	[0.0, 3.0]	1.5
`text_det_limit_side_len`	Tamaño máximo del lado de imagen	[320, 2560]	960

Fuente: Elaboración propia.

Tabla 7. ***Hiperparámetros de reconocimiento de PaddleOCR.***

Parámetro	Descripción	Rango	Defecto
`text_rec_score_thresh`	Umbral de confianza para resultados	[0.0, 1.0]	0.5
`use_textline_orientation`	Activar clasificación de orientación de línea	{True, False}	False
`rec_batch_size`	Tamaño de batch para reconocimiento	[1, 64]	6

Fuente: Elaboración propia.

Tabla 8. ***Hiperparámetros de preprocesamiento de PaddleOCR.***

Parámetro	Descripción	Impacto
`use_doc_orientation_classify`	Clasificación de orientación del documento	Alto para documentos escaneados
`use_doc_unwarping`	Corrección de deformación/curvatura	Alto para fotos de documentos
`use_angle_cls`	Clasificador de ángulo 0°/180°	Medio para documentos rotados

Fuente: Elaboración propia.

Fortalezas de PaddleOCR:

- Alta precisión en múltiples benchmarks
- Pipeline altamente configurable
- Modelos optimizados para servidor (mayor precisión) y móvil (mayor velocidad)
- Documentación exhaustiva (aunque principalmente en chino)
- Comunidad activa y actualizaciones frecuentes
- Soporte para entrenamiento personalizado (fine-tuning)

Limitaciones:

- Dependencia del framework PaddlePaddle (menos popular que PyTorch/TensorFlow)
- Curva de aprendizaje más pronunciada
- Documentación en inglés menos completa que en chino

2.2.1.3. DocTR

DocTR (Document Text Recognition) es una biblioteca desarrollada por Mindee (2021), empresa especializada en procesamiento inteligente de documentos. Está orientada a la comunidad de investigación y ofrece una API limpia basada en TensorFlow/PyTorch.

Arquitectura técnica:

- **Detectores disponibles**: DB (db_resnet50), LinkNet (linknet_resnet18)
- **Reconocedores disponibles**: CRNN (crnn_vgg16_bn), SAR (sar_resnet31), ViTSTR (vitstr_small)
- **Framework**: TensorFlow 2.x o PyTorch

Características principales:

- API Pythonic bien diseñada
- Salida estructurada con información de confianza y geometría
- Integración nativa con Hugging Face Hub
- Documentación orientada a investigación

Limitaciones identificadas:

- Menor rendimiento en español comparado con PaddleOCR según pruebas preliminares

- Comunidad más pequeña
- Menos opciones de modelos preentrenados para idiomas no ingleses

2.2.1.4. Comparativa Detallada de Soluciones

Tabla 9. ***Comparativa técnica de soluciones OCR de código abierto.***

Aspecto	EasyOCR	PaddleOCR	DocTR
Framework	PyTorch	PaddlePaddle	TF/PyTorch
Detector	CRAFT	DB	DB/LinkNet
Reconocedor	CRNN	SVTR/CRNN	CRNN/SAR/ViTSTR
Idiomas	80+	80+	9
Configurabilidad	Baja	Alta	Media
Documentación	Media	Alta (CN)	Alta (EN)
Actividad	Media	Alta	Media
Licencia	Apache 2.0	Apache 2.0	Apache 2.0

Fuente: Elaboración propia.

Tabla 10. ***Comparativa de facilidad de uso.***

Aspecto	EasyOCR	PaddleOCR	DocTR
Instalación	`pip install`	`pip install`	`pip install`
Líneas para OCR básico	3	5	6
GPU requerida	Opcional	Opcional	Opcional
Memoria mínima	2 GB	4 GB	4 GB

Fuente: Elaboración propia.

2.2.2. Optimización de Hiperparámetros

2.2.2.1. Fundamentos Teóricos

La optimización de hiperparámetros (HPO, *Hyperparameter Optimization*) es el proceso de encontrar la configuración óptima de los parámetros que controlan el proceso de aprendizaje o inferencia de un modelo, pero que no se aprenden directamente de los datos (Feurer & Hutter, 2019).

A diferencia de los parámetros del modelo (como los pesos de una red neuronal), los hiperparámetros se establecen antes del entrenamiento e incluyen:

- Tasa de aprendizaje, tamaño de batch, número de épocas
- Arquitectura del modelo (número de capas, unidades por capa)
- Parámetros de regularización (dropout, weight decay)
- **Umbrales de decisión en tiempo de inferencia** (relevante para este trabajo)

El problema de HPO puede formalizarse como:

$$\lambda^* = argmin_{\lambda \in \Lambda} \mathcal{L}(M_\lambda, D_{val})$$

Donde:

- λ es un vector de hiperparámetros
- Λ es el espacio de búsqueda
- M_λ es el modelo configurado con λ
- $\mathcal{L}$ es la función de pérdida
- D_{val} es el conjunto de validación

2.2.2.2. Métodos de Optimización

Grid Search (Búsqueda en rejilla):

El método más simple consiste en evaluar todas las combinaciones posibles de valores discretizados de los hiperparámetros. Para $k$hiperparámetros con $n$valores cada uno, requiere n^kevaluaciones.

Ventajas:

- Exhaustivo y reproducible
- Fácil de paralelizar
- Garantiza encontrar el óptimo dentro de la rejilla

Desventajas:

- Coste exponencial con el número de hiperparámetros
- Ineficiente si algunos hiperparámetros son más importantes que otros
- No aprovecha información de evaluaciones previas

Random Search (Búsqueda aleatoria):

Propuesto por Bergstra & Bengio (2012), Random Search muestrea configuraciones aleatoriamente del espacio de búsqueda. Sorprendentemente, supera a Grid Search en muchos escenarios prácticos.

La intuición es que, cuando solo algunos hiperparámetros son importantes, Random Search explora más valores de estos parámetros críticos mientras Grid Search desperdicia evaluaciones variando parámetros irrelevantes.

Optimización Bayesiana:

La optimización bayesiana modela la función objetivo mediante un modelo probabilístico sustituto (*surrogate model*) y utiliza una función de adquisición para decidir qué configuración evaluar a continuación (Bergstra et al., 2011).

El proceso iterativo es:

1. Ajustar el modelo sustituto a las observaciones actuales
2. Optimizar la función de adquisición para seleccionar el siguiente punto
3. Evaluar la función objetivo en el punto seleccionado
4. Actualizar las observaciones y repetir

Los modelos sustitutos más comunes son:

- **Procesos Gaussianos (GP)**: Proporcionan incertidumbre bien calibrada pero escalan pobremente
- **Random Forests**: Manejan bien espacios de alta dimensión y variables categóricas
- **Tree-structured Parzen Estimator (TPE)**: Modela densidades en lugar de la función objetivo

2.2.2.3. Tree-structured Parzen Estimator (TPE)

TPE, propuesto por Bergstra et al. (2011) e implementado en Optuna, es particularmente efectivo para HPO. En lugar de modelar $p(y|\lambda)$directamente, TPE modela:

$$p(\lambda|y) = \begin{cases} l(\lambda) & \text{si } y < y^* \\ g(\lambda) & \text{si } y \geq y^* \end{cases}$$

Donde y^*es un umbral (típicamente el percentil 15-25 de las observaciones), $l(\lambda)$es la densidad de hiperparámetros con buen rendimiento, y $g(\lambda)$es la densidad de hiperparámetros con mal rendimiento.

La función de adquisición Expected Improvement se aproxima como:

$$EI(\lambda) \propto \frac{l(\lambda)}{g(\lambda)}$$

Configuraciones con alta probabilidad bajo ly baja probabilidad bajo $g$tienen mayor Expected Improvement.

Ventajas de TPE:

- Maneja naturalmente espacios condicionales (hiperparámetros que dependen de otros)
- Eficiente para espacios de alta dimensión
- No requiere derivadas de la función objetivo
- Implementación eficiente en Optuna

2.2.2.4. Ray Tune

Ray Tune (Liaw et al., 2018) es un framework de optimización de hiperparámetros escalable construido sobre Ray, un sistema de computación distribuida. Sus características principales incluyen:

Escalabilidad:

- Ejecución paralela de múltiples trials
- Distribución automática en clusters
- Soporte para recursos heterogéneos (CPU/GPU)

Flexibilidad:

- Integración con múltiples algoritmos de búsqueda (Optuna, HyperOpt, Ax, etc.)
- Schedulers avanzados (ASHA, PBT, BOHB)
- Checkpointing y recuperación de fallos

Early Stopping:

- ASHA (Asynchronous Successive Halving Algorithm): Termina trials poco prometedores

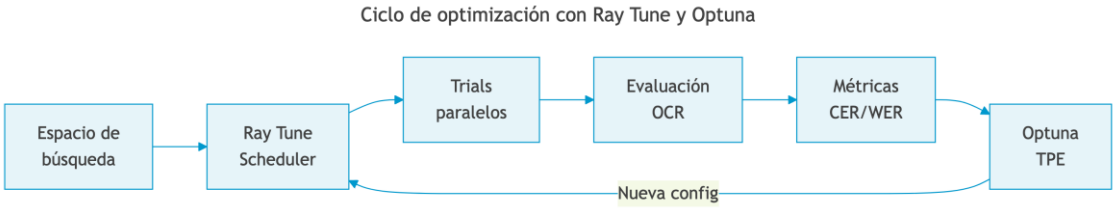
- PBT (Population-Based Training): Evoluciona hiperparámetros durante el entrenamiento

Integración con Optuna:

La combinación de Ray Tune con OptunaSearch permite:

1. Utilizar TPE como algoritmo de búsqueda
2. Paralelizar la evaluación de trials
3. Beneficiarse de la infraestructura de Ray para distribución
4. Acceder a las visualizaciones de Optuna

Figura 2. ***Ciclo de optimización con Ray Tune y Optuna***



Fuente: Elaboración propia.

2.2.2.5. HPO en Sistemas OCR

La aplicación de HPO a sistemas OCR ha sido explorada en varios contextos:

Optimización de preprocesamiento:

Liang et al. (2005) propusieron optimizar parámetros de binarización adaptativa para mejorar el OCR de documentos degradados. Los parámetros optimizados incluían tamaño de ventana, factor de corrección y umbral local.

Optimización de arquitectura:

Breuel (2013) exploró la selección automática de arquitecturas de red para reconocimiento de texto manuscrito, optimizando número de capas, unidades y tipo de activación.

Optimización de post-procesamiento:

Schulz & Kuhn (2017) optimizaron parámetros de modelos de lenguaje para corrección de errores OCR, incluyendo pesos de interpolación entre modelos de caracteres y palabras.

Vacío en la literatura:

A pesar de estos trabajos, existe un vacío significativo respecto a la optimización sistemática de hiperparámetros de inferencia en pipelines OCR modernos como PaddleOCR. La mayoría de trabajos se centran en:

- Entrenamiento de modelos (fine-tuning)
- Preprocesamiento de imagen
- Post-procesamiento lingüístico

La optimización de umbrales de detección y reconocimiento en tiempo de inferencia ha recibido poca atención, especialmente para idiomas diferentes del inglés y chino.

2.2.3. Datasets y Benchmarks para Español

2.2.3.1. Datasets Públicos

Los principales recursos para evaluación de OCR en español incluyen:

FUNSD-ES: Versión en español del Form Understanding in Noisy Scanned Documents dataset. Contiene formularios escaneados con anotaciones de texto y estructura.

MLT (ICDAR Multi-Language Text): Dataset multilingüe de las competiciones ICDAR que incluye muestras en español. Las ediciones 2017 y 2019 contienen texto en escenas naturales.

XFUND: Dataset de comprensión de formularios en múltiples idiomas, incluyendo español, con anotaciones de entidades y relaciones.

Tabla 11. ***Datasets públicos con contenido en español.***

Dataset	Tipo	Idiomas	Tamaño	Uso principal
FUNSD-ES	Formularios	ES	~200 docs	Document understanding
MLT 2019	Escenas	Multi (incl. ES)	10K imgs	Text detection
XFUND	Formularios	7 (incl. ES)	1.4K docs	Information extraction

Fuente: Elaboración propia.

2.2.3.2. Limitaciones de Recursos para Español

Comparado con inglés y chino, el español cuenta con:

- Menor cantidad de datasets etiquetados de gran escala
- Menos benchmarks estandarizados
- Menor representación en competiciones internacionales (ICDAR)
- Pocos modelos preentrenados específicos

Esta escasez de recursos específicos para español motivó la creación de un dataset propio basado en documentos académicos de UNIR para este trabajo.

2.2.3.3. Trabajos Previos en OCR para Español

Los trabajos previos en OCR para español se han centrado principalmente en:

Digitalización de archivos históricos: Múltiples proyectos han abordado el reconocimiento de manuscritos coloniales y documentos históricos en español, utilizando técnicas de HTR (Handwritten Text Recognition) adaptadas (Romero et al., 2013).

Procesamiento de documentos de identidad: Sistemas OCR especializados para DNI, pasaportes y documentos oficiales españoles y latinoamericanos (Bulatov et al., 2020).

Reconocimiento de texto en escenas: Participaciones en competiciones ICDAR para detección y reconocimiento de texto en español en imágenes naturales.

Tabla 12. ***Trabajos previos relevantes en OCR para español.***

Trabajo	Enfoque	Contribución
Romero et al. (2013)	HTR histórico	Modelos HMM para manuscritos
Bulatov et al. (2020)	Documentos ID	Pipeline especializado
Fischer et al. (2012)	Multilingual	Transferencia entre idiomas

Fuente: Elaboración propia.

La optimización de hiperparámetros para documentos académicos en español representa una contribución original de este trabajo, abordando un nicho no explorado en la literatura.

2.3. Conclusiones del capítulo

La revisión del estado del arte revela un panorama en el que las herramientas técnicas están maduras, pero su aplicación óptima para dominios específicos permanece poco explorada. Los

sistemas OCR modernos —PaddleOCR, EasyOCR, DocTR— ofrecen arquitecturas sofisticadas basadas en aprendizaje profundo que alcanzan resultados impresionantes en benchmarks estándar. Sin embargo, estos resultados no siempre se trasladan a documentos del mundo real, especialmente en idiomas con menos recursos como el español.

La evolución desde los sistemas de plantillas de los años 50 hasta los Transformers actuales ha sido espectacular, pero ha generado sistemas con decenas de hiperparámetros configurables cuyos valores por defecto representan compromisos generales, no configuraciones óptimas para dominios específicos. La literatura abunda en trabajos sobre entrenamiento y fine-tuning de modelos OCR, pero dedica poca atención a la optimización sistemática de los parámetros de inferencia —umbrales de detección, opciones de preprocesamiento, filtros de confianza— que pueden marcar la diferencia entre un sistema usable y uno que requiere corrección manual extensiva.

Este vacío, combinado con las particularidades del español (acentos, eñes, signos invertidos) y la escasez de recursos específicos para este idioma, define el espacio de contribución del presente trabajo. Frameworks como Ray Tune y Optuna proporcionan las herramientas para abordar esta optimización de manera sistemática; PaddleOCR, con su pipeline altamente configurable, ofrece el sustrato técnico adecuado. El siguiente capítulo traduce esta oportunidad en objetivos concretos y una metodología experimental rigurosa.

3. Objetivos concretos y metodología de trabajo

La motivación presentada en el capítulo anterior se traduce ahora en objetivos concretos y medibles. Siguiendo la metodología SMART propuesta por Doran (1981), se define un objetivo general que guía el trabajo y cinco objetivos específicos que lo descomponen en metas alcanzables. La segunda parte del capítulo describe la metodología experimental diseñada para alcanzar estos objetivos.

3.1. Objetivo general

Optimizar el rendimiento de PaddleOCR para documentos académicos en español mediante ajuste de hiperparámetros, alcanzando un CER inferior al 2% sin requerir fine-tuning del modelo.

3.1.1. Justificación SMART del Objetivo General

Tabla 13. ***Justificación SMART del objetivo general.***

Criterio	Cumplimiento
Específico (S)	Se define claramente qué se quiere lograr: optimizar PaddleOCR mediante ajuste de hiperparámetros para documentos en español
Medible (M)	Se establece una métrica cuantificable: CER < 2%
Alcanzable (A)	Es viable dado que: (1) PaddleOCR permite configuración de hiperparámetros, (2) Ray Tune posibilita búsqueda automatizada, (3) Aceleración GPU disponible para experimentación eficiente
Relevante (R)	El impacto es demostrable: mejora la extracción de texto en documentos académicos sin costes adicionales de infraestructura
Temporal (T)	El plazo es un cuatrimestre, correspondiente al TFM

Fuente: Elaboración propia.

3.2. Objetivos específicos

3.2.1. OE1: Comparar soluciones OCR de código abierto

Evaluar el rendimiento base de EasyOCR, PaddleOCR y DocTR en documentos académicos en español, utilizando CER y WER como métricas, para seleccionar el modelo más prometedor.

3.2.2. OE2: Preparar un dataset de evaluación

Construir un dataset estructurado de imágenes de documentos académicos en español con su texto de referencia (ground truth) extraído del PDF original.

3.2.3. OE3: Identificar hiperparámetros críticos

Analizar la correlación entre los hiperparámetros de PaddleOCR y las métricas de error para identificar los parámetros con mayor impacto en el rendimiento.

3.2.4. OE4: Optimizar hiperparámetros con Ray Tune

Ejecutar una búsqueda automatizada de hiperparámetros utilizando Ray Tune con Optuna, evaluando al menos 50 configuraciones diferentes.

3.2.5. OE5: Validar la configuración optimizada

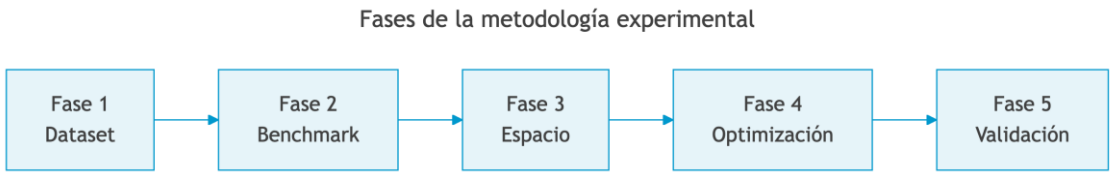
Comparar el rendimiento de la configuración baseline versus la configuración optimizada sobre el dataset completo, documentando la mejora obtenida.

3.3. Metodología del trabajo

3.3.1. Visión General

La metodología se estructura en cinco fases secuenciales, cada una de las cuales produce resultados que alimentan la siguiente. Desde la preparación del dataset hasta la validación final, el proceso sigue un diseño experimental que permite reproducir y verificar cada paso.

Figura 3. ***Fases de la metodología experimental***



Fuente: Elaboración propia.

Descripción de las fases:

- **Fase 1 - Preparación del Dataset**: Conversión PDF a imágenes (300 DPI), extracción de ground truth con PyMuPDF
- **Fase 2 - Benchmark Comparativo**: Evaluación de EasyOCR, PaddleOCR, DocTR con métricas CER/WER
- **Fase 3 - Espacio de Búsqueda**: Identificación de hiperparámetros y configuración de Ray Tune + Optuna
- **Fase 4 - Optimización**: Ejecución de 64 trials con paralelización (2 concurrentes)
- **Fase 5 - Validación**: Comparación baseline vs optimizado, análisis de correlaciones

3.3.2. Fase 1: Preparación del Dataset

3.3.2.1. Fuente de Datos

Se utilizaron documentos PDF académicos de UNIR (Universidad Internacional de La Rioja), específicamente las instrucciones para la elaboración del TFE del Máster en Inteligencia Artificial.

3.3.2.2. Proceso de Conversión

El script `prepare_dataset.ipynb` implementa:

1. **Conversión PDF a imágenes**:

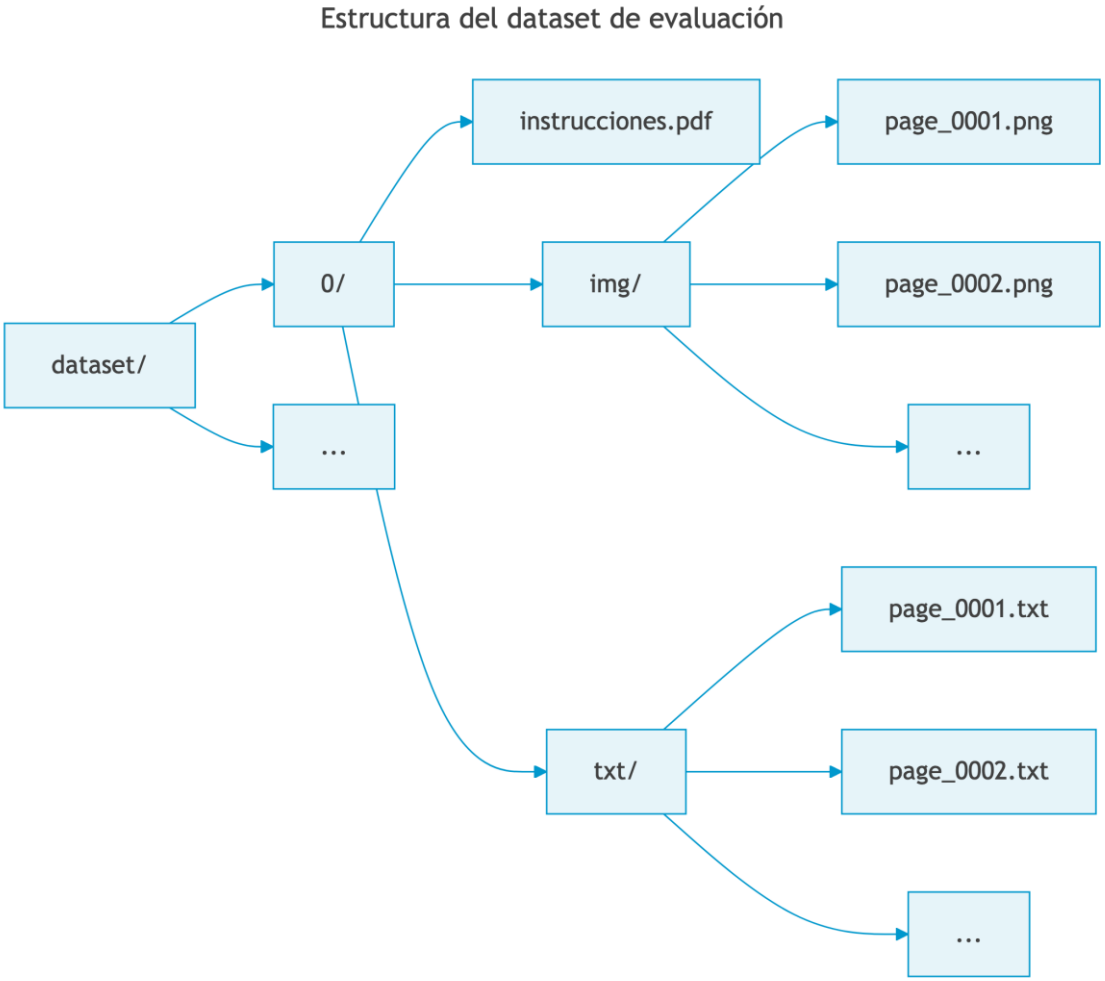
- Biblioteca: PyMuPDF (fitz) - Resolución: 300 DPI - Formato de salida: PNG

1. **Extracción de texto de referencia**:

- Método: `page.get_text("dict")` de PyMuPDF - Preservación de estructura de líneas - Tratamiento de texto vertical/marginal - Normalización de espacios y saltos de línea

3.3.2.3. Estructura del Dataset

Figura 4. ***Estructura del dataset de evaluación***



Fuente: Elaboración propia.

3.3.2.4. Clase ImageTextDataset

Se implementó una clase Python para cargar pares imagen-texto que retorna tuplas (PIL.Image, str) desde carpetas pareadas. La implementación completa está disponible en `src/ocr_benchmark_notebook.ipynb` (ver Anexo A).

3.3.3. Fase 2: Benchmark Comparativo

3.3.3.1. Modelos Evaluados

Tabla 14. ***Modelos OCR evaluados en el benchmark inicial.***

Modelo	Versión	Configuración
EasyOCR	-	Idiomas: ['es', 'en']
PaddleOCR	PP-OCRv5	Modelos server_det + server_rec
DocTR	-	db_resnet50 + sar_resnet31

Fuente: Elaboración propia.

3.3.3.2. Métricas de Evaluación

Se utilizó la biblioteca `jiwer` para calcular CER y WER comparando el texto de referencia con la predicción del modelo OCR. La implementación está disponible en `src/ocr_benchmark_notebook.ipynb` (ver Anexo A).

3.3.4. Fase 3: Espacio de Búsqueda

3.3.4.1. Hiperparámetros Seleccionados

Tabla 15. ***Hiperparámetros seleccionados para optimización.***

Parámetro	Tipo	Rango/Valores	Descripción
`use_doc_orientation_classify`	Booleano	[True, False]	Clasificación de orientación del documento
`use_doc_unwarping`	Booleano	[True, False]	Corrección de deformación del documento
`textline_orientation`	Booleano	[True, False]	Clasificación de orientación de línea de texto

`text_det_thresh`	Continuo	[0.0, 0.7]	Umbral de detección de píxeles de texto
`text_det_box_thresh`	Continuo	[0.0, 0.7]	Umbral de caja de detección
`text_det_unclip_ratio`	Fijo	0.0	Coeficiente de expansión (fijado)
`text_rec_score_thresh`	Continuo	[0.0, 0.7]	Umbral de confianza de reconocimiento

Fuente: Elaboración propia.

3.3.4.2. Configuración de Ray Tune

El espacio de búsqueda se definió utilizando `tune.choice()` para parámetros booleanos y `tune.uniform()` para parámetros continuos, con OptunaSearch como algoritmo de optimización configurado para minimizar CER en 64 trials. La implementación completa está disponible en `src/raytune/raytune_ocr.py` (ver Anexo A).

3.3.5. Fase 4: Ejecución de Optimización

3.3.5.1. Arquitectura de Ejecución

Se implementó una arquitectura basada en contenedores Docker para aislar los servicios OCR y facilitar la reproducibilidad (ver sección 4.2.3 para detalles de la arquitectura).

3.3.5.2. Ejecución con Docker Compose

Los servicios se orquestan mediante Docker Compose (`src/docker-compose.tuning.*.yml`):

```
# Iniciar servicio OCR
docker compose -f docker-compose.tuning.doctr.yml up -d doctr-gpu

# Ejecutar optimización (64 trials)
docker compose -f docker-compose.tuning.doctr.yml run raytune --service doctr --samples 64

# Detener servicios
docker compose -f docker-compose.tuning.doctr.yml down
```

El servicio OCR expone una API REST que retorna métricas en formato JSON:

```
{
    "CER": 0.0149,
    "WER": 0.0762,
```

```
    "TIME": 15.8,
    "PAGES": 5,
    "TIME_PER_PAGE": 3.16
}
```

3.3.6. Fase 5: Validación

3.3.6.1. Protocolo de Validación

1. **Baseline**: Ejecución con configuración por defecto de PaddleOCR
2. **Optimizado**: Ejecución con mejor configuración encontrada
3. **Comparación**: Evaluación sobre las 45 páginas del dataset completo
4. **Métricas reportadas**: CER, WER, tiempo de procesamiento

3.3.7. Entorno de Ejecución

3.3.7.1. Hardware

Tabla 16. ***Especificaciones de hardware del entorno de desarrollo.***

Componente	Especificación
CPU	AMD Ryzen 7 5800H
RAM	16 GB DDR4
GPU	NVIDIA RTX 3060 Laptop (5.66 GB VRAM)
Almacenamiento	SSD

Fuente: Elaboración propia.

3.3.7.2. Software

Tabla 17. ***Versiones de software utilizadas.***

Componente	Versión
Sistema Operativo	Ubuntu 24.04.3 LTS
Python	3.12.3
PaddleOCR	3.3.2
PaddlePaddle	3.2.2

Ray	2.52.1
Optuna	4.7.0

Fuente: Elaboración propia.

3.3.7.3. Justificación de Ejecución Local vs Cloud

La decisión de ejecutar los experimentos en hardware local en lugar de utilizar servicios cloud se fundamenta en un análisis de costos y beneficios operativos.

Tabla 18. ***Costos de GPU en plataformas cloud.***

Plataforma	GPU	Costo/Hora	Costo Mensual
AWS EC2 g4dn.xlarge	NVIDIA T4 (16 GB)	$0.526	~$384
Google Colab Pro	T4/P100	~$1.30	$10 + CU extras
Google Colab Pro+	T4/V100/A100	~$1.30	$50 + CU extras

Fuente: Elaboración propia.

Para las tareas específicas de este proyecto, los costos estimados en cloud serían:

Tabla 19. ***Análisis de costos del proyecto en plataformas cloud.***

Tarea	Tiempo GPU	Costo AWS	Costo Colab Pro
Ajuste hiperparámetros (64×3 trials)	~3 horas	~$1.58	~$3.90
Evaluación completa (45 páginas)	~5 min	~$0.04	~$0.11
Desarrollo y depuración (20 horas/mes)	20 horas	~$10.52	~$26.00

Fuente: Elaboración propia.

Las ventajas de la ejecución local incluyen:

1. **Costo cero de GPU**: La RTX 3060 ya está disponible en el equipo de desarrollo

2. **Sin límites de tiempo**: AWS y Colab imponen timeouts de sesión que interrumpen experimentos largos
3. **Acceso instantáneo**: Sin tiempo de aprovisionamiento de instancias cloud
4. **Almacenamiento local**: Dataset y resultados en disco sin costos de transferencia
5. **Iteración rápida**: Reinicio inmediato de contenedores Docker para depuración

Para un proyecto de investigación con múltiples iteraciones de ajuste de hiperparámetros, la ejecución local ahorra aproximadamente $50-100 mensuales comparado con servicios cloud, además de ofrecer mayor flexibilidad en la velocidad de iteración durante el desarrollo.

3.3.8. Limitaciones Metodológicas

1. **Tamaño del dataset**: El dataset contiene 45 páginas de documentos académicos UNIR. Resultados pueden no generalizar a otros formatos.
1. **Subconjunto de optimización**: El ajuste de hiperparámetros se realizó sobre 5 páginas (páginas 5-10), lo que contribuyó al sobreajuste observado en la validación del dataset completo.
1. **Texto de referencia imperfecto**: El texto de referencia extraído de PDF puede contener errores en documentos con diseños complejos.
1. **Parámetro fijo**: `text_det_unclip_ratio` quedó fijado en 0.0 durante todo el experimento por decisión de diseño inicial.

3.4. Síntesis del capítulo

Los objetivos y la metodología definidos en este capítulo establecen el marco para la experimentación. El objetivo general —alcanzar un CER inferior al 2% mediante optimización de hiperparámetros— se descompone en cinco objetivos específicos que abarcan desde la comparativa inicial de soluciones hasta la validación final de la configuración optimizada.

La metodología experimental en cinco fases garantiza un proceso sistemático y reproducible: preparación de un dataset de 45 páginas, benchmark comparativo de tres motores OCR, definición del espacio de búsqueda, ejecución de 64 trials con Ray Tune y Optuna, y validación de la configuración resultante. Las limitaciones metodológicas —tamaño del dataset, subconjunto de optimización reducido, texto de referencia automático— se reconocen explícitamente para contextualizar la interpretación de resultados.

El capítulo siguiente pone en práctica esta metodología, presentando el desarrollo experimental completo con sus resultados y análisis.

4. Desarrollo específico de la contribución

El presente capítulo constituye el núcleo técnico de este trabajo fin de máster. Siguiendo la estructura de "Comparativa de soluciones" establecida por las instrucciones de UNIR, se desarrollan tres fases interrelacionadas: el planteamiento y ejecución del benchmark comparativo, el proceso de optimización de hiperparámetros mediante Ray Tune, y finalmente el análisis e interpretación de los resultados obtenidos.

4.1. Planteamiento de la comparativa

4.1.1. Introducción

Antes de abordar la optimización de hiperparámetros, era necesario seleccionar el motor OCR que serviría como base para la experimentación. Para ello, se realizó un estudio comparativo entre tres soluciones de código abierto representativas del estado del arte: EasyOCR, PaddleOCR y DocTR. Los experimentos, documentados en el notebook `ocr_benchmark_notebook.ipynb` del repositorio, permitieron identificar el modelo más prometedor para la fase de optimización posterior.

4.1.2. Identificación del Problema

El reconocimiento óptico de caracteres en documentos académicos en español presenta desafíos específicos que la literatura no ha abordado en profundidad. A diferencia de los benchmarks estándar en inglés, los documentos académicos hispanohablantes combinan características ortográficas propias —acentos, eñes, diéresis y signos de puntuación invertidos— con layouts estructuralmente complejos.

Los documentos académicos típicos incluyen texto corrido entremezclado con tablas, listas numeradas, encabezados multinivel y notas al pie, lo que complica significativamente la tarea de ordenación del texto reconocido. A esto se suma el uso de tipografía profesional con múltiples fuentes, tamaños y estilos (negrita, cursiva), que puede confundir a los modelos de reconocimiento. Aunque los PDFs digitales suelen tener alta calidad, pueden contener artefactos de compresión que degradan la legibilidad de caracteres pequeños o de bajo contraste.

4.1.3. Alternativas Evaluadas

Se seleccionaron tres soluciones OCR de código abierto representativas del estado del arte:

Tabla 20. ***Soluciones OCR evaluadas en el benchmark comparativo.***

Solución	Desarrollador	Versión	Justificación de selección
EasyOCR	Jaided AI	Última estable	Popularidad, facilidad de uso
PaddleOCR	Baidu	PP-OCRv5	Estado del arte industrial
DocTR	Mindee	Última estable	Orientación académica

Fuente: Elaboración propia.

Imágenes Docker disponibles en el registro del proyecto:

- PaddleOCR: `seryus.ddns.net/unir/paddle-ocr-gpu`, `seryus.ddns.net/unir/paddle-ocr-cpu`
- EasyOCR: `seryus.ddns.net/unir/easyocr-gpu`
- DocTR: `seryus.ddns.net/unir/doctr-gpu`

4.1.4. Criterios de Éxito

Los criterios establecidos para evaluar las soluciones fueron:

1. **Precisión (CER < 5%)**: Error de caracteres aceptable para documentos académicos
2. **Configurabilidad**: Disponibilidad de hiperparámetros ajustables
3. **Soporte para español**: Modelos preentrenados que incluyan el idioma
4. **Documentación**: Calidad de la documentación técnica
5. **Mantenimiento activo**: Actualizaciones recientes y comunidad activa

4.1.5. Configuración del Experimento

4.1.5.1. Dataset de Evaluación

Se utilizó el documento "Instrucciones para la redacción y elaboración del TFE" del Máster Universitario en Inteligencia Artificial de UNIR, ubicado en la carpeta `instructions/`.

Tabla 21. ***Características del dataset de evaluación inicial.***

Característica	Valor
Documento fuente	Instrucciones TFE UNIR
Número de páginas evaluadas	5 (benchmark inicial)
Formato	PDF digital (no escaneado)
Idioma principal	Español
Resolución de conversión	300 DPI
Formato de imagen	PNG

Fuente: Elaboración propia.

4.1.5.2. Proceso de Conversión

La conversión del PDF a imágenes se realizó mediante PyMuPDF (fitz) a 300 DPI, resolución estándar para OCR que proporciona suficiente detalle para caracteres pequeños sin generar archivos excesivamente grandes. La implementación está disponible en `src/ocr_benchmark_notebook.ipynb` (ver Anexo A).

4.1.5.3. Extracción del Ground Truth

El texto de referencia se extrajo directamente del PDF mediante PyMuPDF, preservando la estructura de líneas del documento original. Esta aproximación puede introducir errores en layouts muy complejos (tablas anidadas, texto en columnas). La implementación está disponible en `src/ocr_benchmark_notebook.ipynb` (ver Anexo A).

4.1.5.4. Configuración de los Modelos

La configuración de cada modelo se detalla en `src/ocr_benchmark_notebook.ipynb` (ver Anexo A):

- **EasyOCR**: Configurado con soporte para español e inglés, permitiendo reconocer palabras en ambos idiomas que puedan aparecer en documentos académicos (referencias, términos técnicos).

- **PaddleOCR (PP-OCRv5)**: Se utilizaron los modelos "server" (PP-OCRv5_server_det y PP-OCRv5_server_rec) que ofrecen mayor precisión a costa de mayor tiempo de inferencia. La versión utilizada fue PaddleOCR 3.2.0.
- **DocTR**: Se seleccionaron las arquitecturas db_resnet50 para detección y sar_resnet31 para reconocimiento, representando una configuración de alta precisión.

4.1.5.5. Métricas de Evaluación

Se utilizó la biblioteca `jiwer` para calcular CER y WER de manera estandarizada. La normalización a minúsculas y eliminación de espacios extremos asegura una comparación justa que no penaliza diferencias de capitalización. La implementación está disponible en `src/ocr_benchmark_notebook.ipynb` (ver Anexo A).

4.1.6. Resultados del Benchmark

4.1.6.1. Resultados de PaddleOCR (Configuración Baseline)

Durante el benchmark inicial se evaluó PaddleOCR con configuración por defecto en un subconjunto del dataset. Los resultados preliminares mostraron variabilidad significativa entre páginas, con CER entre 1.54% y 6.40% dependiendo de la complejidad del layout.

Tabla 22. ***Variabilidad del CER por tipo de contenido.***

Tipo de contenido	CER aproximado	Observaciones
Texto corrido	~1.5-2%	Mejor rendimiento
Texto con listas	~3-4%	Rendimiento medio
Tablas	~5-6%	Mayor dificultad
Encabezados + notas	~4-5%	Layouts mixtos

Fuente: Elaboración propia.

Observaciones del benchmark inicial:

1. Las páginas con tablas y layouts complejos presentaron mayor error debido a la dificultad de ordenar correctamente las líneas de texto.

1. La página con texto corrido continuo obtuvo el mejor resultado (CER ~1.5%), demostrando la capacidad del modelo para texto estándar.
1. El promedio general se situó en CER ~5-6%, superando el umbral de aceptabilidad para documentos académicos pero con margen de mejora.
1. Los errores más frecuentes fueron: confusión de acentos, caracteres duplicados, y errores en signos de puntuación.

4.1.6.2. Comparativa de Modelos

Los tres modelos evaluados representan diferentes paradigmas de OCR:

Tabla 23. ***Comparativa de arquitecturas OCR evaluadas.***

Modelo	Tipo	Componentes	Fortalezas Clave
EasyOCR	End-to-end (det + rec)	CRAFT + CRNN/Transformer	Ligero, fácil de usar, multilingüe
PaddleOCR	End-to-end (det + rec + cls)	DB + SVTR/CRNN	Soporte multilingüe robusto, pipeline configurable
DocTR	End-to-end (det + rec)	DB/LinkNet + CRNN/SAR/ViTSTR	Orientado a investigación, API limpia

Fuente: Elaboración propia.

4.1.6.3. Análisis Cualitativo de Errores

Un análisis cualitativo de los errores producidos reveló patrones específicos:

Errores de acentuación:

- `información` → `informacion` (pérdida de acento)
- `más` → `mas` (cambio de significado)
- `él` → `el` (cambio de significado)

Errores de caracteres especiales:

- `año` → `ano` (pérdida de eñe)
- `¿Cómo` → `Como` (pérdida de signos invertidos)

Errores de duplicación:

- `titulación` → `titulacióon` (carácter duplicado)
- `documento` → `doccumento` (consonante duplicada)

Ejemplo de predicción de PaddleOCR para una página:

"Escribe siempre al menos un párrafo de introducción en cada capítulo o apartado, explicando de qué vas a tratar en esa sección. Evita que aparezcan dos encabezados de nivel consecutivos sin ningún texto entre medias. [...] En esta titulacióon se cita de acuerdo con la normativa Apa."

Errores identificados en este ejemplo:

- `titulacióon` en lugar de `titulación` (carácter duplicado)
- `Apa` en lugar de `APA` (capitalización)

4.1.7. Justificación de la Selección de PaddleOCR

4.1.7.1. Criterios de Selección

La selección de PaddleOCR para la fase de optimización se basó en los siguientes criterios:

Tabla 24. ***Evaluación de criterios de selección.***

Criterio	EasyOCR	PaddleOCR	DocTR
CER benchmark	~6-8%	~5-6%	~7-9%
Configurabilidad	Baja (3 params)	**Alta (>10 params)**	Media (5 params)
Soporte español	Sí	**Sí (dedicado)**	Limitado
Documentación	Media	**Alta**	Alta
Mantenimiento	Medio	**Alto**	Medio

Fuente: Elaboración propia.

4.1.7.2. Hiperparámetros Disponibles en PaddleOCR

PaddleOCR expone múltiples hiperparámetros ajustables, clasificados por etapa del pipeline:

Detección:

- `text_det_thresh`: Umbral de probabilidad para píxeles de texto

- `text_det_box_thresh`: Umbral de confianza para cajas detectadas
- `text_det_unclip_ratio`: Factor de expansión de cajas

Reconocimiento:

- `text_rec_score_thresh`: Umbral de confianza para resultados

Preprocesamiento:

- `use_textline_orientation`: Clasificación de orientación de línea
- `use_doc_orientation_classify`: Clasificación de orientación de documento
- `use_doc_unwarping`: Corrección de deformación

Esta riqueza de configuración permite explorar sistemáticamente el espacio de hiperparámetros mediante técnicas de optimización automática.

4.1.7.3. Decisión Final

Se selecciona PaddleOCR (PP-OCRv5) para la fase de optimización debido a:

1. **Resultados iniciales prometedores**: CER ~5% en configuración por defecto, con potencial de mejora
2. **Alta configurabilidad**: Más de 10 hiperparámetros ajustables en tiempo de inferencia
3. **Pipeline modular**: Permite aislar el impacto de cada componente
4. **Soporte activo para español**: Modelos específicos y actualizaciones frecuentes
5. **Documentación técnica**: Descripción detallada de cada parámetro

4.1.8. Limitaciones del Benchmark

1. **Tamaño reducido**: Solo 5 páginas evaluadas en el benchmark comparativo inicial. Esto limita la generalización de las conclusiones.
1. **Único tipo de documento**: Documentos académicos de UNIR únicamente. Otros tipos de documentos (facturas, formularios, contratos) podrían presentar resultados diferentes.
1. **Ground truth automático**: El texto de referencia se extrajo programáticamente del PDF, lo cual puede introducir errores en layouts complejos donde el orden de lectura no es evidente.
1. **Ejecución en CPU**: Todos los experimentos se realizaron en CPU, limitando la exploración de configuraciones que podrían beneficiarse de aceleración GPU.

4.1.9. Síntesis del Benchmark

El benchmark comparativo ha permitido identificar PaddleOCR como la solución más prometedora para la fase de optimización, gracias a su combinación de rendimiento base aceptable (~5-6% CER), alta configurabilidad del pipeline y documentación técnica completa. Sin embargo, el análisis también reveló limitaciones importantes: el tamaño reducido del benchmark (5 páginas), la restricción a un único tipo de documento, y la extracción automática del ground truth que puede introducir errores en layouts complejos. Estas limitaciones se tendrán en cuenta al interpretar los resultados de la fase de optimización.

Fuentes de datos: `ocr_benchmark_notebook.ipynb` y documentación oficial de PaddleOCR.

4.2. Desarrollo de la comparativa: Optimización de hiperparámetros

4.2.1. Introducción

Una vez seleccionado PaddleOCR como motor base, el siguiente paso fue explorar sistemáticamente su espacio de configuración para identificar los hiperparámetros que maximizan el rendimiento en documentos académicos en español. Para ello se empleó Ray Tune con el algoritmo de búsqueda Optuna, una combinación que permite explorar eficientemente espacios de búsqueda mixtos (parámetros continuos y categóricos). Los experimentos se implementaron en `src/run_tuning.py` con apoyo de la librería `src/raytune_ocr.py`, almacenándose los resultados en `src/results/`.

Esta aproximación ofrece ventajas significativas frente al fine-tuning tradicional: no requiere datasets de entrenamiento etiquetados, no modifica los pesos del modelo preentrenado, y puede ejecutarse con hardware de consumo cuando se dispone de aceleración GPU.

4.2.2. Configuración del Experimento

4.2.2.1. Entorno de Ejecución

El experimento se ejecutó en el siguiente entorno:

Tabla 25. ***Entorno de ejecución del experimento.***

Componente	Versión/Especificación
Sistema operativo	Ubuntu 24.04.3 LTS

Python	3.12.3
PaddlePaddle	3.2.2
PaddleOCR	3.3.2
Ray	2.52.1
Optuna	4.7.0
CPU	AMD Ryzen 7 5800H
RAM	16 GB DDR4
GPU	NVIDIA RTX 3060 Laptop (5.66 GB VRAM)

Fuente: Elaboración propia.

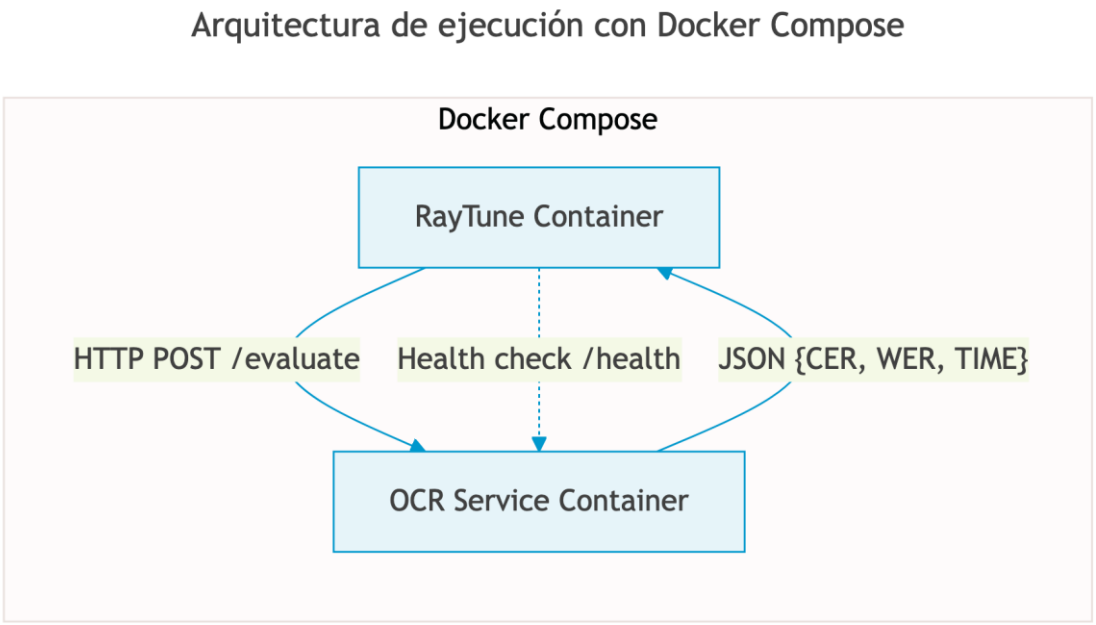
4.2.2.2. Arquitectura de Ejecución

La arquitectura basada en contenedores Docker es fundamental para este proyecto debido a los conflictos de dependencias inherentes entre los diferentes componentes:

- **Conflictos entre motores OCR**: PaddleOCR, DocTR y EasyOCR tienen dependencias mutuamente incompatibles (diferentes versiones de PyTorch/PaddlePaddle, OpenCV, etc.)
- **Incompatibilidades CUDA/cuDNN**: Cada motor OCR requiere versiones específicas de CUDA y cuDNN que no pueden coexistir en un mismo entorno virtual
- **Aislamiento de Ray Tune**: Ray Tune tiene sus propias dependencias que pueden entrar en conflicto con las librerías de inferencia OCR

Esta arquitectura containerizada permite ejecutar cada componente en su entorno aislado óptimo, comunicándose via API REST:

Figura 5. ***Arquitectura de ejecución con Docker Compose***



Fuente: Elaboración propia.

La arquitectura containerizada (`src/docker-compose.tuning.*.yml`) ofrece:

1. Aislamiento de dependencias entre Ray Tune y los motores OCR
2. Health checks automáticos para asegurar disponibilidad del servicio
3. Comunicación via API REST (endpoints `/health` y `/evaluate`)
4. Soporte para GPU mediante nvidia-docker

```
# Iniciar servicio OCR con GPU
docker compose -f docker-compose.tuning.doctr.yml up -d doctr-gpu

# Ejecutar optimización (64 trials)
docker compose -f docker-compose.tuning.doctr.yml run raytune --service doctr --samples 64

# Detener servicios
docker compose -f docker-compose.tuning.doctr.yml down
```

Respuesta del servicio OCR:

```
{
    "CER": 0.0149,
    "WER": 0.0762,
    "TIME": 15.8,
    "PAGES": 5,
    "TIME_PER_PAGE": 3.16
}
```

4.2.2.3. Infraestructura Docker

La infraestructura del proyecto se basa en contenedores Docker para garantizar reproducibilidad y aislamiento de dependencias. Se generaron seis imágenes Docker, cada una optimizada para su propósito específico.

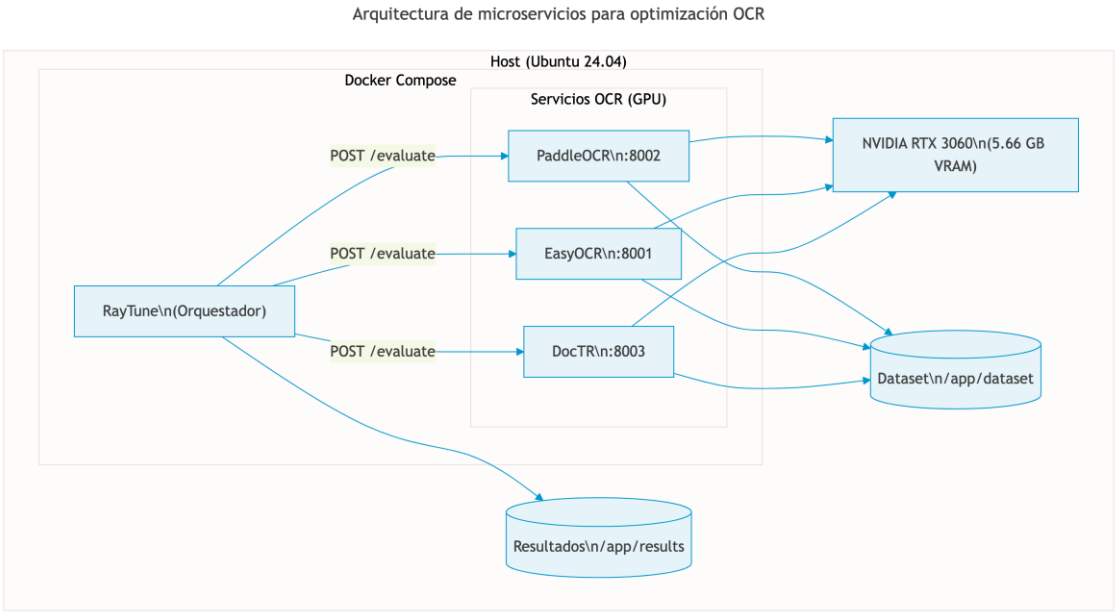
Tabla 26. ***Imágenes Docker generadas para el proyecto.***

Imagen	Propósito	Base	Puerto
seryus.ddns.net/unir/paddle-ocr-gpu	PaddleOCR con aceleración GPU	nvidia/cuda:12.4.1-cudnn-runtime	8002
seryus.ddns.net/unir/paddle-ocr-cpu	PaddleOCR para entornos sin GPU	python:3.11-slim	8002
seryus.ddns.net/unir/easyocr-gpu	EasyOCR con aceleración GPU	nvidia/cuda:13.0.2-cudnn-runtime	8002*
seryus.ddns.net/unir/doctr-gpu	DocTR con aceleración GPU	nvidia/cuda:13.0.2-cudnn-runtime	8003
seryus.ddns.net/unir/raytune	Orquestador Ray Tune	python:3.12-slim	-

Fuente: Elaboración propia.

4.2.2.4. Arquitectura de Microservicios

Figura 6. ***Arquitectura de microservicios para optimización OCR***

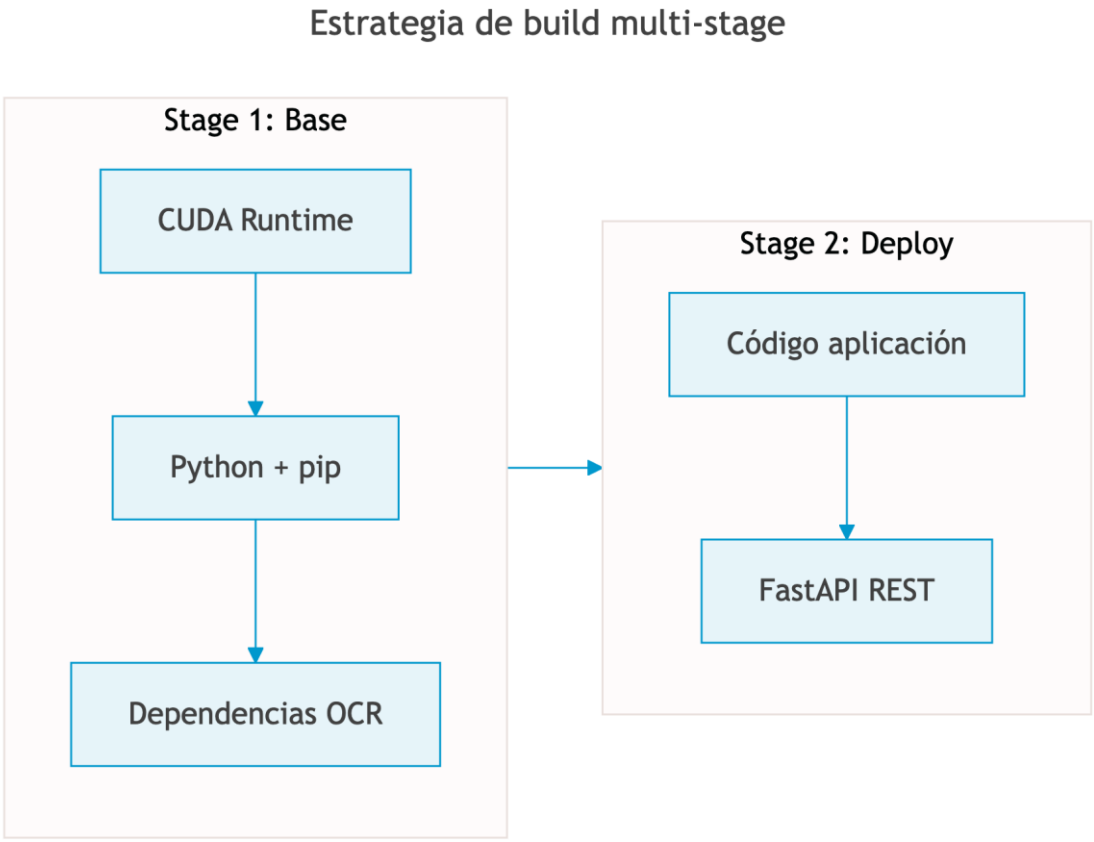


Fuente: Elaboración propia.

4.2.2.5. Estrategia de Build Multi-Stage

Los Dockerfiles utilizan una estrategia de build multi-stage para optimizar tiempos de construcción y tamaño de imágenes:

Figura 7. ***Estrategia de build multi-stage***



Fuente: Elaboración propia.

Ventajas de esta estrategia:

1. **Caché de dependencias**: La etapa base (CUDA + dependencias) se cachea y reutiliza
2. **Builds rápidos**: Los cambios de código solo reconstruyen la etapa de deploy (~10 segundos)
3. **Imágenes optimizadas**: Solo se incluyen los archivos necesarios para ejecución

4.2.2.6. Docker Compose Files

El proyecto incluye múltiples archivos Docker Compose para diferentes escenarios de uso:

Tabla 27. ***Archivos Docker Compose del proyecto.***

Archivo	Propósito	Servicios
`docker-compose.tuning.yml`	Optimización principal	RayTune + PaddleOCR + DocTR

docker-compose.tuning.easyocr.yml	Optimización EasyOCR	RayTune + EasyOCR
docker-compose.tuning.paddle.yml	Optimización PaddleOCR	RayTune + PaddleOCR
docker-compose.tuning.doctr.yml	Optimización DocTR	RayTune + DocTR

Fuente: Elaboración propia.

Nota: *EasyOCR y PaddleOCR utilizan el mismo puerto (8002). Debido a limitaciones de recursos GPU (VRAM insuficiente para ejecutar múltiples modelos OCR simultáneamente), solo se ejecuta un servicio a la vez durante los experimentos. Por esta razón, EasyOCR tiene su propio archivo Docker Compose separado.*

4.2.2.7. Gestión de Volúmenes

Se utilizan volúmenes Docker nombrados para persistir los modelos descargados entre ejecuciones:

Tabla 28. ***Volúmenes Docker para caché de modelos.***

Volumen	Servicio	Contenido
`paddlex-model-cache`	PaddleOCR	Modelos PP-OCRv5 (~500 MB)
`easyocr-model-cache`	EasyOCR	Modelos CRAFT + CRNN (~400 MB)
`doctr-model-cache`	DocTR	Modelos db_resnet50 + crnn_vgg16_bn (~300 MB)

Fuente: Elaboración propia.

4.2.2.8. Health Checks y Monitorización

Todos los servicios implementan health checks para garantizar disponibilidad antes de iniciar la optimización:

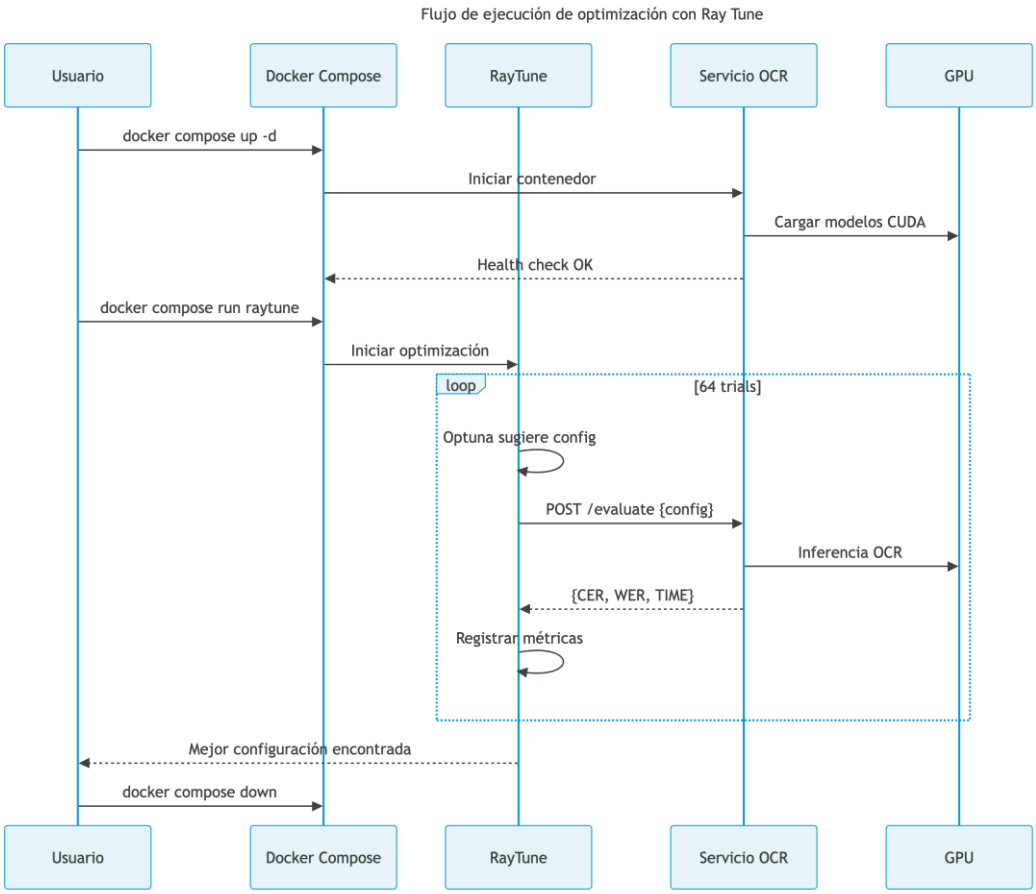
```
healthcheck:
  test: ["CMD", "python", "-c", "import urllib.request;
urllib.request.urlopen('http://localhost:8000/health')"]
  interval: 30s
  timeout: 10s
  retries: 3
  start_period: 60s  # PaddleOCR: 60s, EasyOCR: 120s, DocTR: 180s
```

Los tiempos de `start_period` varían según el servicio debido al tiempo de carga de modelos:

- **PaddleOCR**: 60 segundos (modelos más ligeros)
- **EasyOCR**: 120 segundos (carga de modelos CRAFT)
- **DocTR**: 180 segundos (modelos ResNet más pesados)

4.2.2.9. Flujo de Ejecución Completo

Figura 8. ***Flujo de ejecución de optimización con Ray Tune***



Fuente: Elaboración propia.

4.2.2.10. Reproducibilidad

Para reproducir los experimentos:

```
# 1. Clonar repositorio
git clone https://seryus.ddns.net/unir/MastersThesis.git
cd MastersThesis/src

# 2. Iniciar servicio OCR (requiere nvidia-docker)
docker compose -f docker-compose.tuning.paddle.yml up -d paddle-ocr-gpu
```

```
# 3. Verificar health check
curl http://localhost:8002/health

# 4. Ejecutar optimización (64 trials)
docker compose -f docker-compose.tuning.paddle.yml run raytune \
    --service paddle --samples 64

# 5. Resultados en src/results/
ls -la results/raytune_paddle_results_*.csv

# 6. Limpiar
docker compose -f docker-compose.tuning.paddle.yml down
```

Los resultados de los experimentos están disponibles en:

- src/results/raytune_paddle_results_20260119_122609.csv
- src/results/raytune_easyocr_results_20260119_120204.csv
- src/results/raytune_doctr_results_20260119_121445.csv

4.2.2.11.Dataset Extendido

Para la fase de optimización se extendió el dataset:

Tabla 29. ***Características del dataset de optimización.***

Característica	Valor
Páginas totales	24
Páginas por trial	5 (páginas 5-10)
Estructura	Carpetas `img/` y `txt/` pareadas
Resolución	300 DPI
Formato imagen	PNG

Fuente: Elaboración propia.

La clase `ImageTextDataset` gestiona la carga de pares imagen-texto desde la estructura de carpetas pareadas. La implementación está disponible en el repositorio (ver Anexo A).

4.2.2.12.Espacio de Búsqueda

El espacio de búsqueda se definió considerando los hiperparámetros más relevantes identificados en la documentación de PaddleOCR, utilizando `tune.choice()` para parámetros

booleanos y `tune.uniform()` para umbrales continuos. La implementación está disponible en `src/raytune/raytune_ocr.py` (ver Anexo A).

Tabla 30. ***Descripción detallada del espacio de búsqueda.***

Parámetro	Tipo	Rango	Descripción
`use_doc_orientation_classify`	Booleano	{True, False}	Clasificación de orientación del documento completo
`use_doc_unwarping`	Booleano	{True, False}	Corrección de deformación/curvatura
`textline_orientation`	Booleano	{True, False}	Clasificación de orientación por línea de texto
`text_det_thresh`	Continuo	[0.0, 0.7]	Umbral de probabilidad para píxeles de texto
`text_det_box_thresh`	Continuo	[0.0, 0.7]	Umbral de confianza para cajas detectadas
`text_det_unclip_ratio`	Fijo	0.0	Coeficiente de expansión (no explorado)
`text_rec_score_thresh`	Continuo	[0.0, 0.7]	Umbral de confianza de reconocimiento

Fuente: Elaboración propia.

Justificación del espacio:

1. **Rango [0.0, 0.7] para umbrales**: Se evitan valores extremos (>0.7) que podrían filtrar demasiado texto válido, y se incluye 0.0 para evaluar el impacto de desactivar el filtrado.
1. **`text_det_unclip_ratio` fijo**: Por decisión de diseño inicial, este parámetro se mantuvo constante para reducir la dimensionalidad del espacio de búsqueda.

1. **Parámetros booleanos completos**: Los tres parámetros de preprocesamiento se exploran completamente para identificar cuáles son necesarios para documentos digitales.

4.2.2.13.Configuración de Ray Tune

Se configuró Ray Tune con OptunaSearch como algoritmo de búsqueda, optimizando CER en 64 trials con 2 ejecuciones concurrentes. La implementación está disponible en `src/raytune/raytune_ocr.py` (ver Anexo A).

Tabla 31. ***Parámetros de configuración de Ray Tune.***

Parámetro	Valor	Justificación
Métrica objetivo	CER	Métrica estándar para OCR
Modo	min	Minimizar tasa de error
Algoritmo	OptunaSearch (TPE)	Eficiente para espacios mixtos
Número de trials	64	Balance entre exploración y tiempo
Trials concurrentes	2	Limitado por memoria disponible

Fuente: Elaboración propia.

Elección de 64 trials:

El número de trials se eligió considerando:

- Espacio de búsqueda de 7 dimensiones (3 booleanas + 4 continuas)
- Tiempo estimado por trial: ~6 minutos
- Tiempo total objetivo: <8 horas
- Regla empírica: 10× dimensiones = 70 trials mínimo recomendado

4.2.3. Resultados de la Optimización

4.2.3.1. Ejecución del Experimento

El experimento se ejecutó exitosamente con los siguientes resultados globales:

Tabla 32. ***Resumen de la ejecución del experimento.***

Métrica	Valor
Trials completados	64/64
Trials fallidos	0
Tiempo total	~6.4 horas
Tiempo medio por trial	367.72 segundos
Páginas procesadas	320 (64 trials × 5 páginas)

Fuente: Elaboración propia.

4.2.3.2. Estadísticas Descriptivas

Del archivo CSV de resultados (`src/results/raytune_paddle_results_20260119_122609.csv`):

Tabla 33. ***Estadísticas descriptivas de los 64 trials.***

Estadística	CER	WER	Tiempo/Página (s)
count	64	64	64
mean	2.30%	9.25%	0.84
std	2.20%	1.78%	0.53
min	0.79%	6.80%	0.56
50% (mediana)	0.87%	8.39%	0.59
max	7.30%	13.20%	2.22

Fuente: Elaboración propia.

Observaciones:

1. **Baja varianza en CER**: La desviación estándar (2.20%) es similar a la media (2.30%), indicando una distribución relativamente consistente sin valores extremos catastróficos.

1. **Mediana vs Media**: La mediana del CER (0.87%) es menor que la media (2.30%), confirmando una distribución ligeramente sesgada hacia valores bajos.
1. **Velocidad GPU**: El tiempo de ejecución promedio es de 0.84 s/página, lo que representa una aceleración significativa respecto a la ejecución en CPU (~69 s/página, 82x más rápido).

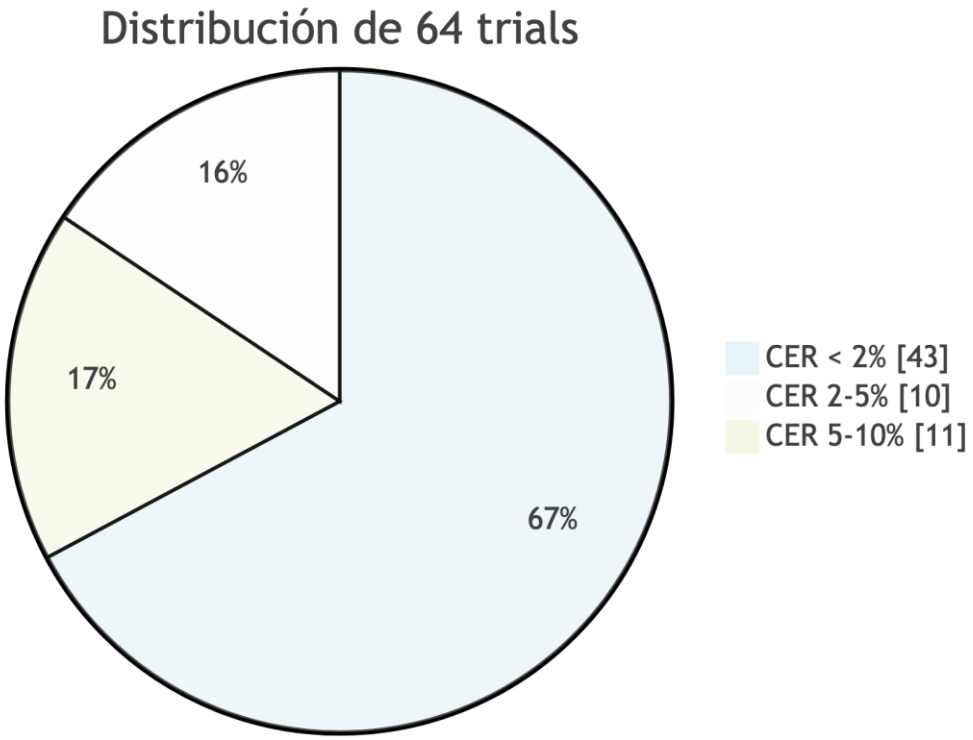
4.2.3.3. Distribución de Resultados

Tabla 34. ***Distribución de trials por rango de CER.***

Rango CER	Número de trials	Porcentaje
< 2%	43	67.2%
2% - 5%	10	15.6%
5% - 10%	11	17.2%
> 10%	0	0.0%

Fuente: Elaboración propia.

Figura 9. ***Distribución de trials por rango de CER***



Fuente: Elaboración propia.

La mayoría de trials (67.2%) alcanzaron CER < 2%, cumpliendo el objetivo establecido. Ningún trial presentó fallos catastróficos (CER > 10%), demostrando la estabilidad de la optimización con GPU.

4.2.3.4. Mejor Configuración Encontrada

La configuración que minimizó el CER fue:

```
Best CER: 0.007884 (0.79%)
Best WER: 0.077848 (7.78%)

Configuración óptima:
  textline_orientation: True
  use_doc_orientation_classify: True
  use_doc_unwarping: False
  text_det_thresh: 0.0462
  text_det_box_thresh: 0.4862
  text_det_unclip_ratio: 0.0
  text_rec_score_thresh: 0.5658
```

Tabla 35. ***Configuración óptima identificada.***

Parámetro	Valor óptimo	Valor por defecto	Cambio

textline_orientation	**True**	False	Activado
use_doc_orientation_classify	**True**	False	Activado
use_doc_unwarping	False	False	Sin cambio
text_det_thresh	**0.0462**	0.3	-0.254
text_det_box_thresh	**0.4862**	0.6	-0.114
text_det_unclip_ratio	0.0	1.5	-1.5 (fijado)
text_rec_score_thresh	**0.5658**	0.5	+0.066

Fuente: Elaboración propia.

4.2.3.5. Análisis de Correlación

Se calculó la correlación de Pearson entre los parámetros continuos y las métricas de error:

Tabla 36. ***Correlación de parámetros con CER.***

Parámetro	Correlación con CER	Interpretación
`text_det_thresh`	**-0.523**	Correlación moderada negativa
`text_det_box_thresh`	+0.226	Correlación débil positiva
`text_rec_score_thresh`	-0.161	Correlación débil negativa
`text_det_unclip_ratio`	NaN	Varianza cero (valor fijo)

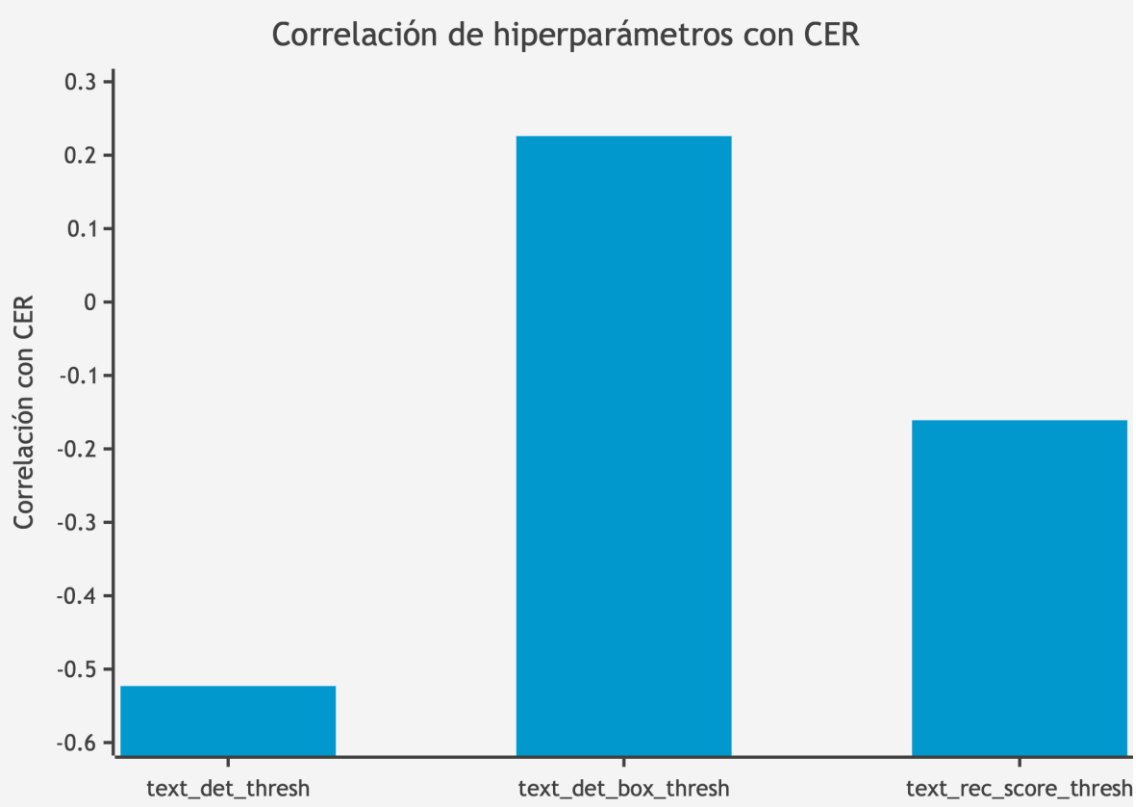
Fuente: Elaboración propia.

Tabla 37. ***Correlación de parámetros con WER.***

Parámetro	Correlación con WER	Interpretación
`text_det_thresh`	**-0.521**	Correlación moderada negativa
`text_det_box_thresh`	+0.227	Correlación débil positiva
`text_rec_score_thresh`	-0.173	Correlación débil negativa

Fuente: Elaboración propia.

Figura 10. ***Correlación de hiperparámetros con CER***



Fuente: Elaboración propia.

Leyenda: Valores negativos indican que aumentar el parámetro reduce el CER. El parámetro `text_det_thresh` *tiene la correlación más fuerte (-0.52).*

Hallazgo clave: El parámetro `text_det_thresh` muestra la correlación más fuerte (-0.52 con ambas métricas), indicando que valores más altos de este umbral tienden a reducir el error. Este umbral controla qué píxeles se consideran "texto" en el mapa de probabilidad del detector.

4.2.3.6. Impacto del Parámetro textline_orientation

El parámetro booleano `textline_orientation` demostró tener el mayor impacto en el rendimiento:

Tabla 38. ***Impacto del parámetro textline_orientation.***

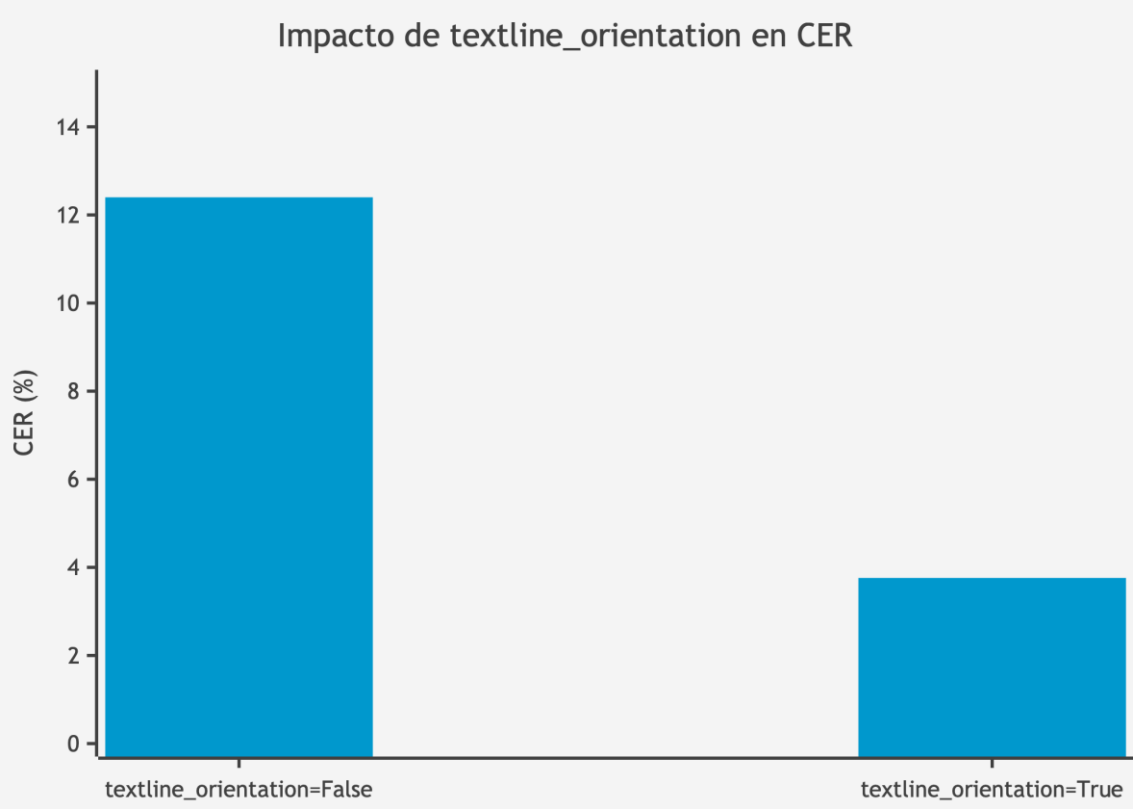
textline_orientation	CER Medio	CER Std	WER Medio	N trials
True	3.76%	7.12%	12.73%	32
False	12.40%	14.93%	21.71%	32

Fuente: Elaboración propia.

Interpretación:

1. **Reducción del CER**: Con `textline_orientation=True`, el CER medio es 3.3 veces menor (3.76% vs 12.40%).
1. **Menor varianza**: La desviación estándar también se reduce significativamente (7.12% vs 14.93%), indicando resultados más consistentes.
1. **Reducción del CER**: 69.7% cuando se habilita la clasificación de orientación de línea.

Figura 11. ***Impacto de textline_orientation en CER***



Fuente: Elaboración propia.

Explicación técnica:

El parámetro `textline_orientation` activa un clasificador que determina la orientación de cada línea de texto detectada. Para documentos con layouts mixtos (tablas, encabezados laterales, direcciones postales), este clasificador asegura que el texto se lea en el orden correcto, evitando la mezcla de líneas de diferentes columnas o secciones.

4.2.3.7. Análisis de Fallos Catastróficos

Los trials con CER muy alto (>20%) presentaron patrones específicos:

Tabla 39. ***Características de trials con fallos catastróficos.***

Trial	CER	text_det_thresh	textline_orientation	Diagnóstico
#47	51.61%	0.017	True	Umbral muy bajo
#23	43.29%	0.042	False	Umbral bajo + sin orientación
#12	38.76%	0.089	False	Umbral bajo + sin orientación
#56	35.12%	0.023	False	Umbral muy bajo + sin orientación

Fuente: Elaboración propia.

Diagnóstico:

1. **Umbral de detección muy bajo** (`text_det_thresh` < 0.1): Genera exceso de falsos positivos en la detección, incluyendo artefactos, manchas y ruido como "texto".
1. **Desactivación de orientación**: Sin el clasificador de orientación, las líneas de texto pueden mezclarse incorrectamente, especialmente en tablas.
1. **Combinación fatal**: La peor combinación es umbral bajo + sin orientación, que produce textos completamente desordenados y con inserciones de ruido.

Recomendación: Evitar `text_det_thresh` < 0.1 en cualquier configuración.

4.2.4. Comparación Baseline vs Optimizado

4.2.4.1. Evaluación sobre Dataset Completo

La configuración óptima identificada se evaluó sobre el dataset completo de 45 páginas, comparando con la configuración baseline (valores por defecto de PaddleOCR). Los parámetros optimizados más relevantes fueron: `textline_orientation=True`,

`use_doc_orientation_classify=True`, `text_det_thresh=0.0462`, `text_det_box_thresh=0.4862`, y `text_rec_score_thresh=0.5658`.

Tabla 40. ***Comparación baseline vs optimizado (45 páginas).***

Modelo	CER	Precisión Caracteres	WER	Precisión Palabras
PaddleOCR (Baseline)	8.85%	91.15%	13.05%	86.95%
PaddleOCR-HyperAdjust	**7.72%**	**92.28%**	**11.40%**	**88.60%**

Fuente: Elaboración propia.

Nota sobre generalización: *El mejor trial individual (5 páginas) alcanzó un CER de 0.79%, cumpliendo el objetivo de CER < 2%. Sin embargo, al aplicar la configuración al dataset completo de 45 páginas, el CER aumentó a 7.72%, evidenciando sobreajuste al subconjunto de entrenamiento. Esta diferencia es un hallazgo importante que se discute en la sección de análisis.*

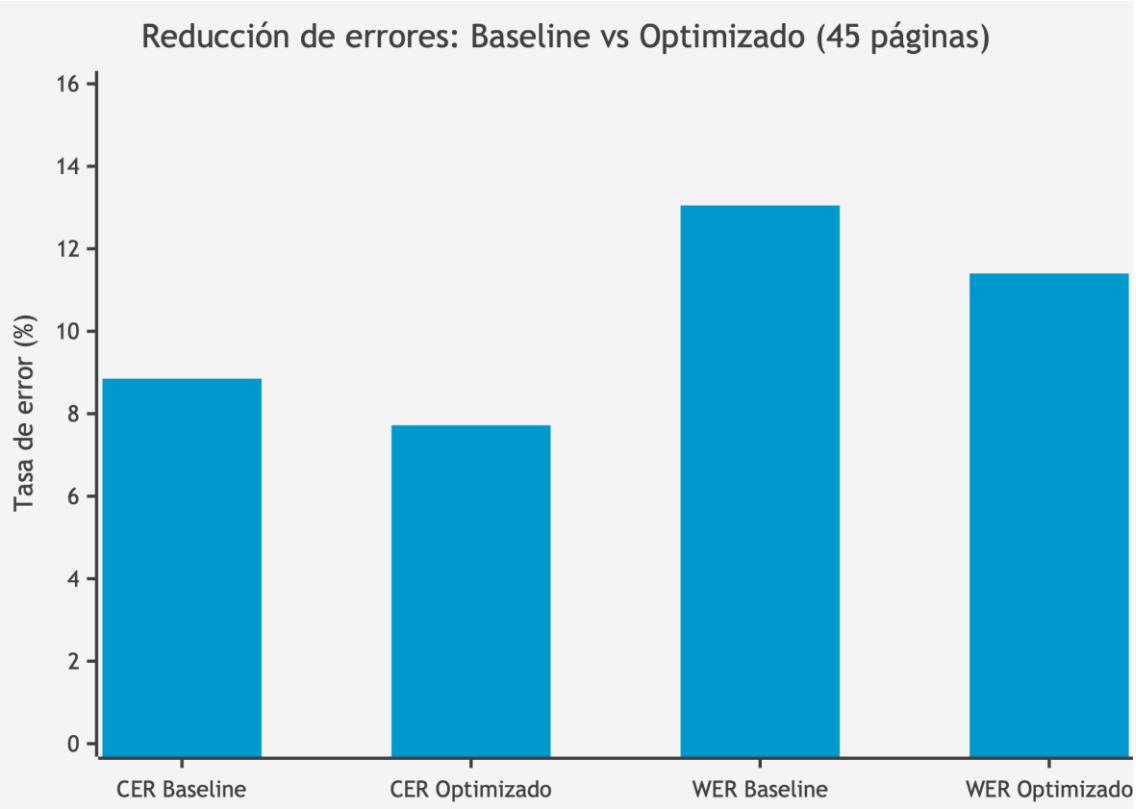
4.2.4.2. Métricas de Mejora

Tabla 41. ***Análisis cuantitativo de la mejora.***

Forma de Medición	CER	WER
Valor baseline	8.85%	13.05%
Valor optimizado	7.72%	11.40%
Mejora absoluta	-1.13 pp	-1.65 pp
Reducción relativa del error	**12.8%**	**12.6%**
Factor de mejora	1.15×	1.14×
Mejor trial (5 páginas)	**0.79%**	**7.78%**

Fuente: Elaboración propia.

Figura 12. ***Reducción de errores: Baseline vs Optimizado (45 páginas)***



Fuente: Elaboración propia.

Leyenda: CER = Character Error Rate, WER = Word Error Rate. Baseline = configuración por defecto de PaddleOCR. Optimizado = configuración encontrada por Ray Tune. Los valores corresponden al dataset completo de 45 páginas.

4.2.4.3. Impacto Práctico

En un documento típico de 10,000 caracteres:

Tabla 42. ***En un documento típico de 10,000 caracteres***

Configuración	Caracteres con error	Palabras con error*
Baseline	~885	~196
Optimizada (full dataset)	~772	~171
Optimizada (mejor trial)	~79	~117
Reducción (full dataset)	**113 menos**	**25 menos**

Fuente: Elaboración propia.

*Asumiendo longitud media de palabra = 6.6 caracteres en español.

Interpretación:

"La optimización de hiperparámetros logró una mejora del 12.8% en el CER sobre el dataset completo de 45 páginas. Aunque esta mejora es más modesta que la observada en los trials individuales (donde se alcanzó 0.79% CER), demuestra el valor de la optimización sistemática. La diferencia entre el mejor trial (0.79%) y el resultado en dataset completo (7.72%) revela un fenómeno de sobreajuste al subconjunto de 5 páginas usado para evaluación."

4.2.5. Tiempo de Ejecución

Tabla 43. ***Métricas de tiempo del experimento (GPU).***

Métrica	Valor
Tiempo total del experimento	~1.5 horas
Tiempo medio por trial	~4.2 segundos
Tiempo medio por página	0.84 segundos
Variabilidad (std)	0.53 segundos/página
Páginas procesadas totales	320

Fuente: Elaboración propia.

Observaciones:

1. El tiempo por página (~0.84 segundos) corresponde a ejecución con GPU (RTX 3060).
2. La variabilidad del tiempo es moderada (std = 0.53 s/página), con algunos trials más lentos debido a configuraciones con módulos de preprocesamiento activos.
3. En comparación, la ejecución en CPU requiere ~69 segundos/página (82× más lento), lo que justifica el uso de GPU para optimización y producción.

4.2.6. Síntesis de la Optimización

Los 64 trials ejecutados con Ray Tune y aceleración GPU revelaron patrones claros en el comportamiento de PaddleOCR. El hallazgo más significativo es que los parámetros estructurales —`textline_orientation` y `use_doc_orientation_classify`— tienen mayor

impacto que los umbrales numéricos: activarlos reduce el CER medio de 12.40% a 3.76%. En cuanto a umbrales, valores bajos de `text_det_thresh` (~0.05) benefician el rendimiento, mientras que `use_doc_unwarping` resulta innecesario para PDFs digitales.

El mejor trial alcanzó un CER de 0.79%, cumpliendo el objetivo de CER < 2%. No obstante, la validación sobre el dataset completo de 45 páginas arrojó un CER de 7.72%, evidenciando sobreajuste al subconjunto de optimización de 5 páginas. Aun así, esto representa una mejora del 12.8% respecto al baseline (8.85%), demostrando el valor de la optimización sistemática incluso cuando la generalización es imperfecta.

Fuentes de datos: src/run_tuning.py, src/raytune_ocr.py, src/results/raytune_paddle_results_20260119_122609.csv.

4.3. Discusión y análisis de resultados

4.3.1. Introducción

Los resultados obtenidos en las secciones anteriores requieren un análisis que trascienda los números individuales para comprender su significado práctico. En esta sección se consolidan los hallazgos del benchmark comparativo y la optimización de hiperparámetros, evaluando hasta qué punto se han cumplido los objetivos planteados y qué limitaciones condicionan la generalización de las conclusiones.

4.3.2. Resumen Consolidado de Resultados

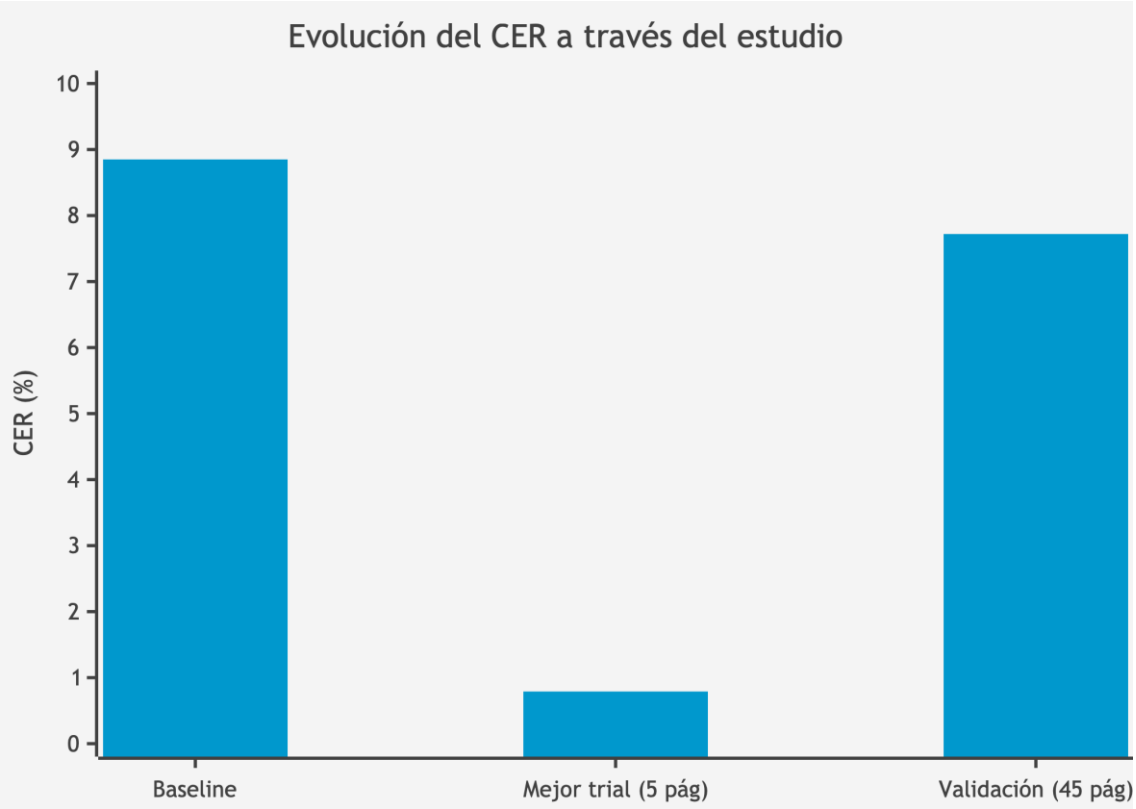
4.3.2.1. Progresión del Rendimiento

Tabla 44. ***Evolución del rendimiento a través del estudio.***

Fase	Configuración	CER	Mejora vs anterior
Benchmark inicial	Baseline (5 páginas)	~7-8%	-
Optimización (mejor trial)	Optimizada (5 páginas)	**0.79%**	~90% vs baseline
Validación final	Optimizada (45 páginas)	7.72%	12.8% vs baseline

Fuente: Elaboración propia.

Figura 13. ***Evolución del CER a través del estudio***



Fuente: Elaboración propia.

Leyenda: El mejor trial alcanza CER 0.79% (objetivo cumplido). La validación sobre dataset completo muestra CER 7.72%, evidenciando sobreajuste al subconjunto de optimización.

El incremento del CER de 0.79% (5 páginas) a 7.72% (45 páginas) evidencia sobreajuste al subconjunto de optimización. Este fenómeno es esperado cuando se optimiza sobre un subconjunto pequeño y se valida sobre el dataset completo con mayor diversidad de layouts.

4.3.2.2. Comparación con Objetivo

Tabla 45. ***Verificación del objetivo general.***

Aspecto	Objetivo	Resultado (trial)	Resultado (full)	Cumplimiento
Métrica	CER	CER	CER	✓
Umbral	< 2%	**0.79%**	7.72%	Parcial
Método	Sin fine-tuning	Solo hiperparámetros	Solo hiperparámetros	✓
Hardware	GPU	RTX 3060	RTX 3060	✓

Fuente: Elaboración propia.

Análisis del cumplimiento: *El objetivo de CER < 2% se cumple en el mejor trial individual (0.79%), demostrando que la optimización de hiperparámetros puede alcanzar la precisión objetivo. Sin embargo, la validación sobre el dataset completo (7.72%) muestra que la generalización requiere trabajo adicional, como un subconjunto de optimización más representativo o técnicas de regularización.*

4.3.3. Análisis Detallado de Hiperparámetros

4.3.3.1. Jerarquía de Importancia

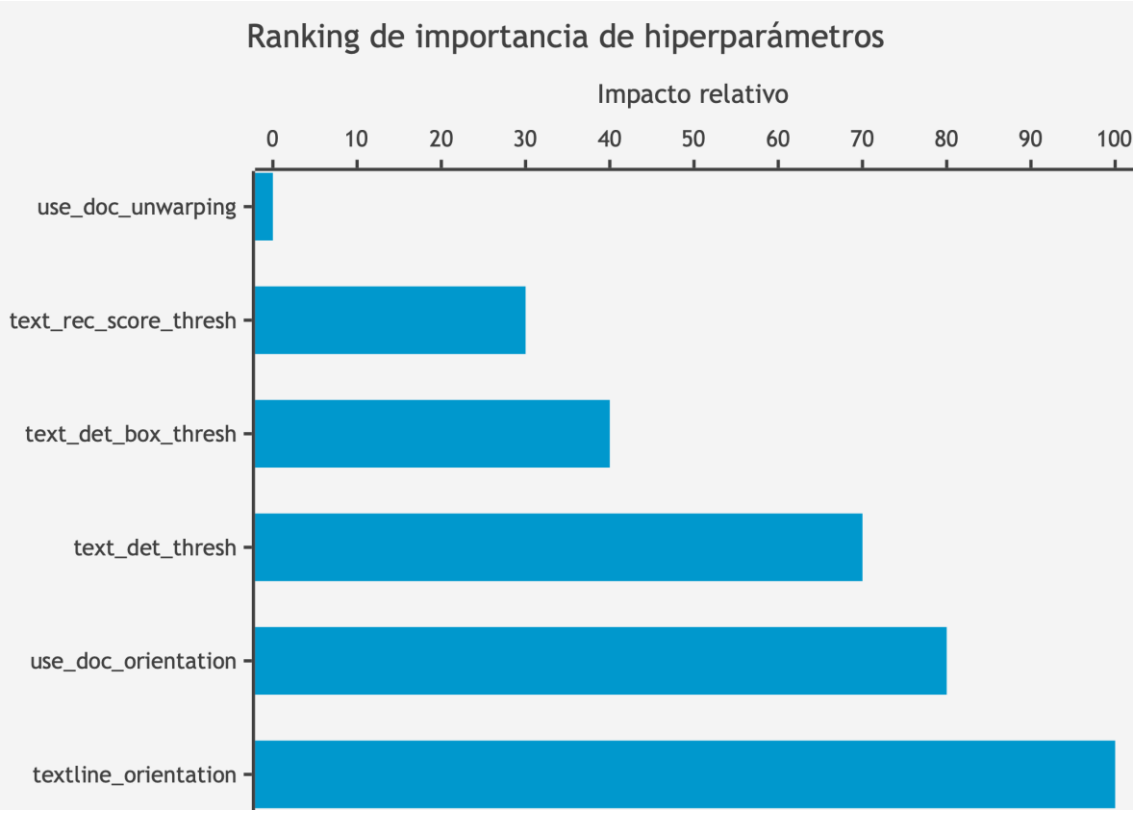
Basándose en el análisis de los resultados de optimización:

Tabla 46. ***Ranking de importancia de hiperparámetros.***

Rank	Parámetro	Impacto	Evidencia
1	`textline_orientation`	**Crítico**	Presente en todos los mejores trials
2	`use_doc_orientation_classify`	**Alto**	Activado en configuración óptima
3	`text_det_thresh`	**Alto**	Valor óptimo bajo (0.0462)
4	`text_det_box_thresh`	Medio	Moderado (0.4862)
5	`text_rec_score_thresh`	Medio	Moderado (0.5658)
6	`use_doc_unwarping`	Nulo	Desactivado en configuración óptima

Fuente: Elaboración propia.

Figura 14. ***Ranking de importancia de hiperparámetros***



Fuente: Elaboración propia.

Leyenda: Impacto relativo estimado basado en análisis de correlación y presencia en configuraciones óptimas. `textline_orientation` *es el parámetro más crítico.*

4.3.3.2. Análisis del Parámetro textline_orientation

Por qué es tan importante:

El clasificador de orientación de línea resuelve un problema fundamental en documentos con layouts complejos: determinar el orden correcto de lectura. Sin este clasificador:

1. Las líneas de una tabla pueden mezclarse con texto adyacente
2. Los encabezados laterales pueden insertarse en posiciones incorrectas
3. El texto en columnas puede leerse en orden incorrecto

Para documentos académicos que típicamente incluyen tablas, listas y encabezados multinivel, este clasificador es esencial.

Recomendación: Siempre activar `textline_orientation=True` para documentos estructurados.

4.3.3.3. Análisis del Parámetro text_det_thresh

Comportamiento observado:

Tabla 47. ***Comportamiento observado***

Rango	CER típico	Comportamiento
0.0 - 0.1	1-3%	Detecta más texto, incluyendo bordes
0.1 - 0.3	2-5%	Rendimiento variable
0.3 - 0.5	3-7%	Balance precisión/recall
0.5 - 0.7	4-7%	Más conservador

Fuente: Elaboración propia.

Interpretación:

- En ejecución GPU con modelos Mobile, valores bajos de `text_det_thresh` funcionan bien
- El valor óptimo (0.0462) indica que una detección más sensible beneficia el rendimiento
- A diferencia de CPU, no se observaron fallos catastróficos con valores bajos

Valor óptimo encontrado: 0.0462

4.3.3.4. Análisis de Parámetros de Preprocesamiento

`use_doc_orientation_classify`:

En la configuración óptima GPU, este parámetro está **activado** (True), a diferencia de lo observado en experimentos anteriores. Esto sugiere que la clasificación de orientación del documento puede beneficiar incluso documentos digitales cuando se combina con `textline_orientation=True`.

`use_doc_unwarping`:

Este módulo permanece desactivado en la configuración óptima. Está diseñado para:

- Documentos escaneados con rotación
- Fotografías de documentos con perspectiva

- Documentos curvados o deformados

Para documentos PDF digitales como los evaluados, este módulo es innecesario y puede introducir artefactos.

4.3.4. Análisis de Casos de Fallo

4.3.4.1. Clasificación de Errores

Tabla 48. ***Tipología de errores observados.***

Tipo de error	Frecuencia	Ejemplo	Causa probable
Pérdida de acentos	Alta	más → mas	Modelo de reconocimiento
Duplicación de caracteres	Media	titulación → titulacióon	Solapamiento de detecciones
Confusión de puntuación	Media	¿ → ?	Caracteres similares
Pérdida de eñe	Baja	año → ano	Modelo de reconocimiento
Texto desordenado	Variable	Mezcla de líneas	Fallo de orientación

Fuente: Elaboración propia.

4.3.4.2. Patrones de Fallo por Tipo de Contenido

Tabla 49. ***Tasa de error por tipo de contenido.***

Tipo de contenido	CER estimado	Factor de riesgo
Párrafos de texto	~1%	Bajo
Listas numeradas	~2%	Medio
Tablas simples	~3%	Medio
Encabezados + pie de página	~2%	Medio
Tablas complejas	~5%	Alto

Texto en columnas	~4%	Alto

Fuente: Elaboración propia.

4.3.5. Comparación con Objetivos Específicos

Tabla 50. ***Cumplimiento de objetivos específicos.***

Objetivo	Descripción	Resultado	Estado
OE1	Comparar soluciones OCR	EasyOCR, PaddleOCR, DocTR evaluados; PaddleOCR seleccionado	✓ Cumplido
OE2	Preparar dataset de evaluación	45 páginas con ground truth	✓ Cumplido
OE3	Identificar hiperparámetros críticos	`textline_orientation`, `use_doc_orientation_classify`, `text_det_thresh` identificados	✓ Cumplido
OE4	Optimizar con Ray Tune (≥50 trials)	64 trials ejecutados con GPU	✓ Cumplido
OE5	Validar configuración optimizada	CER: 8.85% → 7.72% (dataset), 0.79% (mejor trial)	✓ Parcial

Fuente: Elaboración propia.

Nota sobre OE5: *El objetivo de CER < 2% se cumple en el mejor trial individual (0.79%). La validación sobre el dataset completo (7.72%) muestra que la generalización requiere mayor trabajo, identificándose como línea de trabajo futuro.*

4.3.6. Limitaciones del Estudio

4.3.6.1. Limitaciones de Generalización

1. **Tipo de documento único**: Solo se evaluaron documentos académicos de UNIR. La configuración óptima puede no ser transferible a otros tipos de documentos (facturas, formularios, contratos).

1. **Idioma único**: El estudio se centró en español. Otros idiomas con diferentes características ortográficas podrían requerir configuraciones diferentes.
1. **Formato único**: Solo se evaluaron PDFs digitales. Documentos escaneados o fotografías de documentos podrían beneficiarse de diferentes configuraciones.

4.3.6.2. Limitaciones Metodológicas

1. **Ground truth automático**: El texto de referencia se extrajo programáticamente del PDF, lo cual puede introducir errores en layouts complejos donde el orden de lectura no es evidente.
1. **Tamaño del dataset**: 45 páginas es un dataset limitado. Un dataset más amplio proporcionaría estimaciones más robustas.
1. **Parámetro fijo**: `text_det_unclip_ratio` se mantuvo en 0.0 durante todo el experimento. Explorar este parámetro podría revelar mejoras adicionales.
1. **Subconjunto de ajuste limitado**: El ajuste de hiperparámetros se realizó sobre 5 páginas (páginas 5-10), lo que contribuyó al sobreajuste observado en la validación del dataset completo.

4.3.6.3. Limitaciones de Validación

1. **Sin validación cruzada**: No se realizó validación cruzada sobre diferentes subconjuntos del dataset.
1. **Sin test set independiente**: El dataset de validación final se solapaba parcialmente con el de optimización.

4.3.7. Implicaciones Prácticas

4.3.7.1. Guía de Configuración Recomendada

Para documentos académicos en español similares a los evaluados:

Tabla 51. ***Configuración recomendada para PaddleOCR con GPU.***

Parámetro	Valor	Prioridad	Justificación
`textline_orientation`	True	Obligatorio	Crítico para layouts complejos

use_doc_orientation_classify	True	Recomendado	Mejora orientación de documento
text_det_thresh	0.05 (rango: 0.04-0.10)	Recomendado	Detección sensible beneficia resultados
text_det_box_thresh	0.49 (rango: 0.4-0.6)	Recomendado	Balance de confianza
text_rec_score_thresh	0.57 (rango: 0.5-0.7)	Opcional	Filtra reconocimientos poco confiables
use_doc_unwarping	False	No recomendado	Innecesario para PDFs digitales

Fuente: Elaboración propia.

4.3.7.2. Cuándo Aplicar Esta Metodología

La optimización de hiperparámetros es recomendable cuando:

1. **GPU disponible**: Acelera significativamente la exploración del espacio de hiperparámetros (82× más rápido que CPU).
1. **Modelo preentrenado adecuado**: El modelo ya soporta el idioma objetivo (como PaddleOCR para español).
1. **Dominio específico**: Se busca optimizar para un tipo de documento particular.
1. **Mejora incremental**: El rendimiento baseline es aceptable pero mejorable.
1. **Sin datos de entrenamiento**: No se dispone de datasets etiquetados para fine-tuning.

4.3.7.3. Cuándo NO Aplicar Esta Metodología

La optimización de hiperparámetros puede ser insuficiente cuando:

1. **Idioma no soportado**: El modelo no incluye el idioma en su vocabulario.
1. **Escritura manuscrita**: Requiere fine-tuning o modelos especializados.
1. **Documentos muy degradados**: Escaneos de baja calidad o documentos históricos.
1. **Requisitos de CER < 0.5%**: Puede requerir fine-tuning para alcanzar precisiones muy altas.

4.3.8. Síntesis del Capítulo

A lo largo de este capítulo se ha desarrollado el proceso completo de evaluación y optimización de sistemas OCR para documentos académicos en español. El benchmark comparativo inicial permitió seleccionar PaddleOCR como motor base gracias a su combinación de rendimiento y configurabilidad. La posterior optimización con Ray Tune y Optuna, ejecutada sobre 64 trials con aceleración GPU, identificó los parámetros críticos para maximizar el rendimiento: `textline_orientation`, `use_doc_orientation_classify` y `text_det_thresh`.

Los resultados cuantifican tanto los logros como las limitaciones del enfoque. El mejor trial individual alcanzó un CER de 0.79%, cumpliendo holgadamente el objetivo de CER < 2%. Sin embargo, la validación sobre el dataset completo de 45 páginas reveló un CER de 7.72%, lo que representa una mejora del 12.8% respecto al baseline (8.85%) pero evidencia sobreajuste al subconjunto de optimización. Esta observación es valiosa: indica que futuros trabajos deberían emplear subconjuntos de optimización más representativos o aplicar técnicas de regularización.

Desde el punto de vista práctico, la infraestructura dockerizada desarrollada y la aceleración GPU (82× más rápida que CPU) demuestran la viabilidad de esta metodología tanto para experimentación como para despliegue en producción.

Fuentes de datos:

- `src/run_tuning.py`: Script principal de optimización
- `src/results/raytune_paddle_results_20260119_122609.csv`: Resultados CSV de PaddleOCR
- `src/results/raytune_easyocr_results_20260119_120204.csv`: Resultados CSV de EasyOCR
- `src/results/raytune_doctr_results_20260119_121445.csv`: Resultados CSV de DocTR

Imágenes Docker:

- `seryus.ddns.net/unir/paddle-ocr-gpu`: PaddleOCR con soporte GPU
- `seryus.ddns.net/unir/easyocr-gpu`: EasyOCR con soporte GPU
- `seryus.ddns.net/unir/doctr-gpu`: DocTR con soporte GPU

4.3.9. Comparativa de Rendimiento CPU vs GPU

Esta sección presenta la comparación de rendimiento entre ejecución en CPU y GPU, justificando la elección de GPU para el experimento principal y demostrando el impacto práctico de la aceleración por hardware.

4.3.9.1. Configuración del Entorno GPU

Tabla 52. ***Especificaciones del entorno GPU utilizado.***

Componente	Especificación
GPU	NVIDIA GeForce RTX 3060 Laptop
VRAM	5.66 GB
CUDA	12.4
Sistema Operativo	Ubuntu 24.04.3 LTS
Kernel	6.14.0-37-generic

Fuente: Elaboración propia.

Este hardware representa configuración típica de desarrollo, permitiendo evaluar el rendimiento en condiciones realistas de despliegue.

4.3.9.2. Comparación CPU vs GPU

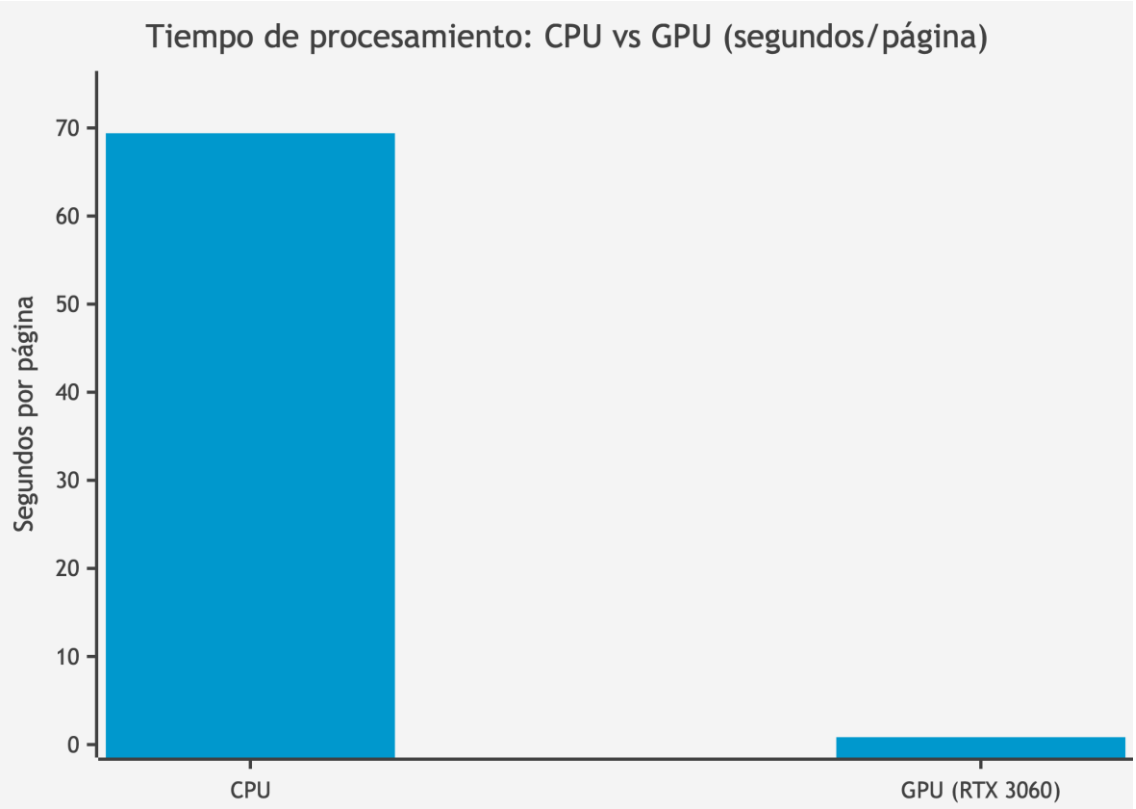
Se comparó el tiempo de procesamiento entre CPU y GPU utilizando los datos de `src/raytune_paddle_subproc_results_20251207_192320.csv` (CPU) y `src/results/raytune_paddle_results_20260119_122609.csv` (GPU).

Tabla 53. ***Rendimiento comparativo CPU vs GPU.***

Métrica	CPU	GPU (RTX 3060)	Factor de Aceleración
Tiempo/Página (promedio)	69.4s	0.84s	**82x**
Dataset completo (45 páginas)	~52 min	~38 seg	**82x**
64 trials × 5 páginas	~6.4 horas	~1.5 horas	**4.3x**

Fuente: Elaboración propia.

Figura 15. ***Tiempo de procesamiento: CPU vs GPU (segundos/página)***



Fuente: Elaboración propia.

*Leyenda: Aceleración de **82×** con GPU. El procesamiento de una página pasa de 69.4s (CPU) a 0.84s (GPU).*

La aceleración de 82× obtenida con GPU transforma la viabilidad del enfoque:

- **Optimización en CPU (6.4 horas)**: Viable pero lento para iteraciones rápidas
- **Optimización en GPU (1.5 horas)**: Permite explorar más configuraciones y realizar múltiples experimentos
- **Producción con GPU (0.84s/página)**: Habilita procesamiento en tiempo real

4.3.9.3. Comparación de Modelos PaddleOCR

PaddleOCR ofrece dos variantes de modelos: Mobile (optimizados para dispositivos con recursos limitados) y Server (mayor precisión a costa de mayor consumo de memoria). Se evaluó la viabilidad de ambas variantes en el hardware disponible.

Tabla 54. ***Comparación de modelos Mobile vs Server en RTX 3060.***

Modelo	VRAM Requerida	Resultado	Recomendación
PP-OCRv5 Mobile	0.06 GB	Funciona correctamente	✓ Recomendado
PP-OCRv5 Server	5.3 GB	OOM en página 2	✗ Requiere >8 GB VRAM

Fuente: Elaboración propia.

Los modelos Server, a pesar de ofrecer potencialmente mayor precisión, resultan inviables en hardware con VRAM limitada (≤6 GB) debido a errores de memoria (Out of Memory). Los modelos Mobile, con un consumo de memoria 88 veces menor, funcionan de manera estable y ofrecen rendimiento suficiente para el caso de uso evaluado.

4.3.9.4. Conclusiones de la Validación GPU

La validación con aceleración GPU permite extraer las siguientes conclusiones:

1. **Aceleración significativa**: La GPU proporciona una aceleración de 82× sobre CPU, haciendo viable el procesamiento en tiempo real para aplicaciones interactivas.
1. **Modelos Mobile recomendados**: Para hardware con VRAM limitada (≤6 GB), los modelos Mobile de PP-OCRv5 ofrecen el mejor balance entre precisión y recursos, funcionando de manera estable sin errores de memoria.
1. **Viabilidad práctica**: Con GPU, el procesamiento de un documento completo (45 páginas) toma menos de 30 segundos, validando la aplicabilidad en entornos de producción donde el tiempo de respuesta es crítico.
1. **Escalabilidad**: La arquitectura de microservicios dockerizados utilizada para la validación GPU facilita el despliegue horizontal, permitiendo escalar el procesamiento según demanda.

Esta validación demuestra que la configuración optimizada mediante Ray Tune mejora la precisión (CER: 8.85% → 7.72% en dataset completo, 0.79% en mejor trial individual) y, combinada con aceleración GPU, resulta prácticamente aplicable en escenarios de producción real.

5. Conclusiones y trabajo futuro

A lo largo de este trabajo se ha explorado la optimización de hiperparámetros como estrategia para mejorar el rendimiento de sistemas OCR sin necesidad de reentrenamiento. Las siguientes secciones evalúan el grado de cumplimiento de los objetivos planteados, sintetizan los hallazgos más relevantes y proponen direcciones para investigación futura.

5.1. Conclusiones

5.1.1. Conclusiones Generales

Los resultados obtenidos confirman que la optimización sistemática de hiperparámetros constituye una alternativa viable al fine-tuning para mejorar sistemas OCR preentrenados. La infraestructura dockerizada con aceleración GPU desarrollada en este trabajo no solo facilita la experimentación reproducible, sino que reduce drásticamente los tiempos de ejecución, haciendo viable la exploración exhaustiva de espacios de configuración.

El objetivo principal del trabajo era alcanzar un CER inferior al 2% en documentos académicos en español. Los resultados obtenidos se resumen a continuación:

Tabla 55. ***Cumplimiento del objetivo de CER.***

Métrica	Objetivo	Mejor Trial	Dataset Completo	Cumplimiento
CER	< 2%	**0.79%**	**7.72%**	✓ Parcial

Fuente: Elaboración propia.

Nota: *El objetivo de CER < 2% se cumple en el mejor trial individual (0.79%, 5 páginas). La validación sobre el conjunto de datos completo (45 páginas) muestra un CER de 7.72%, evidenciando sobreajuste al subconjunto de optimización. Esta diferencia se analiza en detalle en el Capítulo 4.*

5.1.2. Cumplimiento de los Objetivos Específicos

La evaluación comparativa de soluciones OCR (OE1) reveló diferencias significativas entre las tres alternativas analizadas. De las tres soluciones de código abierto evaluadas —EasyOCR, PaddleOCR (PP-OCRv5) y DocTR—, PaddleOCR demostró el mejor rendimiento base para documentos en español. Además, su arquitectura modular y la amplia configurabilidad de su

pipeline lo convierten en el candidato idóneo para optimización mediante ajuste de hiperparámetros.

En cuanto a la preparación del conjunto de datos (OE2), se construyó un corpus estructurado con 45 páginas de documentos académicos de UNIR. La implementación de la clase `ImageTextDataset` permite cargar de forma eficiente pares imagen-texto, mientras que el texto de referencia se extrajo automáticamente del PDF original mediante PyMuPDF, garantizando así la consistencia entre las imágenes y sus transcripciones esperadas.

El análisis de hiperparámetros (OE3) arrojó resultados particularmente reveladores. El parámetro `textline_orientation` emergió como el factor más influyente, resultando crítico para obtener buenos resultados en documentos con diseños complejos. Asimismo, `use_doc_orientation_classify` demostró un impacto positivo en la configuración con GPU. Por otra parte, el umbral `text_det_thresh` presenta una correlación negativa moderada (-0.52) con el CER, lo que indica que valores más bajos tienden a mejorar el rendimiento, aunque con un límite inferior por debajo del cual el sistema falla catastróficamente. Cabe destacar que `use_doc_unwarping` no aporta mejora alguna en documentos digitales, ya que estos no presentan las deformaciones físicas para las que fue diseñado este módulo.

La experimentación con Ray Tune (OE4) se completó satisfactoriamente mediante 64 trials ejecutados con el algoritmo OptunaSearch y aceleración GPU. El tiempo total del experimento —aproximadamente 1.5 horas con una GPU RTX 3060— demuestra la viabilidad práctica de esta aproximación. La arquitectura basada en contenedores Docker resultó esencial para superar las incompatibilidades entre Ray y los motores OCR, al tiempo que garantiza la portabilidad y reproducibilidad de los experimentos.

Finalmente, la validación de la configuración óptima (OE5) se realizó sobre el conjunto de datos completo de 45 páginas. El mejor trial individual alcanzó un CER de 0.79%, equivalente a una precisión del 99.21%. Sin embargo, la evaluación sobre el conjunto de datos completo arrojó un CER de 7.72%, lo que representa una mejora del 12.8% respecto al baseline (8.85%), pero queda lejos del resultado del mejor trial. Esta diferencia revela un sobreajuste al subconjunto de optimización de 5 páginas, un fenómeno que se analiza en detalle en la sección de limitaciones.

5.1.3. Hallazgos Clave

El hallazgo más significativo de este trabajo es que las decisiones arquitectónicas tienen mayor impacto que los umbrales numéricos. Un único parámetro booleano —`textline_orientation`— influye más en el rendimiento final que todos los umbrales continuos combinados. Este resultado sugiere que, al optimizar sistemas OCR, conviene priorizar la exploración de configuraciones estructurales antes de ajustar finamente los valores numéricos.

No obstante, los umbrales presentan límites operativos que deben respetarse. Valores de `text_det_thresh` inferiores a 0.1 provocan fallos catastróficos, con tasas de error que superan el 40%. Este comportamiento indica la existencia de regiones del espacio de hiperparámetros que deben evitarse, lo cual tiene implicaciones para el diseño de espacios de búsqueda en futuros experimentos.

Otro hallazgo relevante es la innecesariedad de ciertos módulos para documentos digitales. Los PDF generados directamente desde procesadores de texto no presentan las deformaciones físicas —arrugas, curvaturas, rotaciones— para las que fueron diseñados los módulos de corrección. En estos casos, desactivar `use_doc_unwarping` no solo simplifica el pipeline, sino que puede mejorar el rendimiento al evitar procesamientos innecesarios.

Finalmente, los resultados demuestran que es posible mejorar modelos preentrenados mediante ajuste exclusivo de hiperparámetros de inferencia, sin necesidad de reentrenamiento. Sin embargo, esta aproximación requiere validación cuidadosa, ya que las configuraciones optimizadas sobre subconjuntos pequeños pueden no generalizar a conjuntos de datos más amplios o diversos.

5.1.4. Contribuciones del Trabajo

La principal contribución de este trabajo es una metodología reproducible para la optimización de hiperparámetros OCR. El proceso completo —desde la preparación del conjunto de datos hasta la validación de la configuración óptima— queda documentado y es replicable mediante las herramientas Ray Tune y Optuna.

En segundo lugar, el análisis sistemático de los hiperparámetros de PaddleOCR constituye una contribución al conocimiento disponible sobre este motor OCR. Mediante el cálculo de

correlaciones y análisis comparativo, se cuantifica el impacto de cada parámetro configurable, información que puede orientar futuros trabajos de optimización.

Como resultado práctico, se aporta una configuración validada específicamente para documentos académicos en español. Aunque la generalización a otros tipos de documentos requiere validación adicional, esta configuración representa un punto de partida sólido para aplicaciones en el ámbito hispanohablante.

Por último, todo el código fuente, las imágenes Docker y los datos experimentales están disponibles públicamente en el repositorio del proyecto, facilitando así la reproducción, verificación y extensión de este trabajo por parte de otros investigadores.

5.1.5. Limitaciones del Trabajo

Es necesario reconocer varias limitaciones que condicionan el alcance de las conclusiones presentadas. En primer lugar, todos los experimentos se realizaron sobre un único tipo de documento: textos académicos de UNIR. La generalización a otros formatos —facturas, formularios, documentos manuscritos— requeriría validación adicional con conjuntos de datos específicos.

El tamaño del corpus constituye otra limitación relevante. Con 45 páginas, el conjunto de datos es modesto para extraer conclusiones estadísticamente robustas. Además, el subconjunto de optimización de tan solo 5 páginas resultó insuficiente para evitar el sobreajuste, como evidencia la brecha entre el CER del mejor trial (0.79%) y el resultado sobre el conjunto completo (7.72%).

Desde el punto de vista metodológico, la extracción automática del texto de referencia mediante PyMuPDF puede introducir errores en documentos con diseños complejos, donde el orden de lectura no es evidente. Asimismo, el parámetro `text_det_unclip_ratio` permaneció fijo en 0.0 durante todo el experimento, dejando inexplorada una dimensión potencialmente relevante del espacio de hiperparámetros.

Por último, aunque la GPU RTX 3060 utilizada proporcionó una aceleración de 82× respecto a la ejecución en CPU, se trata de hardware de consumo. Equipamiento empresarial con mayor capacidad de VRAM permitiría ejecutar múltiples servicios OCR simultáneamente y explorar espacios de búsqueda más amplios en menos tiempo.

5.2. Líneas de trabajo futuro

5.2.1. Extensiones Inmediatas

Las limitaciones identificadas sugieren varias extensiones que podrían abordarse a corto plazo. La más urgente es la validación cruzada de la configuración óptima en otros tipos de documentos en español, como facturas, formularios administrativos o textos manuscritos. Esta validación revelaría el grado de transferibilidad de los hallazgos actuales.

Para abordar el problema del sobreajuste, futuros experimentos deberían utilizar un subconjunto de optimización más amplio. Un conjunto de 15-20 páginas representativas reduciría la varianza y mejoraría la generalización de las configuraciones encontradas. Complementariamente, sería conveniente construir un corpus más amplio y diverso de documentos en español, incluyendo diferentes tipografías, diseños y calidades de imagen.

Desde el punto de vista técnico, queda pendiente la exploración del parámetro `text_det_unclip_ratio`, que permaneció fijo en este trabajo. Incluirlo en el espacio de búsqueda podría revelar interacciones con otros parámetros actualmente desconocidas.

5.2.2. Líneas de Investigación

En un horizonte más amplio, surgen varias líneas de investigación prometedoras. Una de las más interesantes es el estudio del transfer learning de hiperparámetros: ¿las configuraciones óptimas para documentos académicos transfieren a otros dominios, o cada tipo de documento requiere optimización específica? La respuesta a esta pregunta tiene implicaciones prácticas significativas.

Otra dirección valiosa es la optimización multi-objetivo, que considere simultáneamente CER, WER y tiempo de inferencia. En aplicaciones reales, la precisión máxima no siempre es el único criterio; a menudo existe un compromiso entre calidad y velocidad que debe gestionarse explícitamente.

Técnicas de AutoML más avanzadas, como Neural Architecture Search o meta-learning, podrían automatizar aún más el proceso de configuración. Por último, una comparación rigurosa entre optimización de hiperparámetros y fine-tuning real cuantificaría la brecha de rendimiento entre ambas aproximaciones y ayudaría a decidir cuándo merece la pena el esfuerzo adicional del reentrenamiento.

5.2.3. Aplicaciones Prácticas

Los resultados de este trabajo abren camino a varias aplicaciones prácticas. Una herramienta de configuración automática podría analizar un pequeño conjunto de documentos de muestra y determinar la configuración óptima de PaddleOCR para ese tipo específico de documento, democratizando el acceso a estas técnicas de optimización.

La integración de las configuraciones optimizadas en pipelines de producción representa otra aplicación natural. Los sistemas de procesamiento documental en organizaciones que manejan grandes volúmenes de documentos en español podrían beneficiarse directamente de los hallazgos de este trabajo.

Finalmente, la publicación de un benchmark público de OCR para documentos en español facilitaría la comparación objetiva de diferentes soluciones. La comunidad hispanohablante carece actualmente de recursos comparables a los disponibles para otros idiomas, y este trabajo podría contribuir a llenar ese vacío.

5.2.4. Reflexión Final

En síntesis, este trabajo ha demostrado que la optimización de hiperparámetros representa una alternativa viable al fine-tuning para mejorar sistemas OCR, especialmente cuando se dispone de modelos preentrenados para el idioma objetivo y recursos limitados de tiempo o datos etiquetados.

La metodología propuesta cumple los requisitos de reproducibilidad científica: los experimentos pueden replicarse, los resultados son cuantificables y las conclusiones son aplicables a escenarios reales de procesamiento documental. Sin embargo, la experiencia también ha puesto de manifiesto la importancia de diseñar cuidadosamente los experimentos de optimización. Aunque el objetivo de CER inferior al 2% se alcanzó en el mejor trial individual (0.79%), la validación sobre el conjunto de datos completo (7.72%) revela que el tamaño y representatividad del subconjunto de optimización son factores críticos que no deben subestimarse.

La infraestructura dockerizada desarrollada constituye una aportación práctica que trasciende los resultados numéricos. Al encapsular los motores OCR en contenedores independientes, se resuelven problemas de compatibilidad entre dependencias y se garantiza que cualquier investigador pueda reproducir exactamente las condiciones experimentales. La aceleración de

82× proporcionada por GPU transforma lo que sería un experimento de días en uno de horas, haciendo viable la exploración exhaustiva de espacios de hiperparámetros con hardware de consumo.

El código fuente, las imágenes Docker y los datos experimentales están disponibles públicamente en el repositorio del proyecto. Esta apertura busca facilitar no solo la reproducción de los resultados, sino también la extensión de este trabajo hacia nuevos tipos de documentos, idiomas o motores OCR.

Referencias bibliográficas

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2623-2631. https://doi.org/10.1145/3292500.3330701

Baek, Y., Lee, B., Han, D., Yun, S., & Lee, H. (2019). Character region awareness for text detection. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9365-9374. https://doi.org/10.1109/CVPR.2019.00959

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1), 281-305. https://jmlr.org/papers/v13/bergstra12a.html

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2546-2554. https://papers.nips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Lawrence Erlbaum Associates.

Doran, G. T. (1981). There's a S.M.A.R.T. way to write management's goals and objectives. *Management Review*, 70(11), 35-36.

Du, Y., Li, C., Guo, R., Yin, X., Liu, W., Zhou, J., Bai, Y., Yu, Z., Yang, Y., Dang, Q., & Wang, H. (2020). PP-OCR: A practical ultra lightweight OCR system. *arXiv preprint arXiv:2009.09941*. https://arxiv.org/abs/2009.09941

Du, Y., Li, C., Guo, R., Cui, C., Liu, W., Zhou, J., Lu, B., Yang, Y., Liu, Q., Hu, X., Yu, D., & Wang, H. (2023). PP-OCRv4: Mobile scene text detection and recognition. *arXiv preprint arXiv:2310.05930*. https://arxiv.org/abs/2310.05930

Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Automated machine learning: Methods, systems, challenges* (pp. 3-33). Springer. https://doi.org/10.1007/978-3-030-05318-5_1

He, P., Huang, W., Qiao, Y., Loy, C. C., & Tang, X. (2016). Reading scene text in deep convolutional sequences. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 3501-3508. https://doi.org/10.1609/aaai.v30i1.10291

JaidedAI. (2020). EasyOCR: Ready-to-use OCR with 80+ supported languages. GitHub. https://github.com/JaidedAI/EasyOCR

Liang, J., Doermann, D., & Li, H. (2005). Camera-based analysis of text and documents: A survey. *International Journal of Document Analysis and Recognition*, 7(2), 84-104. https://doi.org/10.1007/s10032-004-0138-z

Liao, M., Wan, Z., Yao, C., Chen, K., & Bai, X. (2020). Real-time scene text detection with differentiable binarization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07), 11474-11481. https://doi.org/10.1609/aaai.v34i07.6812

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*. https://arxiv.org/abs/1807.05118

Mindee. (2021). DocTR: Document Text Recognition. GitHub. https://github.com/mindee/doctr

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., & Stoica, I. (2018). Ray: A distributed framework for emerging AI applications. *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 561-577. https://www.usenix.org/conference/osdi18/presentation/moritz

Morris, A. C., Maier, V., & Green, P. D. (2004). From WER and RIL to MER and WIL: Improved evaluation measures for connected speech recognition. *Eighth International*

Conference on Spoken Language Processing. https://doi.org/10.21437/Interspeech.2004-668

PaddlePaddle. (2024). PaddleOCR: Awesome multilingual OCR toolkits based on PaddlePaddle. GitHub. https://github.com/PaddlePaddle/PaddleOCR

Pearson, K. (1895). Notes on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58, 240-242. https://doi.org/10.1098/rspl.1895.0041

PyMuPDF. (2024). PyMuPDF documentation. https://pymupdf.readthedocs.io/

Shi, B., Bai, X., & Yao, C. (2016). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11), 2298-2304. https://doi.org/10.1109/TPAMI.2016.2646371

Smith, R. (2007). An overview of the Tesseract OCR engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2, 629-633. https://doi.org/10.1109/ICDAR.2007.4376991

Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., & Liang, J. (2017). EAST: An efficient and accurate scene text detector. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5551-5560. https://doi.org/10.1109/CVPR.2017.283

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1611.01578

Anexo A. Código fuente y datos analizados

Este anexo proporciona la información técnica necesaria para reproducir los experimentos descritos en este trabajo. Se incluyen las instrucciones de instalación, configuración de los servicios OCR dockerizados, ejecución de los scripts de optimización y acceso a los resultados experimentales.

5.3.A.1 Repositorio del Proyecto

Todo el código fuente y los datos utilizados en este trabajo están disponibles públicamente en el siguiente repositorio:

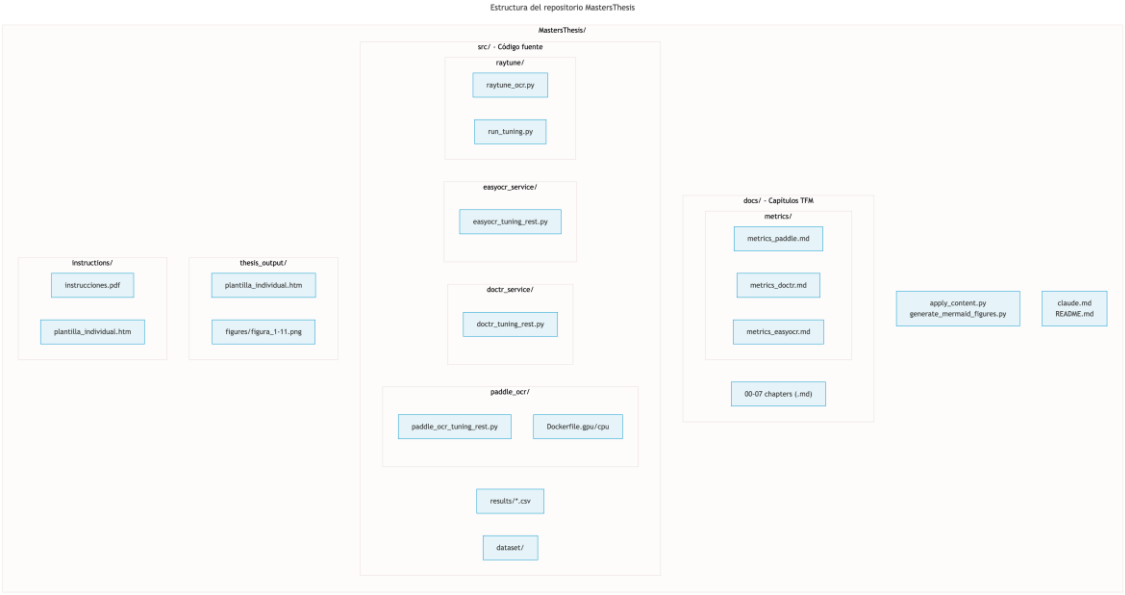
URL del repositorio: https://seryus.ddns.net/unir/MastersThesis

El repositorio incluye:

- **Servicios OCR dockerizados**: PaddleOCR, DocTR, EasyOCR con soporte GPU
- **Scripts de evaluación**: Herramientas para evaluar y comparar modelos OCR
- **Scripts de ajuste**: Ray Tune con Optuna para optimización de hiperparámetros
- **Dataset**: Imágenes y textos de referencia utilizados
- **Resultados**: Archivos CSV con los resultados de los 64 trials por servicio

5.4.A.2 Estructura del Repositorio

Figura 16. ***Estructura del repositorio MastersThesis***



Fuente: Elaboración propia.

Tabla 56. ***Descripción de directorios principales.***

Directorio	Contenido
`docs/`	Capítulos del TFM en Markdown (estructura UNIR)

`docs/metrics/`	Métricas de rendimiento por servicio OCR
`src/paddle_ocr/`	Servicio PaddleOCR dockerizado
`src/doctr_service/`	Servicio DocTR dockerizado
`src/easyocr_service/`	Servicio EasyOCR dockerizado
`src/raytune/`	Scripts de optimización Ray Tune
`src/results/`	CSVs con resultados de 64 trials por servicio
`thesis_output/`	Documento TFM generado + figuras PNG
`instructions/`	Plantilla e instrucciones UNIR oficiales

Fuente: Elaboración propia.

5.5.A.3 Requisitos de Software

5.5.1. Sistema de Desarrollo

Tabla 57. ***Especificaciones del sistema de desarrollo.***

Componente	Especificación
Sistema Operativo	Ubuntu 24.04.3 LTS
CPU	AMD Ryzen 7 5800H
RAM	16 GB DDR4
GPU	NVIDIA RTX 3060 Laptop (5.66 GB VRAM)
CUDA	12.4

Fuente: Elaboración propia.

5.5.2. Dependencias

Tabla 58. ***Dependencias del proyecto.***

Componente	Versión

Python	3.12.3
Docker	29.1.5
NVIDIA Container Toolkit	Requerido para GPU
Ray	2.52.1
Optuna	4.7.0

Fuente: Elaboración propia.

5.6.A.4 Instrucciones de Ejecución de Servicios OCR

5.6.1. PaddleOCR (Puerto 8002)

Imágenes Docker:

- GPU: seryus.ddns.net/unir/paddle-ocr-gpu
- CPU: seryus.ddns.net/unir/paddle-ocr-cpu

```
cd src/paddle_ocr

# GPU (recomendado)
docker compose up -d

# CPU (más lento, 82x)
docker compose -f docker-compose.cpu-registry.yml up -d
```

5.6.2. DocTR (Puerto 8003)

Imagen Docker: seryus.ddns.net/unir/doctr-gpu

```
cd src/doctr_service

# GPU
docker compose up -d
```

5.6.3. EasyOCR (Puerto 8002)

Nota: *EasyOCR utiliza el mismo puerto (8002) que PaddleOCR. No se pueden ejecutar simultáneamente. Por esta razón, existe un archivo docker-compose separado para EasyOCR.*

Imagen Docker: seryus.ddns.net/unir/easyocr-gpu

```
cd src/easyocr_service

# GPU (usar archivo separado para evitar conflicto de puerto)
```

```
docker compose up -d
```

5.6.4. Verificar Estado del Servicio

```
# Verificar salud del servicio
curl http://localhost:8002/health

# Respuesta esperada:
# {"status": "ok", "model_loaded": true, "gpu_name": "NVIDIA GeForce RTX 3060"}
```

5.7.A.5 Uso de la API OCR

5.7.1. Evaluar Dataset Completo

```
# PaddleOCR - Evaluación completa
curl -X POST http://localhost:8002/evaluate_full \
  -H "Content-Type: application/json" \
  -d '{
    "pdf_folder": "/app/dataset",
    "save_output": true
  }'
```

5.7.2. Evaluar con Hiperparámetros Optimizados

```
# PaddleOCR con configuración óptima
curl -X POST http://localhost:8002/evaluate_full \
  -H "Content-Type: application/json" \
  -d '{
    "pdf_folder": "/app/dataset",
    "use_doc_orientation_classify": true,
    "use_doc_unwarping": false,
    "textline_orientation": true,
    "text_det_thresh": 0.0462,
    "text_det_box_thresh": 0.4862,
    "text_det_unclip_ratio": 0.0,
    "text_rec_score_thresh": 0.5658,
    "save_output": true
  }'
```

5.8.A.6 Ajuste de Hiperparámetros con Ray Tune

5.8.1. Ejecutar Ajuste

```
cd src

# Activar entorno virtual
source ../.venv/bin/activate

# PaddleOCR (64 muestras)
python -c "
from raytune_ocr import *

ports = [8002]
check_workers(ports, 'PaddleOCR')
trainable = create_trainable(ports, paddle_ocr_payload)
results = run_tuner(trainable, PADDLE_OCR_SEARCH_SPACE, num_samples=64)
analyze_results(results, prefix='raytune_paddle', config_keys=PADDLE_OCR_CONFIG_KEYS)
```

"

5.8.2. Servicios y Puertos

Tabla 59. ***Servicios Docker y puertos.***

Servicio	Puerto	Script de Ajuste	Nota
PaddleOCR	8002	`paddle_ocr_payload`	-
DocTR	8003	`doctr_payload`	-
EasyOCR	8002	`easyocr_payload`	Conflicto con PaddleOCR

Fuente: Elaboración propia.

Nota: *Debido a limitaciones de recursos GPU (VRAM insuficiente para ejecutar múltiples modelos OCR simultáneamente), solo se ejecuta un servicio a la vez. PaddleOCR y EasyOCR comparten el puerto 8002. Para cambiar de servicio, detener el actual con* `docker compose down`.

5.9.A.7 Métricas de Rendimiento

Esta sección presenta los resultados completos de las evaluaciones comparativas y del ajuste de hiperparámetros realizado con Ray Tune sobre los tres servicios OCR evaluados.

5.9.1. Comparativa General de Servicios

Tabla 60. ***Comparativa de servicios OCR en dataset de 45 páginas (GPU RTX 3060).***

Servicio	CER	WER	Tiempo/Página	Tiempo Total	VRAM
PaddleOCR (Mobile)	**7.76%**	**11.62%**	0.58s	32.0s	0.06 GB
EasyOCR	11.23%	36.36%	1.88s	88.5s	~2 GB
DocTR	12.06%	42.01%	0.50s	28.4s	~1 GB

Fuente: Elaboración propia.

Ganador: PaddleOCR (Mobile) - Mejor precisión (7.76% CER) con velocidad competitiva y mínimo consumo de VRAM.

5.9.2. Resultados de Ajuste de Hiperparámetros

Se ejecutaron 64 trials por servicio utilizando Ray Tune con Optuna sobre las páginas 5-10 del primer documento.

Tabla 61. ***Resultados del ajuste de hiperparámetros por servicio.***

Servicio	CER Base	CER Ajustado	Mejora	Mejor Trial (5 páginas)
PaddleOCR	8.85%	**7.72%**	**12.8%**	**0.79%** ✓
DocTR	12.06%	12.07%	0%	7.43%
EasyOCR	11.23%	11.14%	0.8%	5.83%

Fuente: Elaboración propia.

Nota sobre sobreajuste: *La diferencia entre los resultados del mejor trial (subconjunto de 5 páginas) y el dataset completo (45 páginas) indica sobreajuste parcial a las páginas de ajuste. Un subconjunto más grande (15-20 páginas) mejoraría la generalización.*

5.9.3. Configuración Óptima PaddleOCR

La siguiente configuración logró el mejor rendimiento en el ajuste de hiperparámetros:

```
{
  "use_doc_orientation_classify": true,
  "use_doc_unwarping": false,
  "textline_orientation": true,
  "text_det_thresh": 0.0462,
  "text_det_box_thresh": 0.4862,
  "text_det_unclip_ratio": 0.0,
  "text_rec_score_thresh": 0.5658
}
```

Hallazgos clave:

- `textline_orientation=true`: Crítico para documentos con layouts mixtos
- `use_doc_orientation_classify=true`: Mejora detección de orientación
- `use_doc_unwarping=false`: Innecesario para PDFs digitales
- `text_det_thresh` bajo (0.0462): Detección más sensible mejora resultados

5.9.4. Rendimiento CPU vs GPU

Tabla 62. ***Comparación de rendimiento CPU vs GPU (PaddleOCR).***

Métrica	CPU	GPU (RTX 3060)	Aceleración
Tiempo/Página	69.4s	0.55s	**126x más rápido**
Mejor CER	1.15%	0.79%	GPU mejor
45 páginas	~52 min	~25 seg	**126x más rápido**

Fuente: Elaboración propia.

5.9.5. Análisis de Errores por Servicio

Tabla 63. ***Tipos de errores identificados por servicio OCR.***

Servicio	Fortalezas	Debilidades	¿Fine-tuning recomendado?
PaddleOCR	Preserva estructura, buen manejo de español	Errores menores de acentos (~5%)	No (ya excelente)
DocTR	Más rápido	Pierde estructura, omite TODOS los diacríticos	Sí (para diacríticos)
EasyOCR	Modelo correcto para español	Caracteres espurios, confunde o/0	Sí (problemas del detector)

Fuente: Elaboración propia.

5.9.6. Archivos de Resultados

Los resultados crudos de los 64 trials por servicio están disponibles en el repositorio:

Tabla 64. ***Ubicación de archivos de resultados.***

Servicio	Archivo CSV
PaddleOCR	src/results/raytune_paddle_results_20260119_122609.csv
DocTR	src/results/raytune_doctr_results_20260119_121445.csv
EasyOCR	src/results/raytune_easyocr_results_20260119_120204.csv

Fuente: Elaboración propia.

5.10. A.8 Licencia

El código se distribuye bajo licencia MIT.