Recibido 19 Feb 2026 *ReCIBE, Año 15 No. 2, junio 2026*

Aceptado 20 Jun 2026

CNN en la práctica: Guía para la evaluación de exámenes de alveolos

CNN in Practice: A Guide to Evaluating Multiple-Choice Exams.

Aldo Ivan Palma Castillo[1]
aldoivan_palma@hotmail.com

1 Universidad Nacional Autónoma de México, México

Resumen

En este documento se presenta una propuesta de sistema basado en redes neuronales convolucionales (CNN) para la automatización de la evaluación de exámenes tipo alvéolo en el Instituto Electoral del Estado de México. Se utilizó Python y TensorFlow para desarrollar un modelo capaz de detectar respuestas con alta precisión (99.18%), superando las limitaciones del procesamiento digital de imágenes (PDI) y una sensibilidad de 99.45%. Los resultados muestran la efectividad del modelo en la clasificación automática de respuestas. La investigación destaca la importancia del preprocesamiento de imágenes y la selección de la función de activación según el problema para mejorar el rendimiento. Aunque el sistema es eficiente, enfrenta limitaciones relacionadas con la diversidad del dataset y la necesidad de hardware especializado. Se proponen futuras mejoras, como la ampliación del conjunto de datos y la optimización del modelo para dispositivos con menos recursos. Este trabajo contribuye al campo de la inteligencia artificial aplicada a la educación, facilitando la evaluación automatizada de exámenes y optimizando el tiempo de revisión.

Palabras clave: Redes Neuronales Convolucionales, Inteligencia Artificial, Evaluación Automatizada, Procesamiento de Imágenes, Aprendizaje Profundo, Sector público.

Abstract

This guide presents a convolutional neural network (CNN)-based system for automating the evaluation of multiple-choice exams. Python and TensorFlow were used to develop a model capable of detecting answers with high accuracy (99.18%), overcoming the limitations of digital image processing (DIP) and a recall of 99.45%. The results show the model's effectiveness in automatic answer classification. The research highlights the importance of image preprocessing and the selection of the activation function based on the problem of enhancing performance. Although the system is efficient, it faces limitations related to dataset diversity and the need for specialized hardware. Future improvements include expanding the dataset and optimizing the model for low-resource devices. This work contributes to the field of artificial intelligence applied to education by facilitating automated exam evaluation and optimizing grading time.

Keywords: Convolutional Neural Networks, Artificial Intelligence, Automated Evaluation, Image Processing, Deep Learning, Public Sector.

Introducción

La evolución de la inteligencia artificial tiene el potencial de automatizar actividades rutinarias y repetitivas teniendo como resultado la reducción de tiempos de procesamiento y errores (Nama, 2022). En este sentido, la evaluación automatizada de exámenes ha emergido como una solución efectiva para optimizar el tiempo y los recursos en entornos educativos y organizacionales (Ramos Armijos et al., 2024). Este proyecto aborda la implementación de *redes neuronales convolucionales (CNN, por sus siglas en inglés)* para la clasificación de respuestas en exámenes tipo alvéolo, implementado por el *Instituto Electoral del Estado de México* (IEEM) como parte de un proceso de evaluación aplicado a aspirantes a funciones electorales con el objetivo de asegurar criterios objetivos y eficaces en la selección de candidatos idóneos. A través del aprendizaje profundo, las CNN han demostrado una capacidad superior para adaptarse a datos no estructurados y variados, superando en precisión y eficiencia a los métodos tradicionales de procesamiento digital de imágenes (PDI) (Bishop, 2006)

Con base en los experimentos realizados, se identificó que la principal limitación del proceso evaluado fue la lentitud inherente de la revisión manual de exámenes, lo cual motivó la exploración de una alternativa tecnológica que automatizara y acelerara este procedimiento. En

este sentido, se desarrolló una aplicación utilizando Python y TensorFlow, la cual sigue un enfoque metodológico que abarca desde la creación de un conjunto de datos (*dataset*) hasta la validación de un modelo entrenado. Este sistema no solo ofrece una solución práctica, sino que también sirve como una guía para implementar redes neuronales personalizables y ajustadas a necesidades específicas.

En el ámbito educativo, se han desarrollado herramientas basadas en procesamiento digital de imágenes para la evaluación automática de exámenes. Ejemplo de ello es la herramienta HABCO, que emplea algoritmos de análisis de intensidad de píxeles y patrones para identificar respuestas correctas en exámenes tipo alveolo, logrando alta eficiencia en entornos controlados (Mona Peña, 2016). Estas técnicas son particularmente útiles cuando se trabaja con formatos homogéneos y predefinidos, permitiendo una implementación sencilla y bajos requerimientos computacionales.

Sin embargo, el PDI enfrenta desafíos significativos al manejar datos con variaciones, como diferencias en la calidad de digitalización, interferencias visuales o formatos no estándar. Estas limitaciones han motivado la búsqueda de soluciones más avanzadas, que han demostrado ser altamente efectivas para tareas de clasificación de imágenes (LeCun, Bengio, & Hinton, 2015). Las CNN aprenden representaciones jerárquicas a partir de datos, permitiendo una mayor adaptabilidad y precisión en entornos diversos.

El uso de CNN ha sido ampliamente explorado en aplicaciones como la detección de objetos, la clasificación de documentos y el reconocimiento facial. En el contexto educativo, se han implementado para la evaluación automatizada de exámenes, reduciendo significativamente el tiempo invertido para calificar, obteniendo como resultado una mejora en la precisión (Smith et al., 2020). A diferencia de las soluciones basadas en PDI, las CNN ofrecen una capacidad de generalización que minimiza la necesidad de ajustes manuales, haciendo posible su aplicación en formatos de evaluación más complejos (Goodfellow, Bengio, & Courville, 2016).

Estas herramientas representan un avance significativo al combinar eficiencia, precisión y flexibilidad. Sin embargo, las CNN requieren mayores recursos computacionales y un diseño más complejo, lo que puede ser una barrera para su adopción en entornos con infraestructura limitada. La comparación entre el PDI tradicional y las redes neuronales convolucionales resalta cómo estas últimas representan un avance significativo en tareas de clasificación de imágenes (Gómez Pujante., 2023). Mientras que el PDI tradicional es una solución eficiente y de bajo costo para entornos controlados, las CNN destacan por su adaptabilidad y precisión en escenarios más complejos y diversos. Sin embargo, requieren mayores recursos computacionales y una implementación inicial más compleja, lo que puede limitar su adopción en contextos con infraestructura básica (**Tabla 1**). Esta evolución tecnológica subraya la importancia de seleccionar el enfoque adecuado según las necesidades específicas del problema a resolver.

Aspecto	Procesamiento Digital de Imágenes (PDI) Tradicional	Redes Neuronales Convolucionales (CNN)
Precisión	Alta en formatos homogéneos y controlados.	Superior en entornos variados con mayor complejidad visual.
Adaptabilidad	Limitada; requiere ajustes manuales para nuevos formatos.	Alta; puede generalizar con datos representativos.
Requerimientos computacionales	Bajos; ideal para entornos con hardware básico.	Altos; requiere GPUs o hardware especializado para entrenamiento.

Implementación inicial	Más rápida y sencilla; utiliza algoritmos específicos.	Más compleja; requiere diseño y entrenamiento del modelo.
Flexibilidad en aplicaciones	Menos flexible; enfocada a problemas predefinidos.	Más amplia; adaptable a diversos problemas visuales.
Mantenimiento	Requiere intervención manual frecuente.	Menor intervención una vez entrenado.
Implementación en dispositivos	Limitada; puede requerir configuraciones específicas para cada entorno; o volver a desarrollar para el dispositivo en cuestión.	Más fácil de implementar en nube, local, celulares, etc., gracias a su portabilidad y optimización, haciendo uso del mismo modelo.

Tabla 1 PDI tradicional vs CNN
Elaborado: Basada en Krizhevsky, Sutskever y Hinton, 2012; Peña, 2016; Russell y Norvig, 2004.

El potencial de las redes neuronales convolucionales en la evaluación automática de exámenes

Esta sección tiene como objetivo describir el uso de las redes neuronales convolucionales en la evaluación automática de exámenes tipo alvéolo. La inteligencia artificial (IA) es un campo de la informática que busca desarrollar sistemas capaces de realizar tareas que normalmente requieren inteligencia humana, como el reconocimiento de patrones, la toma de decisiones y el aprendizaje autónomo (Russell & Norvig, 2004). Dentro de la IA, el aprendizaje profundo (deep learning) ha emergido como una subárea clave, gracias a su capacidad para procesar grandes volúmenes de datos y extraer patrones complejos de manera autónoma (LeCun, Bengio, & Hinton, 2015). A continuación, se realiza una descripción de las redes neuronales convolucionales.

Redes Neuronales Convolucionales (CNN)

Arquitectura y funcionamiento

Las redes neuronales convolucionales (CNN) son un tipo específico de red neuronal diseñado para procesar datos que tienen una estructura de cuadrícula, como las imágenes. La arquitectura de una CNN se basa en tres tipos principales de capas (Krizhevsky, Sutskever, & Hinton, 2012):

- **Capas convolucionales:** Realizan convoluciones (filtros para obtener características) sobre la imagen de entrada para detectar patrones locales, como bordes, texturas y formas. Estas capas son responsables de extraer características jerárquicas de las imágenes.
- **Capas de pooling:** Reducen la dimensionalidad de las representaciones intermedias, conservando las características más relevantes y mejorando la eficiencia computacional.

- **Capas completamente conectadas:** Integran las características extraídas por las capas anteriores para realizar la clasificación final.

El aprendizaje en las CNN se lleva a cabo mediante un proceso iterativo conocido como propagación hacia atrás, que ajusta los pesos de la red para minimizar una función de pérdida (LeCun et al., 2015).

Entrenamiento

El entrenamiento de una CNN consiste en ajustar los pesos de sus conexiones para minimizar una función de pérdida que mide la discrepancia entre las predicciones del modelo y los valores reales. Este proceso involucra los siguientes pasos principales (Goodfellow, Bengio, & Courville, 2016):

- **Propagación hacia adelante:** Los datos de entrada atraviesan las capas de la red, generando una predicción basada en los pesos actuales.
- **Cálculo de la pérdida:** La función de pérdida compara las predicciones con los valores reales. Ejemplos comunes incluyen el error cuadrático medio para regresión y la entropía cruzada para clasificación.
- **Propagación hacia atrás:** Utilizando el algoritmo de retro propagación, el error se distribuye desde la salida hacia las capas anteriores, calculando gradientes para cada peso.
- **Actualización de pesos:** Los gradientes calculados se utilizan para actualizar los pesos de la red mediante un optimizador, como el descenso por gradiente estocástico (SGD) o Adam.

El proceso se repite para cada época (una pasada completa por los datos de entrenamiento) hasta que el modelo converge, logrando un buen desempeño en los datos de validación (Goodfellow et al., 2016).

Funciones de activación

Las funciones de activación son componentes clave en las CNN, ya que pueden introducir no linealidades en la red, permitiendo que esta aprenda relaciones complejas entre los datos de entrada y las salidas. Algunas de las funciones de activación más comunes al utilizar CNN son (Nair & Hinton, 2010; Goodfellow, Bengio, & Courville, 2016):

- **ReLU (Rectified Linear Unit):** Es la función más utilizada en tareas de clasificación de imágenes por su simplicidad y capacidad para mantener el flujo del gradiente en redes neuronales profundas, evitando el problema de la saturación que tienen otras funciones (Nair & Hinton, 2010).

$$f(x) = max(0, x)$$

- **Leaky ReLU:** Una variante de ReLU que permite pequeños valores negativos, evitando que algunas neuronas queden permanentemente inactivas durante el entrenamiento (Maas et al., 2013).

$$f(x) = \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}$$

- **Softmax:** Esta función se utiliza en la capa de salida de las redes neuronales artificiales para tareas de clasificación, ya que transforma los valores de salida en probabilidades, facilitando la asignación de una clase a cada imagen (Goodfellow et al., 2016).

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j-1}^{K} e^{z_j}}$$

En la **Figura 1** se muestran las diferencias entre las funciones de activación utilizadas en redes neuronales artificiales. La gráfica presenta cómo cada función (ReLU, Leaky ReLU y Softmax), transforma los datos de entrada, reflejando sus particularidades. Por ejemplo, ReLU se distingue por su activación a partir de valores positivos, Leaky ReLU introduce una pequeña pendiente en valores negativos, mientras que Softmax se diferencia por convertir los valores en probabilidades, proporcionando una distribución interpretable. Estas diferencias se representan en curvas que ilustran sus comportamientos matemáticos únicos.

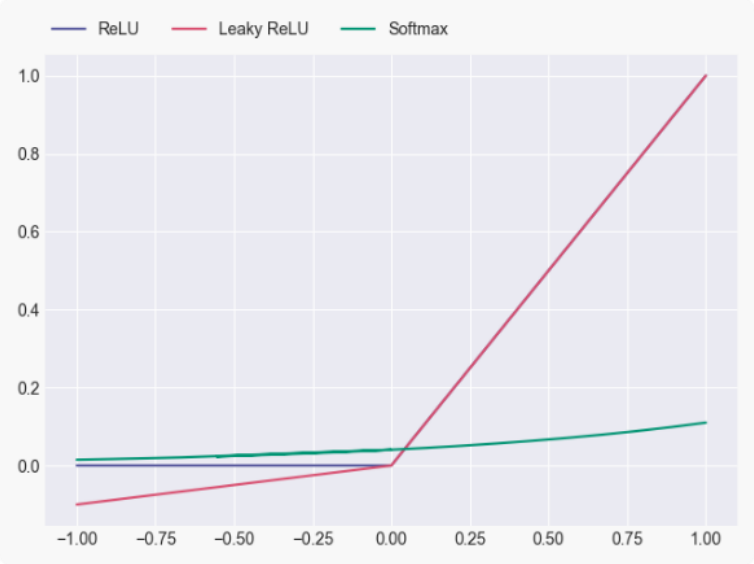

Figura 1 Comparación entre funciones de activación

Arquitectura de una CNN

Una CNN no es una simple colección de neuronas artificiales; es una secuencia cuidadosamente organizada de capas, cada una con un propósito específico en la extracción y transformación de información.

- **Capa convolucional:** La capa convolucional representa el núcleo esencial de toda arquitectura de red neuronal convolucional. A través de la operación de convolución, se aplica un conjunto de filtros sobre la imagen de entrada para identificar patrones relevantes. Posteriormente, se utiliza una *función de activación* que introduce no linealidad en las salidas, potenciando la capacidad de la red para modelar relaciones complejas entre los datos.
- **Capa de pooling:** Una debilidad de los mapas de características generados por las capas convolucionales es su sensibilidad a la localización exacta de los elementos

detectados. Incluso alteraciones mínimas en la imagen de entrada (como desplazamientos, recortes o rotaciones) pueden modificar significativamente la activación resultante. Para mitigar esta variabilidad, las redes neuronales convolucionales incluyen capas de agrupamiento (*pooling*), que promueven una representación más compacta, resistente al ruido y menos dependiente de la posición o las condiciones de iluminación del contenido analizado.

- **Capa totalmente conectada:** En la fase final de una arquitectura CNN orientada a tareas de clasificación, suele incorporarse una serie de capas completamente conectadas. En estas capas, cada unidad está enlazada con todas las neuronas de la capa previa, permitiendo una integración completa de la información extraída. La última de estas capas actúa como el clasificador, encargándose de emitir la predicción final del sistema.

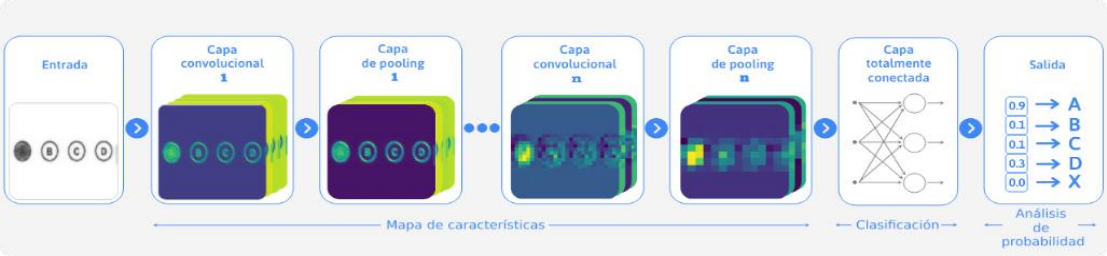

Figura 2 Capas de una CNN

Procesamiento digital de imágenes

El procesamiento digital de imágenes (PDI) es una disciplina que utiliza técnicas matemáticas y algorítmicas para analizar y manipular imágenes digitales. En el contexto de la evaluación de exámenes, el PDI ha sido ampliamente utilizado para identificar patrones y validar respuestas mediante:

- **Análisis de intensidad de píxeles:** Determinación de áreas sombreadas que representan respuestas seleccionadas.
- **Búsqueda de patrones:** Verificación de consistencia en las marcas hechas en los exámenes (Mona Peña, 2016).

Herramientas y tecnologías empleadas

El lenguaje de programación utilizado para desarrollar fue Python y TensorFlow como biblioteca de aprendizaje profundo. TensorFlow es una biblioteca de aprendizaje profundo de código abierto ampliamente utilizada para construir y entrenar redes neuronales artificiales. Proporciona herramientas para el diseño de modelos, manejo de datos y optimización de pérdidas, siendo clave para el desarrollo en el área de la inteligencia artificial (Abadi et al., 2016). Por otro lado, Python destaca como el lenguaje de programación preferido debido a su simplicidad y su amplia adopción en IA. Además, bibliotecas como NumPy, Matplotlib y OpenCV enriquecen el desarrollo de soluciones que integran redes neuronales convolucionales y procesamiento de imágenes (Van Rossum & Drake, 2009).

LabVIEW, usado en HABCO, es una plataforma de desarrollo basada en diagramas de bloques que facilita la implementación de procesamiento digital de imágenes para tareas específicas. Aunque tiene menor flexibilidad en comparación con TensorFlow, su accesibilidad para usuarios con conocimientos limitados en programación la convierte en una alternativa viable para soluciones de menor escala (Mona Peña, 2016).

Metodología

En esta sección se describe la metodología utilizada para evaluar los exámenes tipo alvéolo con la técnica de redes neuronales convolucionales. La clasificación de imágenes sigue una serie de pasos organizados en un flujo de trabajo metódico que abarca desde la preparación de los datos hasta la generación del modelo final. A continuación, se describe una metodología general que incluye el tratamiento de imágenes, el entrenamiento de la red y la generación del modelo. Este flujo de trabajo se muestra en **Figura 3**, donde se ilustran las etapas clave y su interrelación dentro del proceso.

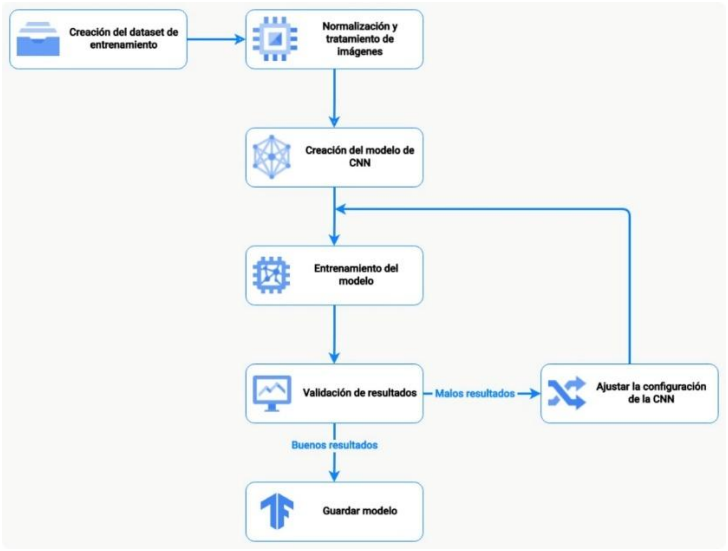

Figura 3 Flujo de trabajo

El código fuente desarrollado para este proyecto está disponible públicamente en el repositorio de GitHub: Aldoivan10/AI-NetEval. Este repositorio contiene todos los scripts necesarios para replicar el sistema de clasificación automática de respuestas, los cuales pueden ser ejecutados en cualquier editor de código compatible con Python en su versión *3.10*. Además, en caso de no contar con un equipo con las especificaciones óptimas para el entrenamiento de modelos de redes neuronales convolucionales (CNN), se recomienda utilizar plataformas gratuitas como Google Colab o Kaggle, las cuales ofrecen acceso a recursos computacionales, como GPUs (Unidad de Procesamiento Gráfico) y TPUs (Unidad de Procesamiento de Tensor), que facilitan el proceso de entrenamiento sin la necesidad de hardware especializado.

Creación del dataset

El primer paso es la creación del dataset (colección de datos) para el entrenamiento del modelo. Para lograrlo, se utilizó un diseño sencillo y que está dividido en varias secciones <<ver **Anexo A Figura 6**>>: en la parte superior se encuentra el encabezado con información del examen, incluyendo el número de versión del examen (opciones 1, 2 o 3 marcadas con círculos), y un código QR en la esquina superior derecha para la identificación de la persona a evaluar. Debajo del encabezado se ubica la sección principal etiquetada como "RESPUESTAS", que contiene la hoja de respuestas propiamente dicha. Sin embargo, es necesario centrarse en las respuestas, que están organizadas en columnas, y cada columna contiene filas con sus respectivos alveolos.

Este diseño inicial debe modificarse para dejar la parte de las respuestas lo más limpia posible. Con ello, se obtendrán mejores resultados en el entrenamiento de la CNN y será más fácil

identificar las respuestas. A continuación, los números de cada respuesta pasarán a estar fuera de su recuadro correspondiente, ya que estos serían un obstáculo, pues las únicas características relevantes deben ser los alveolos A-D. La adición de dichos números también sería una característica que el modelo tendría que identificar, lo cual no es deseado, ya que los números no son constantes y pueden variar en cada respuesta, lo que dificultaría la identificación precisa, además, se eliminó el borde inferior de cada fila. <<Como se muestra en el **Anexo A Figura 7**>>.

Con el examen rediseñado, se procede a imprimirlo y rellenarlo (**Figura 4¡Error! No se encuentra el origen de la referencia.**) con respuestas tanto correctas como incorrectas. A mayor número de exámenes contestados, se obtienen mejores resultados; es decir, mientras más datos de entrenamiento se utilicen, mayor es la precisión de la CNN.

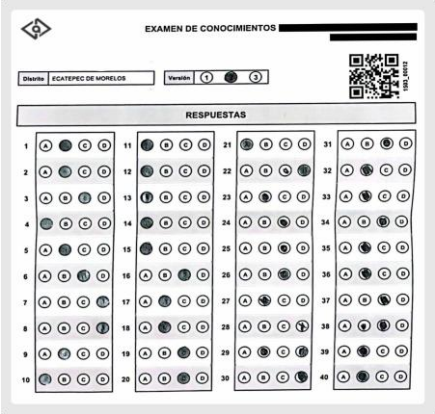

Figura 4 Muestra de examen contestado

Detección de polígonos

Con un conjunto de exámenes contestados útiles para comenzar a crear el set de datos, es necesario instalar algunas dependencias indispensables para la manipulación de imágenes: OpenCV e imutils. A continuación, se elaborará un script dataset.py para identificar los polígonos presentes en la imagen. Los pasos por seguir son:

1. Convertir la imagen a escala de grises y aplicar filtro de reducción de ruido (Línea 12).
2. Encontrar todos los polígonos dentro de la imagen, después, filtrar por los polígonos que tienen 4 lados y un alto mínimo (Línea 14).

Escala de grises

Para obtener las columnas que contienen los alveolos de las respuestas, es necesario llevar a cabo un proceso de tratamiento de imágenes. Esta parte es fundamental, ya que se utilizará este procedimiento más adelante para calificar los exámenes. El primer paso es identificar el espacio de color en el que se está trabajando. Existen varios espacios de color, entre ellos:

- **RGB (Red, Green, Blue) <<ver Anexo B>>:** Se refiere a una representación digital de una imagen donde cada píxel se define mediante tres valores (3 canales), correspondientes a la intensidad de los canales Rojo, Verde y Azul. Estos valores varían entre 0 y 255, permitiendo una combinación de hasta 16.7 millones de colores.
- **HSV (Hue, Saturation, Value) <<ver Anexo C>>:** Es un espacio de color que se aproxima más a cómo los humanos perciben el color, descompone el color en tres componentes (3 canales):

    - *Hue (Matíz):* Representa el color propiamente dicho y va desde 0 a 360 grados en la rueda de colores (por ejemplo, 0 es rojo, 120 es verde, 240 es azul).
    - *Saturation (Saturación):* Representa la intensidad o pureza del color, va de 0% (gris) a 100% (color puro).
    - *Value (Valor)*: Representa el brillo del color, va de 0% (negro) a 100% (color más brillante).
- **Escala de grises (grayscale):** Es un espacio de color que representa las imágenes en tonos de gris. A diferencia de RGB, donde cada píxel se define por tres valores de color, en *grayscale* cada píxel se define por un solo valor (1 canal), que varía de 0 (negro) a 255 (blanco). Los valores intermedios representan diferentes tonos de gris.

El primer paso es identificar el *espacio de color* de la imagen (generalmente RGB) para pasarlo a *escala de grises* (de ser necesario), ya que el color no es crucial para el análisis, ni representa una característica destacable en este proyecto, con ello se obtiene mayor simplicidad y eficacia, reduciendo la complejidad computacional. Mantener el espacio de color *RGB* o similares puede resultar ventajoso si el color ayuda a clasificar objetos, áreas, características faciales (como tono de piel, color de ojos), entre otros. La dependencia ***PIL*** cuenta con un método llamado Image.convert para cambiar entre espacios de color.

Una vez aplicado el cambio (aiutil.py línea 38), puede que los resultados no sean perceptibles a simple vista, dado que la imagen presenta tonalidades grisáceas. Sin embargo, el cambio es más evidente a nivel de código, ya que se pasa de tener 3 canales de color a únicamente 1 canal.

Filtro de desenfoque

El siguiente paso consiste en aplicar un filtro de desenfoque (aiutil.py línea 42) para reducir el ruido de la imagen. Existen varios tipos de filtros, como el filtro *gaussiano*, *bilateral*, *box*, y *median*, entre otros <<ver **D**>>. Se utilizará el filtro bilinear, ya que, además de reducir el ruido, no difumina los bordes como otros filtros. Esto es crucial, puesto que el siguiente paso es la detección de bordes. Otro filtro para considerar es el *Median Blur*, que tampoco difumina los bordes.

Para aplicar el *filtro bilateral*, *OpenCV* facilita el proceso, ya que la función ***bilateralFilter*** realiza el trabajo. Se pueden ajustar los parámetros para obtener mejores resultados (Filtro_Bilateral.png). Es importante tener en cuenta que filtros grandes ($d > 5$) son muy lentos, por lo que se recomienda utilizar $d = 5$ para aplicaciones en tiempo real, y quizás $d = 9$ para aplicaciones offline que requieran un filtrado intenso de ruido. Además, para simplificar los valores de sigma, ambos pueden establecerse como iguales. Si los valores son pequeños (< 10), el filtro tendrá poco efecto, mientras que si son grandes (> 150), el efecto será muy fuerte, haciendo que la imagen parezca *"caricaturesca"*.

Detección de bordes

Una vez aplicado el filtro para reducir el ruido, se procede a aplicar un filtro para la detección de bordes (aiutil.py línea 51). Existen varios filtros, cada uno con sus pros y contras. Si se busca rapidez sin importar tanto la precisión, el *filtro Sobel* es una buena opción. Por otro lado, si se desea la mayor precisión a costa de un mayor costo computacional, se puede optar por el *filtro Laplaciano de Gauss*. En este proyecto, se utilizó un punto intermedio, empleando el *filtro Canny*, que ofrece la precisión y robustez necesarias.

La dependencia *imutils* cuenta con la función ***auto_canny***, la cual aplica el filtro utilizando la mediana de las intensidades de los píxeles en escala de grises para derivar los umbrales superior e inferior (Filtro_Canny.png).

Detección de contornos

A continuación, es el turno de buscar los contornos (aiutil.py línea 53). *OpenCV* nuevamente facilita esta tarea con la función ***findContours***, que nos permite obtener los contornos como una lista de conjuntos de puntos y su jerarquía.

Para este proyecto se utilizará *RETR_EXTERNAL*, ya que no es necesaria ninguna jerarquía, y *CHAIN_APPROX_SIMPLE*, dado que es lo suficientemente precisa y eficiente para nuestro propósito.

Ya con los contornos encontrados, se filtran aquellos que cuenten únicamente con 4 lados. Al mismo tiempo, se identifican sus dimensiones para realizar el siguiente filtrado, el cual consiste en descartar los contornos que no cumplan con una altura mínima, calculada en el paso anterior, obteniendo así las columnas deseadas (aiutil.py línea 61).

Obtención de respuestas

Para obtener las imágenes que formarán parte del *dataset* de entrenamiento (dataset.py línea 26), se deben extraer las imágenes de las columnas. Posteriormente, cada imagen debe dividirse según la cantidad de respuestas que contiene (10 en este proyecto). Una vez realizada esa división, cada imagen debe almacenarse en una carpeta exclusiva para el entrenamiento y otra para pruebas, posteriormente deben clasificarse en subcarpetas que identifiquen a qué categoría pertenece cada imagen.

Para obtener las columnas como imágenes independientes, se hace uso de la función *four_point_transform*, la cual, dados 4 puntos, obtiene la imagen y la *"endereza"*, considerando que las imágenes pueden tener cierta inclinación debido a que se obtienen escaneando los exámenes y no siempre estarán perfectamente rectas.

Una vez que se obtienen las columnas, el siguiente paso es recortar la imagen de la columna para obtener las respuestas. Esto se hace utilizando una lista que contiene las coordenadas del punto inicial *(x0, y0)* y del punto final *(x1, y1)* de la sub-imágen deseada. En este proceso, *x0* siempre tendrá un valor constante de 0 para todas las sub-imágenes, ya que el punto de partida en el eje x es siempre 0. Por otro lado, *x1* será igual al ancho total de la imagen, ya que marca el punto final en dicho eje.

Para los valores de y, es necesario calcular el alto de cada fila en la columna. Esto se logra dividiendo la altura total de la imagen de la columna entre el número total de filas que contiene. El valor resultante, llamado *h*, corresponde a la altura de cada fila.

Con el valor de *h* calculado, *y0* se determina multiplicando *h* por el número de fila, comenzando desde el índice 0. Mientras que *y1* se calcula multiplicando *h* por el número de fila, comenzando desde el índice 1. De esta forma, se segmenta cada respuesta de la columna en imágenes individuales, listas para ser procesadas en los siguientes pasos del proyecto. Finalizando, se hace uso de la función *Image.crop* para dividir las columnas dados los valores calculados.

Con las respuestas obtenidas, es momento de crear el conjunto de datos. Para ello, en la carpeta donde se guardaron todas las imágenes se crean dos carpetas, una para entrenamiento y otra para las pruebas, dentro se generan subcarpetas con los nombres correspondientes a las distintas clasificaciones donde cada imagen debe ser ubicada en la subcarpeta que le corresponda. La estructura final debe ser similar a la mostrada en el repositorio (/images exceptuando la carpeta *article*), en la que las subcarpetas llevan los nombres de las posibles respuestas (A-D). Además, se debe incluir una carpeta adicional llamada *"X"*, que contendrá todas las respuestas consideradas incorrectas, según el criterio de quien realiza la clasificación.

Normalización y tratamiento de imágenes

Algunas de las imágenes obtenidas en el paso anterior pueden presentar líneas que pueden introducir ruido durante el entrenamiento. No obstante, las CNN suelen ser suficientemente robustas para ignorar la mayor parte de este ruido. Si se desea reducir o mitigar estos efectos, se puede aplicar código que genera un *"marco"* blanco alrededor de las imágenes obtenidas (dataset.py línea 30).

Otro aspecto importante por considerar es que estas imágenes son rectangulares, lo cual representa un desafío adicional. Esto no significa que no se puedan utilizar, pero requerirían ajustes en la estructura de la red, lo que podría hacerla más compleja y aumentar el costo computacional. Por esta razón, se redimensionarán las imágenes a un tamaño de *256x256* píxeles (dataset.py línea 32), ya que este tamaño ofrece un buen equilibrio entre detalle y eficiencia computacional. También se pueden considerar otros tamaños, dependiendo del tipo de proyecto y sus necesidades, manteniendo siempre un formato cuadrado.

Para ello, necesario añadir un *padding* a las imágenes, este puede aplicarse automáticamente al leer el conjunto de datos (como se detallará en el siguiente paso). Sin embargo, el padding automático se aplica en color negro, lo cual no es adecuado para este caso, ya que el fondo de las imágenes es de un tono blanquecino, mientras que las características (respuestas) son de un color más oscuro. Por lo tanto, el uso de un padding blanco es lo más recomendable para mantener la coherencia visual de las imágenes.

Creación del modelo

Para la creación del modelo, es importante tener en cuenta que las redes neuronales convolucionales presentan ciertas características en su estructura, que generalmente incluyen:

- *Input layer* (capa de entrada)
- *Convolutional layers* (capas convolucionales)
- *Pooling layers* (capas de agrupamiento)
- *Batch normalization* (normalización por lotes)
- *Flatten layer* (capa de aplanado)
- *Fully Connected Layers* (capas ompletamente conectadas)
- *Dropout* (descarte)
- *Output layer* (capa de salida)

Esta estructura ofrece varias ventajas: eficiencia, gracias a las capas de pooling y convolución, que optimizan el modelo en términos de computación y memoria; flexibilidad, ya que permite adaptar el modelo a distintos tamaños de imágenes y tareas; y reducción del sobreajuste mediante el uso de batch normalization y dropout. Esta estructura se puede modificar añadiendo más capas, diferentes tipos de capas, alternar entre funciones de activación, técnicas de aumento de datos, entre otras mejoras según la complejidad del problema.

A partir de esta estructura, se utilizó *Keras* para crear el modelo, que sigue una estructura muy similar a la descrita anteriormente. Es posible experimentar con los parámetros y modificar la estructura para optimizar los resultados, así como explorar otros parámetros adicionales que estas capas aceptan.

Para comenzar, se creó un modelo *Sequential*, disponible en *Keras*, que facilita enormemente la creación del modelo al permitir añadir las capas deseadas de forma secuencial, una tras otra (train.py línea 42).

La estructura propuesta es la siguiente:

1. ***Capa de entrada (Input):*** Define el tamaño y la forma de los datos de entrada. Esta capa sirve como un punto de entrada para los datos en el modelo. Dado que las imágenes usadas son de 256x256 y en escala de grises, se le estará pasando un valor de (256,256,1).
2. ***Aumento de datos (RandomTranslation, RandomRotation, RandomZoom, RandomBrightness y RandomContrast):*** Estas capas ayudan a ampliar nuestros datos de entrenamiento aplicando transformaciones con un desplazamiento específico. Es importante tener cuidado al utilizar técnicas como *RandomTranslation*, *RandomRotation* y *RandomZoom*, ya que, dependiendo del fill_mode, pueden producir resultados no deseados si se utiliza el valor por defecto *reflect*, aunque esto dependerá del proyecto. Al igual que en el paso de padding, queremos mantener un fondo blanco al aplicar estas transformaciones. Por lo tanto, se ha configurado el *fill_mode* en "constant" y se ha asignado un *fill_value* de 255.
3. ***Capa de normalización (Rescaling):*** Transforma los valores de las imágenes de 0-255 a un rango de 0-1, lo cual facilita que la red neuronal trabaje con valores más pequeños y homogéneos, optimizando tanto el aprendizaje como la convergencia. Un beneficio adicional es que, al incluir esta capa dentro del modelo, ya no es necesario realizar esta transformación externamente.
4. ***Capa convolucional (Conv2D):*** Aplica varias convoluciones 2D sobre la entrada, usando filtros de tamaño fijo, y genera un mapa de características (*feature map*). Cada filtro está diseñado para detectar un tipo específico de característica en la imagen. Ir aumentando gradualmente el número de filtros ayuda a que la red se enfoque en características más complejas y abstractas. Además, en este modelo se utilizó la función de activación *mish*, ya que ha demostrado ser efectiva para la clasificación de imágenes.
5. ***Normalización por lotes (BatchNormalization):*** Una capa opcional, pero útil para estabilizar y acelerar el entrenamiento. Ajusta y escala las activaciones.
6. ***Agrupación máxima (MaxPool2D):*** Reduce el tamaño de las representaciones espaciales (ancho y alto) de la entrada, lo que disminuye la cantidad de parámetros y el tiempo de cálculo en la red.
7. ***Capa de aplanado (Flatten):*** Es utilizada para transformar una entrada multidimensional, como la salida de una capa convolucional, en un vector unidimensional. Se coloca justo antes de las capas densas para que la salida de las capas convolucionales y de pooling se convierta en un formato adecuado para la clasificación o regresión.
8. ***Capas totalmente conectadas (Dense):*** Se realiza una transformación lineal de la entrada y se aplica una función de activación para introducir no linealidades en el modelo. Esto permite que la red aprenda patrones más complejos. La primera capa actúa como un extractor de características, mientras que la última capa convierte las salidas en una lista de probabilidades con valores entre 0 y 1, para ello se usa la función de activación softmax. Esto facilita la clasificación de las imágenes, por lo que el número de neuronas en esta capa es igual al número de clases posibles.
9. ***Capa de descarte (Dropout):*** Desactiva un porcentaje de las neuronas durante el entrenamiento, lo que reduce la co-adaptación de las neuronas. Esto significa que las neuronas no pueden confiar en la presencia de otras neuronas, lo que fomenta el

aprendizaje de características más generales. El parámetro dado es la proporción de neuronas que se desactivan en cada iteración de entrenamiento (por ejemplo, un valor de 0.5 significa que el 50% de las neuronas se desactivarán aleatoriamente).

Por último, se compila el modelo. Este paso es crucial, ya que reúne todos los elementos necesarios para el entrenamiento. Para ello, solo es necesario invocar la función *compile* en el objeto *Sequential* (nuestro modelo).

Entrenamiento de la red

Obtención del dataset

Una vez que se han clasificado los datos, es momento de crear un *dataset* a partir de ellos. Afortunadamente, TensorFlow facilita esta tarea gracias a *Keras*, que incluye la función *image_dataset_from_directory*, la cual cuenta con varios parámetros, de los cuales solo se modificaron algunos, mientras que los demás conservaron su valor por defecto. Esta función crea un dataset a partir de la carpeta raíz que contiene las imágenes y asigna automáticamente una etiqueta a cada dato según su clase correspondiente (train.py línea 12). Una vez obtenidos los *datasets*, para evitar que el modelo aprenda un orden específico, se barajan los datos con la función *shuffle*, especificando el tamaño del buffer para un mezclado eficiente.

Además, una forma de optimizar el rendimiento es utilizando las funciones *cache* y *prefetch*. La función cache permite almacenar los datos en memoria después de la primera lectura, evitando la recarga desde el disco en cada época y acelerando así el tiempo de entrenamiento. Por otro lado, prefetch prepara previamente los datos mientras el modelo entrena, y se usa *AUTOTUNE* para determinar automáticamente el tamaño óptimo del buffer, maximizando el rendimiento.

Entrenamiento

Con el modelo ya compilado obtenido del punto creación del modelo, y los *datasets* del paso anterior se usa la función *fit* que tiene el modelo (train.py línea 89), está función entrena al modelo de acuerdo con la configuración que se le pase por parámetros.

Resultados

Una vez iniciado el entrenamiento, se despliega información sobre el entrenamiento en cada época, como se muestra a continuación:

```
Epoch 1/50

20/20 [========] - 5s 85ms/step - loss: 1.6173 - accuracy: 0.1875 - val_loss: 1.6164 - val_accuracy: 0.2390

Epoch 2/50
```

```
20/20 [========] - 1s 39ms/step - loss: 1.6151 - accuracy: 0.1906 - val_loss: 1.6189 - val_accuracy: 0.1509

Epoch 3/50

20/20 [========] - 1s 40ms/step - loss: 1.6153 - accuracy: 0.1922 - val_loss: 1.6701 - val_accuracy: 0.1509

....
```

Estos datos ayudan a entender y ajustar el rendimiento del modelo a lo largo del entrenamiento, donde:

- **loss:** Es la función de pérdida medida en el conjunto de entrenamiento. Indica qué tan bien (o mal) está funcionando el modelo, con valores más bajos generalmente indicando mejor rendimiento.
- **accuracy:** La precisión en el conjunto de entrenamiento, que muestra el porcentaje de predicciones correctas del modelo.
- **val_loss:** Es la función de pérdida medida en el conjunto de validación. Sirve para monitorear si el modelo se está *sobreajustando*. Valores más bajos son mejores.
- **val_accuracy:** La precisión en el conjunto de validación, que indica qué tan bien está generalizando el modelo a datos no vistos.

Si los datos se han almacenado (train.py línea 91), pueden mostrarse de manera gráfica con la ayuda de la librería *matplotlib* para facilitar su interpretación. Al desplegar, se obtienen dos gráficos: uno para la pérdida y otro para la precisión del modelo, los cuales detallan su evolución como se muestra en la **Figura 5**.

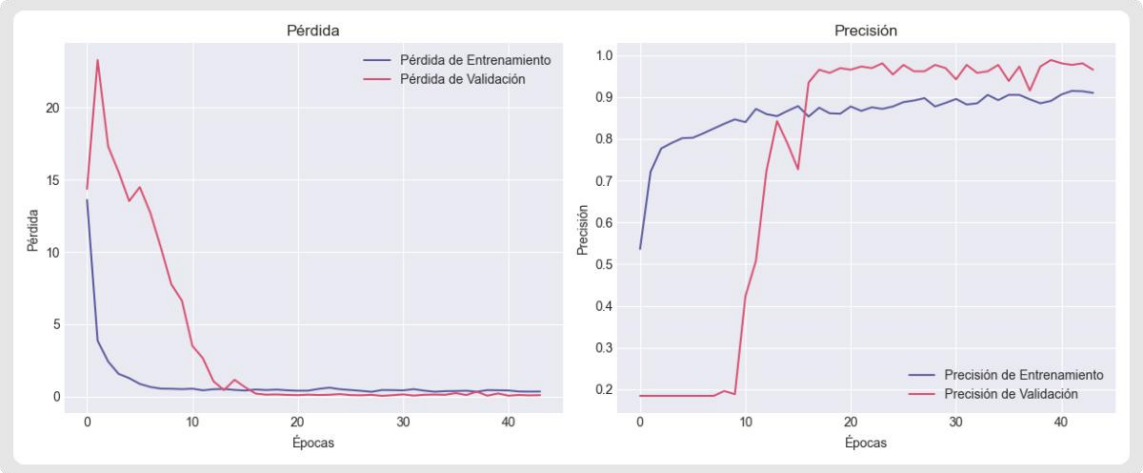

Figura 5 Entrenamiento con mish y datos aumentados

En las gráficas, se puede concluir que fue un entrenamiento exitoso con un buen equilibrio entre rendimiento en entrenamiento y validación, esto usando la función de activación *Mish* y el aumento de datos.

Para comprobar esto, se puede calcular los valores de **F1-score, Recall y Precisión**, las cuales son métricas de evaluación para modelos de clasificación, especialmente cuando se trata de

problemas multiclase o desbalanceados (Tabla 2). Los datos de esta evaluación se guardan dentro del archivo summary.txt.

Métrica	Evalúa	Fórmula	Valor obtenido	Interpretación
Recall	Representa la capacidad del modelo para encontrar todas las instancias positivas (clases correctas).	$F1 = \frac{TP}{TP + FN}$	0.9945	Un valor alto (cercano a 1) significa que el modelo está identificando casi todas las muestras correctamente, pero podría estar clasificando algunas erróneamente.
Precisión	Mide cuántas de las predicciones positivas del modelo realmente pertenecen a la clase correcta.	$P = \frac{TP}{TP + FP}$	0.9918	Un valor alto significa que cuando el modelo predice una clase, casi siempre es correcta.
F1-score	Es la media armónica entre Recall y Precisión, y equilibra ambos valores	$F1 = \frac{2 * TP}{2 * TP + FP + FN}$	0.9925	Un F1-score alto significa que el modelo tiene un buen balance entre recall y precisión.

Tabla 2 Métricas de evaluación

Además, se realizaron entrenamientos adicionales, alternando las funciones de activación *Mish sin aumento de datos* y *ReLU con aumento de datos y sin aumento de datos*. Los datos de los entrenamientos pueden ser encontrados en el mismo repositorio, dentro de la carpeta *models*.

Utilizando las imágenes preparadas para la prueba en la sección **Obtención de respuestas**, se comprobará el funcionamiento del modelo (test.py), obteniendo visualmente los resultados (Lote_#.png) donde en él Lote_4.png se puede apreciar que puede discernir correctamente la respuesta D aun con "ruido", sin embargo, en el Lote_3.png se aprecia que con más ruido la respuesta D no es clasificada de buena manera, lo cual coincide con la evaluación hecha al modelo generado.

Ajustar configuración

Las gráficas anteriores (**Figura 5¡Error! No se encuentra el origen de la referencia.**) brindan información valiosa para identificar los puntos débiles de nuestro modelo y, con ello, minimizar o incluso eliminarlos. Para ello, es necesario conocer dos conceptos importantes que ayudarán a interpretar los resultados obtenidos del entrenamiento:

- **Underfitting:** el modelo no logra aprender los patrones del entrenamiento; rinde mal en ambos conjuntos.
- **Overfitting:** el modelo aprende demasiado los datos de entrenamiento, incluyendo el ruido; pierde capacidad de generalización.

En las Tablas 3 y 4 se presentan algunos casos específicos para cada gráfica. Para el **gráfico de pérdida**:

Entrenamiento	Validación	Análisis	Notas
Curva que se va acercando a 0	Curva que se va acercando a 0	El modelo ha logrado aprender sin sobreajuste	No requiere modificaciones
Curva que desciende, pero se estabiliza	Curva que se mantiene alta	Indica subajuste (*underfitting*), el modelo no ha aprendido suficientemente	Aumentar la capacidad del modelo o entrenar más épocas
Curva desciende a 0	Curva desciende y luego sube	Sobreajuste (*overfitting*), el modelo memoriza los datos de entrenamiento	Aplicar regularización, dropout, o usar técnicas de aumento de datos
Curva inestable o ruidosa	Curva también inestable o errática	Problemas de optimización o datos mal preparados	Ajustar tasa de aprendizaje, normalizar datos o revisar outliers
Curva se mantiene alta	Curva se mantiene alta	El modelo no está aprendiendo	Revisar arquitectura, función de pérdida, o problemas en los datos

Tabla 3 Posibles resultados para el gráfico de pérdida

La tabla anterior clasifica los patrones de las curvas de pérdida, que miden el error o discrepancia entre las predicciones del modelo y los valores reales. Esta métrica es fundamental porque cuantifica qué tan equivocadas son las predicciones del modelo: valores cercanos a cero indican predicciones precisas, mientras que valores altos señalan predicciones deficientes. La importancia de analizar estos patrones radica en que la relación entre la pérdida de entrenamiento y validación revela la capacidad de generalización del modelo, es decir, si puede hacer predicciones acertadas con datos nuevos que nunca ha visto, o si simplemente ha memorizado los ejemplos de entrenamiento sin aprender los patrones subyacentes

Y para el gráfico de precisión:

Entrenamiento	Validación	Análisis	Notas
Curva que asciende y se estabiliza	Curva que asciende y se estabiliza	El modelo generaliza bien	No requiere modificaciones
Curva que asciende	Curva baja y plana	El modelo está sobreajustando	Usar técnicas de regularización o aumentar el conjunto de validación
Curva plana	Curva plana	Subajuste, la red no está aprendiendo patrones	Cambiar la arquitectura, optimizador o aumentar datos
Curva con oscilaciones fuertes	Curva también con oscilaciones	El entrenamiento es inestable	Reducir la tasa de aprendizaje o usar técnicas como batch normalization

Curva muy alta	Curva muy baja	Memoriza entrenamiento, pero no generaliza	Problema clásico de *overfitting*

Tabla 4 Posibles resultados para el gráfico de precisión

Esta tabla describe los patrones de las curvas de precisión, que miden el porcentaje de predicciones correctas que realiza el modelo sobre el total de predicciones. La precisión es importante porque ofrece una interpretación directa e intuitiva del desempeño del modelo: un 95% de precisión significa que, de cada 100 predicciones, 95 son correctas. Mientras que la pérdida indica la magnitud del error, la precisión responde a la pregunta práctica más relevante: *¿qué tan bien funciona el modelo en términos de aciertos?* Monitorear la divergencia entre la precisión de entrenamiento y validación es crucial porque revela si las mejoras en el conjunto de entrenamiento se traducen en un mejor desempeño real con datos no vistos, lo cual determina la utilidad práctica del modelo en aplicaciones del mundo real.

Guardar modelo

Guardar

Una vez que se ha conseguido un entrenamiento satisfactorio, existen diversas formas de guardar el modelo para su uso posterior. Una de ellas es mediante el uso del callback *ModelCheckpoint* (train.py línea 81), que guarda el mejor modelo durante el proceso de entrenamiento. Sin embargo, si se desea guardar el último modelo resultante del entrenamiento, es necesario invocar la función *save*, pasándole como parámetro la ruta y el nombre del archivo.

```
model.save("model.keras")
```

Cargar y usar el modelo

Una vez que el modelo ha sido guardado, se puede cargar y utilizar simulando un ambiente ya de uso como se muestra en el código de use.py, para fines prácticos se carga el modelo en cada análisis, sin embargo, el modelo puede ser fácilmente montado en un servidor para únicamente hacer peticiones, otra opción es crear una interfaz gráfica para poder cargar el modelo al inicio e ir seleccionando los exámenes a evaluar, las posibilidades son muchas y no se está sujeto a una sola.

Discusión

El presente estudio muestra la eficacia de las redes neuronales convolucionales (CNN) en la automatización de la evaluación de exámenes tipo alvéolo. El prototipo desarrollado logró calificar correctamente los exámenes analizados durante las pruebas de validación, con un tiempo de procesamiento menor o igual a 1 segundo por examen. Estos resultados superan significativamente los métodos tradicionales de revisión manual e incluso los métodos que hacen uso del procesamiento digital de imágenes (PDI), que requieren varios minutos por examen y llegan a presentar un mayor margen de error humano.

Se alcanzó una tasa de precisión del 99.18% y un recall del 99.45% en el conjunto de pruebas. Estas métricas fueron decisivas para que el Instituto Electoral del Estado de México optara por usar el modelo, ya que el tiempo disponible para la revisión era contado y se requería un método con alta confiabilidad. El prototipo cumplió exitosamente con su propósito de automatizar la

calificación de exámenes, demostrando que puede manejar eficientemente el volumen de evaluaciones requerido en procesos de selección masivos.

A pesar del éxito general, se identificaron algunas áreas problemáticas durante el proceso de validación. Como se evidenció en el Lote_3.png, las respuestas con alto nivel de interferencia visual (marcas adicionales, borrones o múltiples intentos de respuesta) no fueron clasificadas correctamente en algunos casos. La diversidad del conjunto de datos inicial fue relativamente reducida, lo que podría afectar el rendimiento con formatos de examen muy diferentes a los utilizados en el entrenamiento. Aunque el sistema maneja bien variaciones menores, escaneos de muy baja calidad o con distorsiones severas aún representan un desafío. Adicionalmente, situaciones donde el evaluado marcó débilmente una respuesta o hizo marcas en múltiples opciones requieren reglas de decisión adicionales que el modelo actual no contempla completamente.

El desarrollo del prototipo proporcionó varias lecciones importantes sobre la implementación de redes neuronales convolucionales en contextos educativos reales. El preprocesamiento adecuado de las imágenes antes de alimentarlas al modelo resultó fundamental para el éxito del sistema. La detección precisa de contornos y la normalización fueron aspectos críticos en esta etapa. Se confirmó que una arquitectura de CNN relativamente sencilla, bien configurada y entrenada puede superar a sistemas más complejos si se ajusta correctamente a las características específicas del problema. Incluso con un dataset inicial modesto, las técnicas de aumento de datos permitieron crear suficiente variabilidad para entrenar un modelo robusto. La elección de la función de activación demostró tener un impacto significativo en el rendimiento final, con Mish mostrando ventajas sobre ReLU en este contexto específico de clasificación de imágenes con características sutiles.

Para mejorar aún más el sistema, se identificaron varias áreas de desarrollo futuro. Es necesario ampliar el dataset para incluir mayor diversidad de formatos de examen, calidades de digitalización y estilos de marcado, lo que mejoraría la generalización del modelo. La implementación de lógica adicional para detectar y reportar respuestas ambiguas (múltiples marcas, marcas débiles) en lugar de forzar una clasificación sería una mejora valiosa. Explorar técnicas de compresión de modelos (quantization, pruning) permitiría la ejecución en hardware menos potente sin pérdida significativa de precisión. La adición de un mecanismo que reporte el nivel de confianza en cada clasificación facilitaría la revisión manual de casos dudosos. El desarrollo de una aplicación de escritorio o web más robusta que facilite la carga de exámenes, visualización de resultados y corrección de errores de manera intuitiva mejoraría la experiencia del usuario final. Finalmente, probar el sistema con exámenes de diferentes instituciones y formatos permitiría evaluar su portabilidad y necesidades de adaptación.

Dentro de las limitaciones identificadas están la cantidad y diversidad del dataset utilizado, que podría afectar la generalización a otros formatos de exámenes, así como la presencia de ruido excesivo en algunas imágenes que afectó predicciones específicas. El uso de hardware especializado (GPUs) puede representar una barrera para instituciones con recursos limitados, aunque las notebooks en línea (Google Colab, Kaggle) ofrecen una alternativa accesible y viable.

Los resultados de este estudio pueden aplicarse en entornos educativos y organizacionales para optimizar la calificación de exámenes, reduciendo la carga de trabajo manual. Se recomienda su implementación en plataformas de evaluación automatizada y su integración con sistemas de gestión educativa. El código desarrollado está disponible públicamente y puede servir como base para futuras mejoras y adaptaciones según necesidades específicas.

Conclusiones

Basado en los hallazgos se encontró que las redes neuronales convolucionales representan una solución viable y efectiva para la automatización de procesos de evaluación educativa, específicamente en la calificación de exámenes tipo alvéolo. La implementación exitosa del sistema en el Instituto Electoral del Estado de México valida la aplicabilidad práctica de estas tecnologías en contextos reales donde la eficiencia, precisión y confiabilidad son requisitos indispensables.

La investigación confirma que, contrario a la percepción común de que las soluciones basadas en aprendizaje profundo requieren infraestructuras computacionales robustas, es posible desarrollar e implementar sistemas de CNN efectivos utilizando recursos gratuitos y de acceso público. Las plataformas como Google Colab y Kaggle democratizan el acceso a estas tecnologías, eliminando barreras económicas significativas para instituciones educativas y organizaciones con presupuestos limitados.

Un hallazgo fundamental es que la calidad y diversidad del conjunto de datos tienen un impacto más significativo en el rendimiento del modelo que la complejidad arquitectónica de la red neuronal. Una CNN relativamente sencilla, correctamente configurada y entrenada con datos bien preparados, puede superar a arquitecturas más complejas con datos insuficientes. Esto subraya la importancia de invertir esfuerzos en la recolección, curación y preprocesamiento de datos por encima de la búsqueda de arquitecturas cada vez más sofisticadas.

La comparación realizada entre el procesamiento digital de imágenes tradicional y las redes neuronales convolucionales revela que, si bien el PDI mantiene relevancia en aplicaciones con requisitos específicos y entornos altamente controlados, las CNN ofrecen ventajas superiores en términos de adaptabilidad, generalización y escalabilidad. Esta transición tecnológica no implica el reemplazo absoluto de métodos tradicionales, sino la selección estratégica de la herramienta más apropiada según el contexto específico del problema.

El código abierto y la documentación detallada proporcionados en este proyecto aspiran a servir como recurso educativo y punto de partida para futuros desarrollos en el campo de la evaluación automatizada. La metodología presentada es transferible a otros contextos educativos y tipos de evaluación, facilitando la replicación y adaptación del sistema según necesidades particulares.

Las limitaciones identificadas, particularmente en el manejo de respuestas con alto nivel de interferencia visual y la necesidad de datasets más diversos, representan oportunidades concretas de mejora y líneas de investigación futuras. El desarrollo de mecanismos de detección de ambigüedad y sistemas de reportería de confianza constituyen extensiones naturales que fortalecerían la robustez del sistema.

Finalmente, este trabajo contribuye al campo emergente de la inteligencia artificial aplicada a la educación, demostrando que la automatización inteligente puede liberar recursos humanos de tareas repetitivas hacia actividades de mayor valor. La reducción del tiempo de calificación de varios minutos a menos de un segundo por examen, manteniendo niveles de precisión aceptables, representa un avance significativo.

La implementación exitosa del sistema valida la hipótesis inicial de que las CNN pueden superar las limitaciones del procesamiento digital de imágenes tradicional en tareas de clasificación de respuestas, abriendo caminos para futuras aplicaciones en evaluación educativa automatizada, análisis de documentos y procesamiento de formularios en diversos sectores organizacionales.

Referencias

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 265-283.

Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

Chen, J., Qi, X., Wang, W., Li, B., & Liu, Y. (2020). Real-time location of surgical incisions in cataract phacoemulsification. Springer Science+Business Media, LLC, part of Springer Nature. https://doi.org/10.1007/s0010-2020

Chollet, F. (2017). Deep learning with Python. Manning Publications Co.

Cordón Luis, S. (2019). Reconocimiento de torres de alta tensión mediante algoritmos de visión por computador e inteligencia artificial: LR, SVM, CNN.

De Carvalho, A. S. A., Pereira, E. C. D. S., Da Silva, E. J., & Piva Jr, D. (2022). Análise comparativa entre os principais algoritmos de detecção facial: Haar Cascade, HOG, CNN, YOLO e DeepFace. In OPEN SCIENCE RESEARCH V (Vol. 5, pp. 439-454). Editora Científica Digital.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics, 249-256.

Gómez Pujante, B. (2023). Redes convolucionales. Aplicación a la clasificación de imágenes médicas [Trabajo de fin de grado, Universidad Miguel Hernández de Elche]. Universidad Miguel Hernández de Elche. https://hdl.handle.net/11000/30233

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Haykin, S. (2009). Neural networks and learning machines (Vol. 3). Pearson Education.

Heaton, J., Goodfellow, I., Bengio, Y., & Courville, A. (2018). Deep learning. Genetic Programming and Evolvable Machines, 19(3-4), 305–307. https://doi.org/10.1007/s10710-017-9314-z

Kanabur, V., Harakannanavar, S. S., Purnikmath, V. I., Hullole, P., & Torse, D. (2020). Detection of leaf disease using hybrid feature extraction techniques and CNN classifier. In Computational Vision and Bio-Inspired Computing: ICCVBIC 2019 (pp. 1213-1220). Springer International Publishing.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *arX

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems (Vol. 25). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015).

Maas, A.L., Hannun, A.Y. and Ng, A.Y. (2013) Rectifier Nonlinearities Improve Neural Network Acoustic Models. Proceedings of the 30th International Conference on Machine Learning, Vol. 28, 3. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th international conference on machine learning (ICML-10), 807-814.

Nama, P. (2022). Optimizing automation systems with AI: A study on enhancing workflow efficiency through intelligent decision-making algorithms. World Journal of Advanced Engineering Technology and Sciences, 7(02), 296-307.

Pardo, Á. (s.f.). Filtrado de imágenes. Departamento de Ingeniería Eléctrica, Universidad Católica del Uruguay. Recuperado de https://grupivis.dc.uba.ar/archivos/UBA_FiltradoImagenes.pdf

Peña, L. J. M. (2016). HABCO: Herramienta informática para la automatización de la calificación de exámenes para apoyo al docente. Res. Comput. Sci., 111, 9-21.

Ramos Armijos , D. F., Ramos Armijos , D. G., Ramos Armijos , N. J., Tapia Puga , V. M., & Tapia Puga , L. I. (2024). Explorando las Fronteras: la Aplicación de Inteligencia Artificial en la Evaluación Educativa. Ciencia Latina Revista Científica Multidisciplinar, 7(6), 5657-5672. https://doi.org/10.37811/cl_rcm.v7i6.9108

Russell, S. J., & Norvig, P. (2004). Inteligencia artificial: Un enfoque moderno (2ª ed.). Pearson Educación.

Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0

Van Rossum, G., & Drake, F. L. (2009). Introduction to python 3: python documentation manual part 1. CreateSpace.

Anexos

Anexo A Exámenes

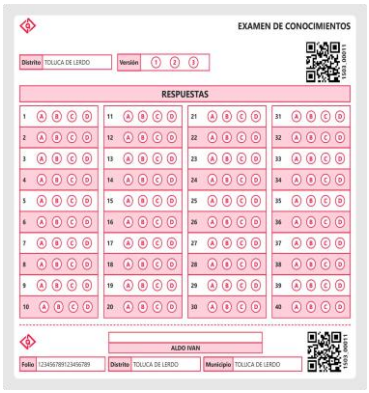

Figura 6 Examen de referencia

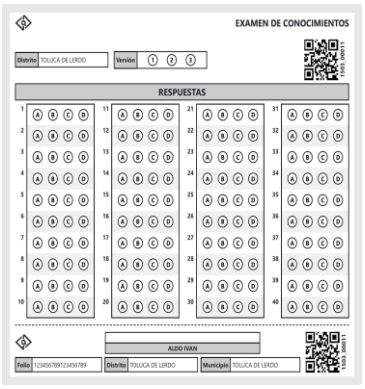

Figura 7 Examen rediseñado

Anexo B Espacio de color RGB

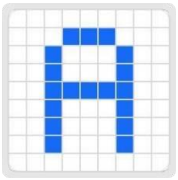

Figura 7 Imagen RGB

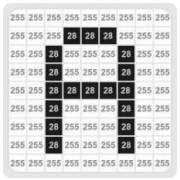

Figura 8 Canal R

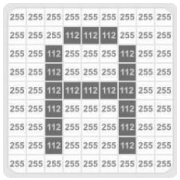

Figura 9 Canal G

Figura 10 Canal B

Anexo C Espacio de color HSV

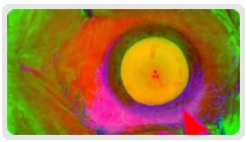

Figura 11 Imagen HSV

Figura 12 Canal H

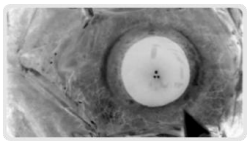

Figura 13 Canal S

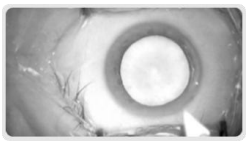

Figura 14 Canal V

Anexo D Filtros de desenfoque

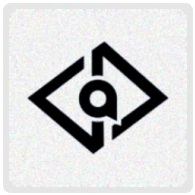

Figura 15 Sin filtro

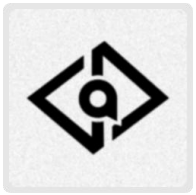

Figura 16 Filtro Gaussiano

Figura 17 Filtro Bilateral

Anexo E Modos de relleno

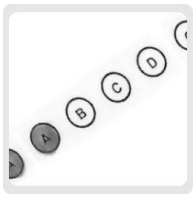

Figura 18 Modo reflect

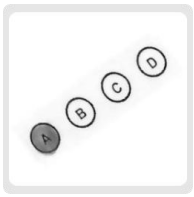

Figura 19 Modo constant

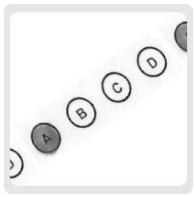

Figura 20 Modo wrap

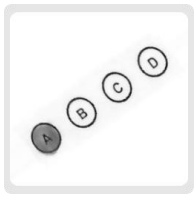

Figura 21 Modo nearest