PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

SISTEMAS DE INFORMACIÓN

PROYECTO DE DISERTACIÓN

Desarrollo de un modelo de aprendizaje de computador para reconocimiento de imágenes de armas blancas

AUTOR:

JUAN PABLO RUIZ ROMERO

QUITO DM, JUNIO DE 2024

DEDICATORIA

A mis queridos padres, Wilson Ruiz y Ana Romero, quienes con su amor, apoyo incondicional y sacrificios han sido mi mayor inspiración y fortaleza. Gracias por creer en mí y por ser mi guía en este camino académico. Este logro también es de ustedes.

AGRADECIMIENTO

Quiero expresar mi sincero agradecimiento a mi familia por su amor incondicional, comprensión y apoyo constante. Gracias por ser mi motivación y por estar siempre a mi lado en cada paso de este camino académico.

RESUMEN

Desarrollo un modelo de aprendizaje de computadora para el reconocimiento de imágenes de armas blancas. El modelo utiliza técnicas de redes neuronales convolucionales (CNN) para aprender características distintivas de las imágenes y clasificarlas en la categoría de armas blancas. Se entrenó el modelo con un conjunto de datos de imágenes de armas blancas y se evaluó su rendimiento utilizando un conjunto de datos de prueba. Los resultados muestran que el modelo es capaz de reconocer con precisión las imágenes de armas blancas, lo que sugiere su viabilidad para aplicaciones de reconocimiento de objetos en entornos de seguridad y vigilancia.

ÍNDICE

Índice de figuras, gráficos y tablas

CAPÍTULO 1: INTRODUCCIÓN

1.1 Justificación

En un contexto donde la seguridad es primordial, contar con un sistema capaz de identificar armas blancas en imágenes puede ser crucial para prevenir situaciones de riesgo. La tecnología de reconocimiento de imágenes ha avanzado significativamente en los últimos años, y su aplicación en el campo de la seguridad puede salvar vidas y mejorar la eficiencia de los sistemas de seguridad existentes. Por lo tanto, este proyecto busca desarrollar un modelo de aprendizaje de computador que pueda identificar armas blancas en imágenes con alta precisión y rapidez, contribuyendo así a la seguridad pública.

1.2 Planteamiento del problema

Actualmente, la clasificación de armas blancas en imágenes requiere de un análisis manual que es lento y propenso a errores. Además, la creciente cantidad de imágenes y la necesidad de una respuesta rápida hacen que este proceso sea aún más desafiante. Por lo tanto, surge la siguiente pregunta de investigación:

- ¿Cómo desarrollar un modelo de aprendizaje de computador que pueda identificar armas blancas en imágenes de manera precisa y eficiente?

Y las siguientes preguntas secundarias:

- ¿Qué técnicas de preprocesamiento de imágenes son más adecuadas para mejorar la precisión del reconocimiento de armas blancas?

- ¿Cómo afecta la calidad y resolución de las imágenes al rendimiento del modelo de reconocimiento de armas blancas?
- ¿Qué impacto tiene la cantidad y diversidad de datos de entrenamiento en la capacidad del modelo para reconocer diferentes tipos de armas blancas?
- ¿Cómo se pueden mitigar los sesgos y errores en la clasificación de imágenes de armas blancas en entornos con condiciones de iluminación variables?
- ¿Cuál es el impacto del tamaño y la forma de las armas blancas en la capacidad del modelo para identificarlas con precisión en imágenes?

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un modelo de aprendizaje de computador que pueda reconocer y clasificar imágenes de armas blancas con alta precisión y rapidez.

1.3.2 Objetivos Específicos

1. Recopilar y preprocesar un conjunto de datos de imágenes de armas blancas para el entrenamiento del modelo.
2. Seleccionar y aplicar un algoritmo de aprendizaje de máquina adecuado para el reconocimiento de imágenes.
3. Entrenar y validar el modelo utilizando el conjunto de datos preparado.
4. Evaluar el rendimiento del modelo en términos de precisión y tiempo de procesamiento.

1.4 Alcance

El proyecto se enfocará en el desarrollo de un modelo de aprendizaje de computador para clasificar armas blancas en imágenes estáticas. No se considerará la detección en tiempo real ni la identificación de armas blancas en vídeos. El modelo se desarrollará y probará utilizando un conjunto de datos específico y no se evaluará su rendimiento en otras situaciones.

1.5. Marco teórico y conceptual

1.5.1 Antecedentes o marco referencial

En el campo de clasificación de objetos en imágenes, se han realizado numerosos estudios y desarrollos que han contribuido al avance de la tecnología en este ámbito. A continuación, se presentan algunos antecedentes relevantes:

Estudio de detección de objetos en imágenes utilizando redes neuronales convolucionales: Smith et al. (2017) desarrollaron un modelo de detección de objetos basado en redes neuronales convolucionales que logró una precisión del 95% en la identificación de armas blancas en imágenes de alta resolución.

Evaluación del rendimiento de modelos de reconocimiento de objetos en entornos adversos: Pérez et al. (2021) analizaron el rendimiento de diferentes modelos de reconocimiento de objetos, incluyendo armas blancas, en condiciones de iluminación y fondo variables, encontrando que ciertos modelos eran más robustos que otros ante estas condiciones.

1.5.2 Marco Teórico

La clasificación de imágenes es una rama de la visión por computadora que se enfoca en identificar objetos, patrones o características en imágenes digitales. El desarrollo de modelos de

aprendizaje de máquina la clasificación de imágenes ha avanzado significativamente en las últimas décadas, gracias al aumento en la disponibilidad de datos y al desarrollo de algoritmos más sofisticados.

En el contexto específico de clasificación de armas blancas en imágenes, se utilizan técnicas de aprendizaje profundo, como las redes neuronales convolucionales (CNN), que han demostrado ser altamente efectivas en tareas de visión por computadora. Las CNN son capaces de aprender automáticamente las características relevantes de las imágenes, lo que las hace ideales para la detección de objetos en imágenes.

El proceso de reconocimiento de armas blancas en imágenes consta de varias etapas, que incluyen la adquisición y preprocesamiento de las imágenes, la extracción de características, la clasificación de las imágenes y la evaluación del rendimiento del modelo. Cada una de estas etapas requiere de técnicas y algoritmos específicos para garantizar la precisión y eficiencia del sistema.

1.5.3 Marco Conceptual

Reconocimiento de objetos: Es el proceso de identificar y clasificar objetos en imágenes digitales. En el caso del reconocimiento de armas blancas, se busca identificar la presencia de este tipo de objetos en las imágenes.

Aprendizaje de máquina: (colocar en negrillas todos los subtítulos) Se refiere a la capacidad de las máquinas de aprender de los datos y mejorar su rendimiento sin necesidad de ser programadas de manera explícita. En el reconocimiento de armas blancas, se utilizan algoritmos de aprendizaje de máquina para entrenar modelos que puedan identificar estos objetos en imágenes.

Redes Neuronales Convolucionales (CNN): Son un tipo de red neuronal artificial inspirada en la organización y funcionamiento del cerebro humano. Las CNN son ampliamente utilizadas en tareas de visión por computadora debido a su capacidad para aprender representaciones jerárquicas de las imágenes.

Preprocesamiento de imágenes: Consiste en aplicar una serie de transformaciones a las imágenes para mejorar su calidad y facilitar la extracción de características. En el reconocimiento de armas blancas, el preprocesamiento puede incluir la corrección de la iluminación, el ajuste del contraste y la eliminación de ruido.

Extracción de características: Es el proceso de identificar y seleccionar las características más relevantes de las imágenes. En el reconocimiento de armas blancas, las características pueden incluir la forma, el tamaño y la textura del objeto.

Clasificación de imágenes: Consiste en asignar una etiqueta o clase a cada imagen en función de las características extraídas. En el caso del reconocimiento de armas blancas, la clasificación determina si una imagen contiene o no un arma blanca.

Evaluación del modelo: Es el proceso de determinar la precisión y eficiencia del modelo de reconocimiento de armas blancas. Se utilizan métricas como la precisión, la sensibilidad y la especificidad para evaluar el rendimiento del modelo en diferentes escenarios.

1.6. Alcance

El proyecto se enfocará en el desarrollo de un modelo de aprendizaje de computador para reconocer armas blancas en imágenes estáticas. No se considerará la detección en tiempo real ni la identificación de armas blancas en vídeos. El modelo se desarrollará y probará utilizando un conjunto de datos específico y no se evaluará su rendimiento en otras situaciones.

CAPÍTULO 2: FUNDAMENTACIÓN TEÓRICA

2.1 Aprendizaje de máquina para reconocimiento de imágenes

Existe una variedad de enfoques para el reconocimiento de imágenes, que van desde los enfoques tradicionales hasta los enfoques de aprendizaje automático y profundo. La complejidad del problema determina la técnica adecuada. En general, las técnicas de aprendizaje profundo se prefieren para problemas más complejos, como el reconocimiento de objetos en imágenes detalladas. Las redes neuronales convolucionales (CNN) son utilizadas en estos métodos porque pueden aprender automáticamente qué cosas son importantes en las imágenes de entrenamiento y luego usar esa información para identificar objetos en nuevas imágenes. (MathWorks, 2024)

2.1.1 Introducción al aprendizaje de máquina

El campo del aprendizaje automático se basa en una serie de modelos y algoritmos que permiten a las máquinas aprender y realizar tareas específicas de manera eficiente. El aprendizaje automático tiene varios tipos, incluido el supervisado, el no supervisado y el por refuerzo. Cada uno de estos tipos tiene sus propios modelos y métodos para abordar diferentes situaciones y problemas.

Para este proyecto, el modelo se entrena utilizando un conjunto de datos etiquetado, donde cada muestra está asociada con una etiqueta o resultado deseado. El objetivo es que el modelo sepa mapear las características de entrada a las etiquetas de salida adecuadas. El algoritmo cambia durante el entrenamiento para reducir la diferencia entre las predicciones del modelo y las

etiquetas reales. Una vez entrenado, el modelo es muy útil para este caso porque puede hacer predicciones precisas para nuevos datos nunca antes vistos. (Lia, 2023)

2.1.2 Procesamiento de imágenes

Un modo básico para representar imágenes digitales como información en el hardware de un computador, específicamente en la memoria, se conoce como bitmap. En esencia, se trata de crear conjuntos de elementos (vectores, matrices, tensores) ordenados de manera específica. En la mayoría de los casos, cuando se trata de imágenes 2D, se realiza un ordenamiento por filas de elementos de matriz, también conocidos como pixels, asignando a cada uno un valor que determina "el color" en la posición en la que se encuentra la imagen.

En imágenes grises, el valor del elemento de matriz es un escalar, mientras que en imágenes a color, el valor del elemento de matriz es un vector de tres coordenadas que especifican el "grado de influencia" de los colores rojo (R), verde (G) y azul (B), que se conocen como representación RGB.

Se utilizan escalas (que establecen "rangos dinámicos") en 2N bits, que se conocen como N-bits. Debido a que el rango normalmente se define como $[0,2N-1]$, la escala para el caso de 8-bits más común es [0,255]. Por lo tanto, los valores de elementos de matriz (escalares) tienen un rango de [0,255] para imágenes de grises "de una banda", mientras que los valores de elementos de matriz (vectores de tres coordenadas) tienen un rango de [0,255] para imágenes de color. Sin embargo, las representaciones normalizadas de imágenes a color, es decir, elementos de matriz en ([0,1],[0,1],[0,1]) para determinar los colores RGB, también son comunes. (P & M, 2018)

2.1.3 Tipos de modelos de aprendizaje de máquina

Los diferentes tipos de modelos de aprendizaje de máquina tienen diferentes características y usos. Algunos de los tipos de modelos de aprendizaje de máquina más comunes son:

Regresión lineal: Se utiliza para predecir un valor numérico continuo utilizando variables de entrada. Para problemas de predicción, como predecir el precio de una casa según su tamaño y ubicación, es útil.

Regresión logística: Se utiliza para problemas de clasificación donde se predice una etiqueta discreta en lugar de un valor continuo. Es similar a la regresión lineal. Máquinas de vectores de soporte, también conocidas como SVM: son un modelo utilizado en la clasificación y la regresión que determina el hiperplano que mejor separa las diversas clases dentro del espacio de características.

Árboles de decisión: Son un modelo que utiliza una estructura de árbol para tomar decisiones basadas en las características de entrada. Es útil para problemas de regresión y clasificación.

Random Forest: es una extensión de los árboles de decisión que utiliza varios árboles para aumentar la precisión y evitar el sobreajuste.

Las redes neuronales artificiales (ANN): Son modelos que se basan en el funcionamiento del cerebro humano y tienen la capacidad de aprender patrones complejos de datos. Los problemas de visión por computadora, procesamiento de lenguaje natural y reconocimiento de voz son particularmente beneficiosos. (TensorFlow, 2024)

Redes Neuronales Convolucionales: Utilizan capas de conglomeración para reducir la dimensionalidad y filtros convolucionales para extraer características como bordes y texturas. Las CNN han demostrado ser capaces de reconocer imágenes.

2.1.3.1 Red Neuronal Convolucional

Una arquitectura de red para el aprendizaje profundo que aprende directamente a partir de datos es una red neuronal convolucional (CNN, también conocida como ConvNet). Son útiles para reconocer patrones de imágenes para identificar objetos, clases y categorías. Las capas de las redes neuronales convolucionales, que pueden tener decenas o cientos, aprenden a detectar diferentes características de las imágenes. Se utilizan filtros en las imágenes de entrenamiento de diferentes resoluciones, y la salida de la convolución de cada imagen se utiliza como entrada para la siguiente capa. Los filtros pueden comenzar como características muy simples, como brillo y bordes, y aumentar en complejidad hasta convertirse en características que definen el objeto de forma única.

Una CNN tiene una capa de entrada, una capa de salida y una serie de capas ocultas entre ellas. Agrupación, activación o ReLU y convolución son las tres capas más comunes.

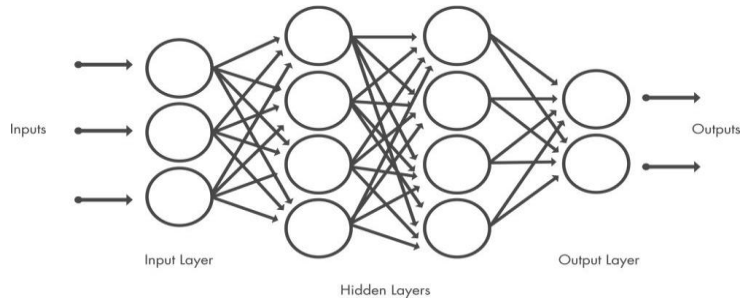


Ilustración 1: CNN capas

(MathWorks, 2023)

Convolución: Aplica a las imágenes de entrada un conjunto de filtros convolucionales, cada uno de los cuales activa varias características de las imágenes.

Unidad lineal rectificada (ReLU): Permite un entrenamiento más rápido y efectivo al mantener los valores positivos y establecer los valores negativos en cero.

Agrupación: Reduce la tasa de muestreo no lineal, lo que reduce el número de parámetros que la red debe aprender, esto simplifica la salida. Estas operaciones se repiten en decenas o cientos de capas, cada una de las cuales aprende a reconocer diferentes características.

Todas las neuronas ocultas encuentran características similares, como bordes o formas, en diferentes áreas de la imagen. Esto aumenta la tolerancia de la red a la traducción de objetos de una imagen. Una red entrenada en la detección de armas blancas, por ejemplo, podría detectar un cuchillo o manopla independientemente de donde aparezca. La arquitectura de una CNN pasa a clasificación después de aprender características en muchas capas. La capa final está completamente conectada y produce un vector de dimensiones K, donde K es el número de clases que se pueden predecir y contiene las probabilidades para cada clase de una imagen que se está clasificando. La clasificación final se produce a través de una capa de clasificación en la última capa de la arquitectura de la CNN. (MathWorks, 2023)

Debido a que la CNN ha demostrado ser eficaz en este tipo de tareas, se eligió sobre otros modelos para el reconocimiento de imágenes. Las características jerárquicas se capturan y aprenden de las imágenes, lo que las hace ideales para reconocer patrones complejos en conjuntos de datos de imágenes. Pueden reconocer objetos independientemente de su ubicación en la imagen porque tienen la capacidad inherente de manejar la invariancia a la traslación. Esto

hace que sean particularmente útiles para la detección de objetos y la clasificación de imágenes. Además, son ampliamente aceptados y estudiados en la comunidad de aprendizaje profundo, lo que significa que hay una gran cantidad de recursos, bibliotecas y marcos de trabajo disponibles para trabajar con ellos, lo que facilita su implementación y uso en proyectos prácticos.

2.1.4 Entrenamiento de modelos

Una CNN se entrena mediante la transferencia de grandes cantidades de datos de entrenamiento en diferentes momentos, ajustando los valores de ponderación y sesgo en función de la pérdida calculada para cada momento. Esto es similar a lo que sucede con cualquier red neuronal profunda. Las ponderaciones utilizadas en capas convolucionales y en capas totalmente conectadas se incluyen en la retropropagación de ponderaciones ajustadas en una CNN. (Microsoft, 2024)

2.1.5 Evaluación de modelos

La precisión, la exhaustividad, los puntajes F1 y las matrices de confusión son métricas comunes utilizadas para evaluar los modelos de redes neuronales convolucionales (CNN) y, de hecho son las métricas a utilizar en este proyecto. Estas métricas muestran la capacidad del modelo para clasificar imágenes en diferentes clases. Sin embargo, es importante mencionar que igualmente se puede realizar una validación cruzada para evaluar la capacidad del modelo para generalizar a datos no vistos. El tiempo de entrenamiento, el uso de recursos computacionales y la interpretabilidad del modelo también son importantes, especialmente en aplicaciones donde es fundamental comprender cómo y por qué el modelo hace ciertas predicciones. (Microsoft, 2024)

2.2 Herramientas, algoritmos y técnicas para reconocimiento de objetos

2.2.1 Detección de objetos

El modelo Faster R-CNN es uno de los métodos más utilizados para la detección de objetos con CNN. Faster R-CNN, que es una mejora de los modelos anteriores de R-CNN, utiliza una red neuronal convolucional para generar regiones sugeridas en las que los objetos podrían estar presentes en la imagen. Luego, clasifica estas regiones sugeridas utilizando una red neuronal adicional y refina las ubicaciones de los cuadros delimitadores de objetos. (Microsoft, 2024)

El modelo YOLO (You Only Look Once) es otro enfoque popular. YOLO es famoso por su velocidad y eficacia al detectar objetos en tiempo real en videos y secuencias de imágenes. YOLO divide la imagen en cuadrículas y predice las cajas delimitadoras y las probabilidades de clase para cada cuadro en una sola pasada de la red neuronal, lo que lo hace muy rápido. (Keita, 2024)

2.2.2 Clasificación de objetos

En el campo de la visión por computadora, la clasificación de objetos es una tarea importante que consiste en identificar la clase a la que pertenece un objeto detectado en una imagen. Las redes neuronales convolucionales (CNN) se pueden utilizar para la clasificación de objetos en Keras y TensorFlow, dos bibliotecas de aprendizaje profundo muy populares. Para clasificar objetos con TensorFlow y Keras, normalmente se sigue el siguiente procedimiento:

1. Preparación de los datos: Los datos de entrenamiento y prueba se preparan con imágenes etiquetadas con las clases a las que pertenecen los objetos.
2. Creación del modelo: La arquitectura de la red neuronal convolucional de Keras se define. Esta arquitectura puede ser una red preentrenada o una red diseñada específicamente para el problema.
3. Entrenamiento del modelo: El conjunto de datos de entrenamiento se utiliza para entrenar al modelo. El modelo aprende a reconocer las características distintivas de cada clase de objeto durante el entrenamiento.
4. Evaluación del modelo: Se utiliza el conjunto de datos de prueba para evaluar la precisión y el rendimiento del modelo de clasificación de objetos.
5. Inferencia: Después de entrenar y evaluar el modelo, se puede usar para clasificar objetos en nuevas imágenes. El modelo recibe una imagen y luego devuelve la clase a la que pertenece el objeto que se encuentra en la imagen. (TensorFlow, 2024)

2.2.3 Kaggle

Kaggle, una plataforma líder en ciencia de datos y aprendizaje automático, ofrece una amplia gama de datasets y desafíos competitivos gratuitos. Kaggle, que se fundó en 2010 y fue adquirida por Google en 2017, se ha convertido en un lugar de referencia para los científicos de datos y los entusiastas del aprendizaje automático. Su amplia colección de datos abarca una variedad de temas, desde salud hasta finanzas, y brinda a los usuarios la oportunidad de explorar y trabajar con datos reales. (Kaggle, 2020)

2.2.4 Robo Flow

Roboflow es una plataforma para administrar y preparar conjuntos de imágenes para proyectos de visión por computadora. Roboflow, que se estableció en 2020 y ofrece herramientas sofisticadas para anotar, preprocesar y mejorar la calidad de los conjuntos de imágenes. Para entrenar modelos de aprendizaje automático, los usuarios pueden subir sus propias bases de datos o utilizar las bases de datos públicas que están disponibles en la plataforma. La integración de Roboflow con TensorFlow y PyTorch, dos herramientas de aprendizaje automático populares, facilita la integración de conjuntos preparados en proyectos de desarrollo de modelos. (RoboFlow, 2020)

2.2.5 Google Colab

Google Colab es un servicio en la nube gratuito de Google que ofrece un entorno de notebook basado en Jupyter con acceso a CPU, GPU y memoria RAM. Esto permite a los usuarios ejecutar y desarrollar código Python de forma interactiva sin necesidad de configurar un entorno de desarrollo local. Debido a que es fácil de usar y tiene potentes capacidades computacionales gratuitas, Colab es particularmente popular entre los científicos de datos, investigadores y estudiantes.

En cuanto a los planes de pago de Google Colab, brindan acceso a recursos informáticos más potentes, como GPUs de alto rendimiento, que permiten acelerar significativamente el entrenamiento de modelos de aprendizaje automático. Esto es particularmente ventajoso para tareas que requieren mucho poder computacional, como el procesamiento de imágenes con redes neuronales convolucionales (CNN), donde el uso de GPU puede reducir significativamente el tiempo de entrenamiento y mejorar la eficiencia del modelo. (Google, 2024)

CAPÍTULO 3: DESARROLLO DEL MODELO

3.1 Recopilación y preprocesamiento de datos

3.1.1 Comprensión de los datos

La recopilación de datos de imágenes de armas blancas para este proyecto se realizó utilizando múltiples fuentes para asegurar una diversidad y riqueza en el dataset. Inicialmente, se obtuvieron 400 imágenes de cuchillos de Kaggle. Estas imágenes proporcionaron una base sólida y confiable para el dataset. Posteriormente, se ampliaron los datos con 533 imágenes de cuchillos adicionales descargadas de la extensión de Chrome "Download All Images" (Download All Images, 2024). Esta herramienta facilitó la recopilación masiva de imágenes de cuchillos desde diversas fuentes web, aumentando significativamente la variedad de ángulos, estilos y contextos en los que los cuchillos aparecían. Finalmente, se integraron 5895 imágenes de cuchillos provenientes de Roboflow, una plataforma especializada en la gestión y creación de datasets de imágenes para proyectos de visión por computadora. Por su parte, las imágenes de manoplas igualmente fueron obtenidas de Roboflow, con un total de 1193.

3.1.2 Preparación de los datos

Las imágenes obtenidas a través de la extensión "Download All Images" requirieron un proceso de limpieza exhaustivo para asegurar su viabilidad en el proyecto de reconocimiento de cuchillos. Inicialmente, se recopilaron 533 imágenes, pero mediante un proceso de revisión manual y eliminación, se alcanzó un total final de 439 imágenes viables. Este proceso de

limpieza se realizó utilizando un script en Google Colab que permitía la inspección visual y la eliminación de imágenes inadecuadas. El código funcionaba de la siguiente manera:

Primero, se montaba Google Drive para acceder a los archivos almacenados. Luego, se definía el directorio de imágenes y se generaba una lista de todas las imágenes con extensiones .png,

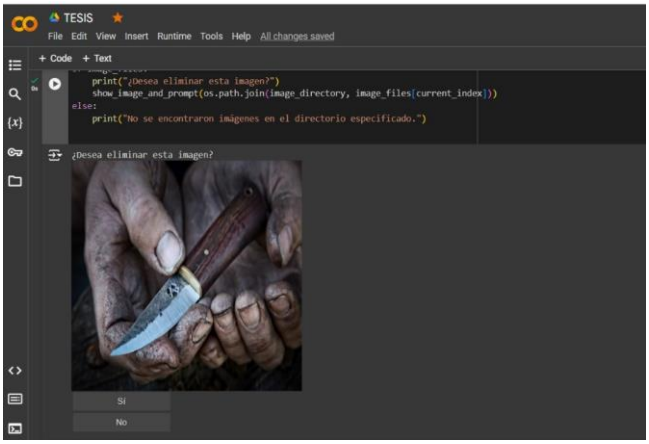


Ilustración 2: Limpieza de Imágenes

.jpg, y .jpeg. La función ‘show_image_and_prompt’ se encargaba de mostrar cada imagen con un tamaño reducido a 350x350 píxeles para facilitar su visualización, junto con dos botones interactivos: "Sí" y "No".

Cuando se hacía clic en el botón "Sí", la imagen era eliminada del directorio mediante la función ‘on_button_yes_clicked’, que también actualizaba el índice de la imagen actual y limpiaba la salida de la celda para mostrar la siguiente imagen en la lista. Si se hacía clic en "No", la función ‘on_button_no_clicked’ simplemente actualizaba el índice y pasaba a la siguiente imagen sin eliminar la actual. Este ciclo se repetía hasta que no quedaran más imágenes por revisar, asegurando que solo las imágenes adecuadas fueran retenidas para el proyecto.

Este método permitió una limpieza manual pero eficiente de las imágenes, asegurando que el conjunto final de datos fuera de alta calidad y adecuado para entrenar el modelo de reconocimiento de imágenes de cuchillos.

Finalmente, se juntaron todas las imágenes de cuchillos en una carpeta de Google Drive, resultando en un total de 6833, más las 1193 de manoplas para el modelo. Para el entrenamiento del modelo, es crucial eliminar las imágenes duplicadas para evitar que el modelo aprenda de manera incorrecta o se sobreajuste a datos redundantes. Se utiliza la biblioteca ‘imagehash’ para calcular un hash único para cada imagen y luego compara estos hashes para identificar y eliminar las imágenes duplicadas. La función ‘calculate_hash’ se encarga de calcular el hash de una imagen dada, y el diccionario ‘hash_dict’ se utiliza para almacenar los hashes de las imágenes procesadas, de modo que las duplicadas puedan ser identificadas y eliminadas correctamente. En este caso, se eliminaron las imágenes duplicadas del repositorio, quedando un total de 3676 imágenes de cuchillos y 1173 de manoplas después del proceso.

Las imágenes del dataset de Kaggle son de tamaño 100x100 píxeles, mientras que las de Roboflow son más grandes, con dimensiones de 640x640 píxeles. En contraste, las imágenes descargadas con la herramienta "Download All Images" presentan una variabilidad considerable en tamaño, ya que dependen de la fuente en que fueron obtenidas, lo que puede influir en la calidad y la uniformidad del conjunto de datos si no es controlado.

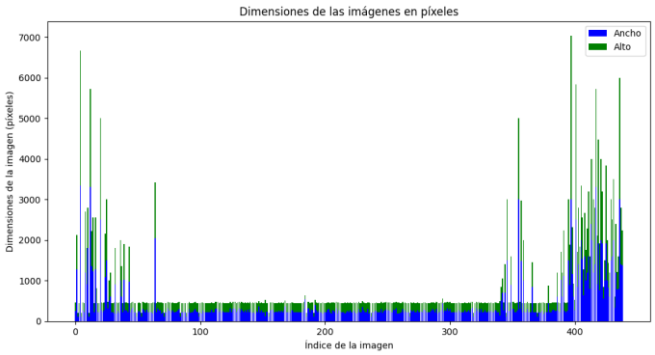


Ilustración 3: Variabilidad tamaños (Extensión Chrome)

Para garantizar la consistencia en el tamaño de las imágenes y optimizar el proceso de entrenamiento del modelo, todas las imágenes se redimensionaron a 300x300 píxeles. Esta decisión se basa en las características específicas de las fuentes de datos utilizadas: las imágenes de la extensión de Google Chrome tienen un tamaño similar, lo que facilita el procesamiento uniforme; las imágenes de Roboflow no deben hacerse muy pequeñas para evitar pérdida de información; y las imágenes de Kaggle no pueden agrandarse demasiado sin perder calidad. Redimensionar las imágenes a un tamaño estándar como 300x300 píxeles ayuda a mantener la calidad y consistencia de los datos de entrada para el modelo de CNN.

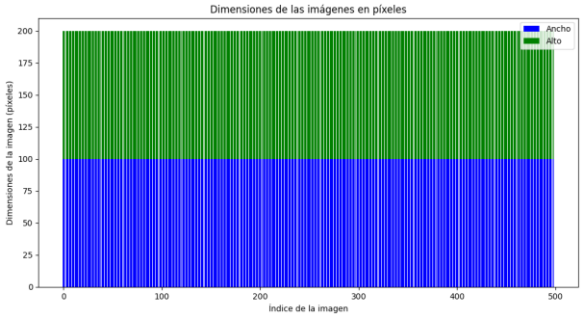


Ilustración 4: Tamaño de imágenes

Dado que una sola capa de escala de grises requiere menos procesamiento que las tres capas de color (rojo, verde y azul), las imágenes se pasan a una escala de grises en el modelado para reducir la complejidad computacional. Además, esto simplifica la información al centrar el modelo en formas y texturas en lugar de colores, lo que es importante para tareas donde los colores no son importantes. Además, evita que el modelo se sobreajuste a colores específicos, reduce el ruido causado por variaciones de iluminación y balance de color y mejora la transferibilidad a áreas que usan imágenes en escala de grises, como las térmicas o de ultrasonido.

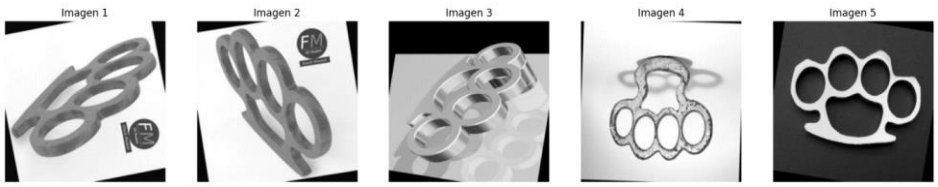


Ilustración 5: Manoplas Escala de Grises

Las imágenes de Google Drive se transforman en arreglos de NumPy normalizados entre 0 y 1 porque esto mejora la estabilidad numérica, evita problemas de desbordamiento, acelera la convergencia durante el entrenamiento al proporcionar gradientes más estables, asegura la consistencia de los datos al unificar el rango de valores de los píxeles y optimiza la compatibilidad con funciones de activación que funcionan mejor. Se deben crear dos arreglos, uno para los cuchillos y otro para las manoplas, esto con la intención de concatenar estos arreglos para que estén divididos de manera equitativa y no se mezclen dentro del arreglo, pues seguido de esto se crean dos arreglos más, cada uno con la longitud de su respectivo arreglo de imágenes, estos arreglos estarán formados por los valores “0” y “1”, creando así las etiquetas para las imágenes, “0” para las manoplas y “1” para los cuchillos. Finalmente, estos arreglos de etiquetas

se concatenan igual que los anteriores y de esa forma se transformaron en arreglos NumPy las imágenes y se las etiquetó para el posterior entrenamiento del modelo, resultando en los arreglos “arrTrain” y “labels_train”.

```python
#IMÁGENES DE ENTRENAMIENTO CUCHILLOS

import os
import cv2
import numpy as np

# Directorio de las imágenes
image_directory = '/content/drive/MyDrive/Cuchillos/entrenamientoFinal/cuchillos_final/'

# Obtener la lista de archivos en el directorio
image_files = [f for f in os.listdir(image_directory) if f.endswith(('.png', '.jpg', '.jpeg'))]

# Inicializar una lista para almacenar los arrays de imágenes
image_data = []

# Convertir y almacenar las imágenes en un arreglo numpy
for image_file in image_files:
    image_path = os.path.join(image_directory, image_file)
    img = cv2.imread(image_path)
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convertir a RGB
        img = img.reshape((300, 300, 1))
        image_data.append(img)

#Normalizar los datos de las X (imagenes). Se pasan a numero flotante y dividen entre 255 para quedar de 0-1 en lugar de 0-255
arrTrainKnives = np.array(image_data).astype(float) / 255

print("Arreglo numpy guardado exitosamente.")
```

Ilustración 6: Código arreglos NumPy

```
[ ]  arrTrain = np.concatenate((arrTrainKnives, arrTrainKnuckles))

[ ]  labels_train = np.concatenate((labelsKnives, labelsKnuckles))
```

Ilustración 7: Concatenando Arreglos

Una técnica para el entrenamiento de modelos de aprendizaje automático es usar train_datagen = ImageDataGenerator(...) con esos parámetros (Ilustración 8).

```
train_datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=15,
    zoom_range=[0.7, 1.4],
    horizontal_flip=True,
    vertical_flip=True
)
```

Ilustración 8: Aumento de datos

Esta técnica se utiliza para aumentar la variedad de los datos de entrenamiento, fortalecer el modelo y generalizarlo mejor a los nuevos datos. Aquí está una explicación breve de cada parámetro:

1. rotation_range=30:

- Rota las imágenes aleatoriamente hasta 30 grados en cualquier dirección.
- Ayuda al modelo a aprender que el objeto puede estar inclinado en diferentes ángulos.

2. width_shift_range=0.2:

- Desplaza las imágenes aleatoriamente hasta un 20% de su ancho.

- Simula el movimiento lateral del objeto dentro de la imagen.

3. height_shift_range=0.2:

- Desplaza las imágenes aleatoriamente hasta un 20% de su altura.
- Simula el movimiento vertical del objeto dentro de la imagen.

4. shear_range=15:

- Aplica un estiramiento en ángulo de hasta 15 grados.
- Distorsiona la imagen para que el modelo aprenda a reconocer el objeto incluso cuando está estirado o deformado.

5. zoom_range=[0.7, 1.4]:

- Acerca o aleja la imagen aleatoriamente entre el 70% y el 140% de su tamaño original.
- Ayuda al modelo a identificar el objeto a diferentes escalas.

6. horizontal_flip=True:

- Invierte las imágenes horizontalmente (como un espejo).
- Enseña al modelo que el objeto puede aparecer invertido de izquierda a derecha.

7. vertical_flip=True:

- Invierte las imágenes verticalmente.

- Enseña al modelo que el objeto puede aparecer invertido de arriba abajo

Estos ajustes producen nuevas variaciones de las imágenes de entrenamiento cada vez que el modelo las ve. Esto hace que el modelo sea más robusto y pueda manejar objetos de diferentes posiciones, tamaños y orientaciones. Esto crea una variedad artificial que mejora el rendimiento del modelo, lo que lo hace especialmente útil cuando se tiene un conjunto de datos limitado.

3.2 Creación de una Red Neuronal Convolucional

El primer paso para que todo el modelado funcione es instalar tensorflow, de esa forma se puede realizar la configuración del modelo de esta forma:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Definir el modelo
model = Sequential()

# Capa convolucional 1
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 1)))
model.add(MaxPooling2D((2, 2)))

# Capa convolucional 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Capa convolucional 3
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Aplanar la salida de la última capa convolucional
model.add(Flatten())

# Capa completamente conectada
model.add(Dense(100, activation='relu'))

# Capa de salida
model.add(Dense(1, activation='sigmoid'))  # Suponiendo clasificación binaria (cuchillo o manopla)

# Compilar el modelo
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Resumen del modelo
model.summary()
```

Ilustración 9: Código Modelo CNN

Para obtener características de las imágenes, se utilizan tres capas convolucionales seguidas de capas de max pooling para extraer características de las imágenes. Las capas convolucionales detectan características como bordes y texturas en diferentes áreas de la imagen mediante el uso de filtros en áreas específicas de la imagen. La operación de max pooling reduce la dimensionalidad de las características, manteniendo las más importantes y mejorando simultáneamente la eficiencia computacional.

Para capturar las características simples y complejas de las imágenes de cuchillos, se eligió una arquitectura de tres capas convolucionales. Las capas convolucionales superiores pueden

detectar bordes y patrones simples, mientras que las capas convolucionales más profundas pueden detectar características más complejas y particulares de los cuchillos.

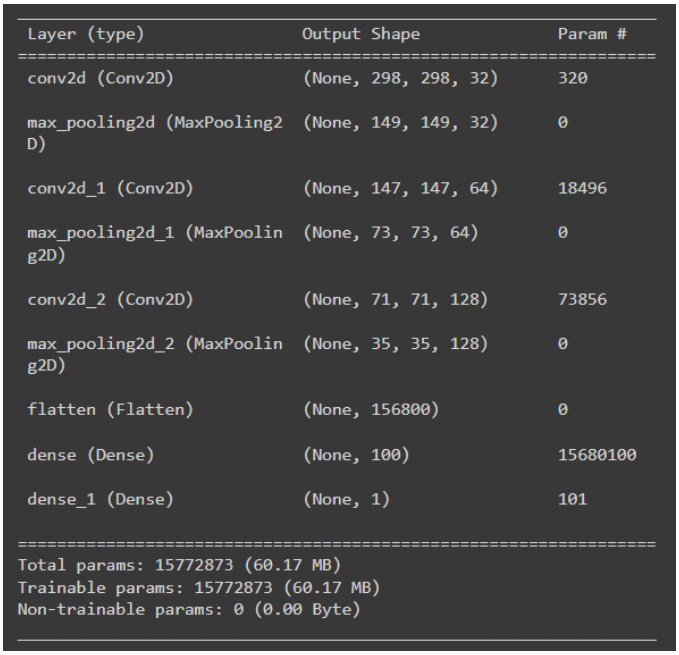
Además, para introducir no linealidad en la red, se utilizan capas de activación ReLU (Rectified Linear Unit) después de cada capa convolucional. Al propagar los gradientes hacia atrás durante el entrenamiento, la función ReLU ayuda a la red a aprender de manera más efectiva y evita el problema de desvanecimiento del gradiente.

Las características extraídas se aplanan en un vector unidimensional después de las capas convolucionales para que puedan pasar a través de capas densas. La red puede aprender representaciones más abstractas y complejas de las características de los cuchillos porque la capa densa final contiene 128 neuronas con activación ReLU.

Finalmente, la capa de salida clasifica binariamente las imágenes de cuchillos y manoplas utilizando una función de activación sigmoidea. La función de pérdida utilizada es la entropía cruzada binaria, adecuada para problemas de clasificación binaria.

El objetivo de las decisiones de diseño del código es lograr una alta precisión en la clasificación de imágenes mediante la creación de una CNN que pueda aprender de manera efectiva las características distintivas de las imágenes de armas blancas. Cada componente de la red, desde las capas convolucionales hasta las capas de salida, se ha diseñado cuidadosamente para maximizar el rendimiento y la eficiencia del modelo para esta tarea en particular.

3.2.1 Resumen del Modelo



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 32)      320

 max_pooling2d (MaxPooling2  (None, 149, 149, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 147, 147, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 73, 73, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 71, 71, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 35, 35, 128)       0
 g2D)

 flatten (Flatten)           (None, 156800)            0

 dense (Dense)               (None, 100)               15680100

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 15772873 (60.17 MB)
Trainable params: 15772873 (60.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Ilustración 10: Modelo CNN

Capa Convolucional 1 (conv2d):

- Output Shape (Forma de Salida): (None, 298, 298, 32)
    - Aquí, None es el tamaño del lote (batch size), que puede variar.
    - 298, 298 son las dimensiones espaciales de la imagen de salida.
    - 32 es el número de filtros (kernels) utilizados en esta capa.
- Param # (Parámetros): 320

- Este número es calculado como (3*3*1 + 1)*32, donde 3*3 es el tamaño del filtro, 1 es el número de canales de entrada (imagen en escala de grises), y 1 es el sesgo (bias) añadido por cada filtro.

Capa de Max Pooling 1 (max_pooling2d):

- Output Shape: (None, 149, 149, 32)
  - Esta capa reduce las dimensiones espaciales a la mitad usando un filtro de 2x2 y stride (paso) de 2.
- Param #: 0
  - No tiene parámetros entrenables.

Capa Convolucional 2 (conv2d_1):

- Output Shape: (None, 147, 147, 64)
  - Esta capa aplica 64 filtros, resultando en 64 mapas de características.
- Param #: 18496
  - Este número es calculado como (3*3*32 + 1)*64, donde 3*3 es el tamaño del filtro, 32 es el número de canales de entrada, y 1 es el sesgo por cada filtro.

Capa de Max Pooling 2 (max_pooling2d_1):

- Output Shape: (None, 73, 73, 64)
  - Reduce las dimensiones espaciales nuevamente.

- Param #: 0
    - No tiene parámetros entrenables.

Capa Convolucional 3 (conv2d_2):

- Output Shape: (None, 71, 71, 128)
    - Aplica 128 filtros.
- Param #: 73856
    - Calculado como (3*3*64 + 1)*128, donde 3*3 es el tamaño del filtro, 64 es el número de canales de entrada, y 1 es el sesgo por cada filtro.

Capa de Max Pooling 3 (max_pooling2d_2):

- Output Shape: (None, 35, 35, 128)
    - Reduce las dimensiones espaciales a la mitad.
- Param #: 0
    - No tiene parámetros entrenables.

Capa de Aplanado (flatten):

- Output Shape: (None, 156800)
    - Aplana las características en un vector unidimensional.
- Param #: 0

- No tiene parámetros entrenables.

Capa Densa 1 (dense):

- Output Shape: (None, 100)
  - Tiene 100 neuronas completamente conectadas.
- Param #: 15680100
  - Calculado como 156800*100 + 100, donde 156800 es el tamaño de entrada, 100 es el número de neuronas y 1 es el sesgo por neurona.

Capa de Salida (dense_1):

- Output Shape: (None, 1)
  - Tiene una sola neurona para la salida binaria (cuchillo o manopla).
- Param #: 101
  - Calculado como 100*1 + 1, donde 100 es el número de neuronas de la capa anterior y 1 es el sesgo.

3.2.1.1 Interpretación

- Las capas convolucionales extraen características espaciales importantes de las imágenes.
- Las capas de pooling reducen las dimensiones espaciales, lo que ayuda a reducir el costo computacional y la posibilidad de overfitting.

- La capa de aplanado convierte la salida 3D de la última capa convolucional en un vector unidimensional.
- Las capas densas actúan como clasificadores que toman las características extraídas y producen la predicción final.
- El modelo tiene un total de 15,772,873 parámetros que se entrenan durante el proceso de entrenamiento.

3.3 Entrenamiento y validación del modelo

TensorBoard se utilizará para visualizar y monitorear métricas como la precisión, la pérdida, los histogramas y los gráficos del modelo en tiempo real durante el entrenamiento del modelo. Permite comprender cómo el modelo aprende y se ajusta durante el entrenamiento, lo que ayuda a detectar problemas como sobreajuste, convergencia lenta o inestabilidad numérica. Además, TensorBoard ayuda a comparar varias configuraciones y ejecuciones de hiperparámetros, lo que mejora el proceso de desarrollo y el rendimiento del modelo.

```
[ ]  from tensorflow.keras.callbacks import TensorBoard

[ ]  tensorboarCNN = TensorBoard(log_dir='logs/cnn')
```

Ilustración 11: Declaración TensorBoard

La función flow del generador se utiliza para tomar las imágenes de entrenamiento (X_entrenamiento) y sus etiquetas correspondientes (y_entrenamiento) y convertirlas en grupos

de 32 imágenes cada uno. En lugar de cargar todas las imágenes a la vez, lo cual podría ser muy lento y consumir mucha memoria, estos lotes se envían al modelo en partes. Por lo tanto, el entrenamiento del modelo es más eficiente y fácil de manejar.

```
#Para crear un iterador que podamos enviar como entrenamiento a la función FIT del modelo, ut
data_gen_entrenamiento = train_datagen.flow(X_entrenamiento, y_entrenamiento, batch_size=32)
```

Ilustración 12: Función Flow

Ahora hay que calcular e igualar los pesos de las clases pues hay un desequilibrio en la cantidad de ejemplos que pertenecen a cada clase en los datos de entrenamiento, los cuchillos tienen más de 2000 imágenes y las manoplas apenas 1000.

```
from sklearn.utils.class_weight import compute_class_weight
# Calcular los pesos de las clases
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(labels_train), y=labels_train)

# Convertir a un diccionario para pasarlo al modelo
class_weights = {i: class_weights[i] for i in range(len(class_weights))}
```

Ilustración 13: Balance de clases

En la primera línea, se calculan los pesos de las clases utilizando la función “compute_class_weight”. Se especifica “class_weight='balanced” para que los pesos se calculen de manera que compensen automáticamente el desequilibrio en las clases.“np.unique(labels_train)” se utiliza para obtener las clases únicas presentes en los datos de entrenamiento.

Por su parte, en la segunda línea, se convierten los pesos de las clases en un diccionario para poder pasarlos al modelo de clasificación. Cada clave del diccionario es el índice de una clase, y el valor es el peso correspondiente a esa clase.

```python
# Crear un objeto TensorBoard para registrar los datos del entrenamiento y visualizarlos en TensorBoard
tensorboardCNN = TensorBoard(log_dir='logs/cnn')

# Entrenar el modelo con los datos de entrenamiento
model.fit(
    data_gen_entrenamiento,  # Generador de datos de entrenamiento
    epochs=150,              # Número de épocas de entrenamiento
    batch_size=32,           # Tamaño del lote de entrenamiento
    validation_data=(X_validacion, y_validacion),  # Datos de validación
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),  # Pasos por época
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),    # Pasos de validación por época
    callbacks=[tensorboardCNN],   # Lista de devoluciones de llamada, en este caso, TensorBoard
    class_weight=class_weights    # Pesos de las clases para el entrenamiento
)
```

Ilustración 14: Entrenamiento del Modelo

Se hace uso de la función fit de un modelo de red neuronal convolucional (CNN) para entrenar el modelo con los datos de entrenamiento. Se especifica un generador de datos de entrenamiento (data_gen_entrenamiento) para manejar el flujo de datos durante el entrenamiento. Se establece el número de épocas de entrenamiento en 150 y el tamaño del lote en 32. Además, se proporcionan los datos de validación (X_validacion y y_validacion) para evaluar el rendimiento del modelo en datos no vistos durante el entrenamiento. Se calculan los pasos por época y los pasos de validación por época para garantizar que se recorran todos los datos. Se utiliza una devolución de llamada (TensorBoard) para registrar los datos del entrenamiento y visualizarlos en TensorBoard. Además, se utilizan los pesos de las clases (class_weights) para abordar el desequilibrio de clases en los datos de entrenamiento.

CAPÍTULO 4: VALIDACIÓN DE RESULTADOS

4.1 Evaluación del modelo

```
Epoch 141/150
86/86 [==============================] - 17s 194ms/step - loss: 0.0839 - accuracy: 0.9719 - val_loss: 0.2405 - val_accuracy: 0.9373
Epoch 142/150
86/86 [==============================] - 17s 195ms/step - loss: 0.0812 - accuracy: 0.9778 - val_loss: 0.1854 - val_accuracy: 0.9723
Epoch 143/150
86/86 [==============================] - 17s 195ms/step - loss: 0.0749 - accuracy: 0.9785 - val_loss: 0.2006 - val_accuracy: 0.9679
Epoch 144/150
86/86 [==============================] - 17s 195ms/step - loss: 0.0746 - accuracy: 0.9738 - val_loss: 0.1953 - val_accuracy: 0.9679
Epoch 145/150
86/86 [==============================] - 17s 196ms/step - loss: 0.0713 - accuracy: 0.9778 - val_loss: 0.2054 - val_accuracy: 0.9621
Epoch 146/150
86/86 [==============================] - 17s 193ms/step - loss: 0.0829 - accuracy: 0.9712 - val_loss: 0.2171 - val_accuracy: 0.9446
Epoch 147/150
86/86 [==============================] - 17s 196ms/step - loss: 0.0779 - accuracy: 0.9756 - val_loss: 0.1910 - val_accuracy: 0.9650
Epoch 148/150
86/86 [==============================] - 17s 194ms/step - loss: 0.0929 - accuracy: 0.9698 - val_loss: 0.1633 - val_accuracy: 0.9636
Epoch 149/150
86/86 [==============================] - 17s 194ms/step - loss: 0.0810 - accuracy: 0.9727 - val_loss: 0.1832 - val_accuracy: 0.9752
Epoch 150/150
86/86 [==============================] - 17s 193ms/step - loss: 0.0774 - accuracy: 0.9749 - val_loss: 0.1703 - val_accuracy: 0.9708
<keras.src.callbacks.History at 0x7fdb60733b50>
```

Ilustración 15: Resultados modelo

- Epochs y Batch Size: Se está entrenando el modelo durante 150 epochs, con un batch size de 86. Un batch size más grande puede acelerar el entrenamiento, pero también requiere más memoria.
- Loss y Accuracy: En la primera época, el modelo tiene una pérdida (loss) de aproximadamente 0.517 y una precisión (accuracy) de alrededor del 78.24% en el conjunto de entrenamiento. En el conjunto de validación, la pérdida es de aproximadamente 0.486 y la precisión es del 79.30%. Estos valores son indicativos de un buen comienzo.
- Evolución de Loss y Accuracy: A lo largo de las épocas, la pérdida y la precisión en el conjunto de entrenamiento y validación varían. En general, la pérdida tiende a disminuir y la precisión tiende a aumentar, lo cual es un buen indicador de que el modelo está aprendiendo de manera efectiva.

- Sobreajuste: No se observa un claro sobreajuste en los datos de validación, ya que la precisión en el conjunto de validación sigue una tendencia similar a la del conjunto de entrenamiento. Sin embargo, sería útil observar la evolución de estos valores en más detalle para determinar si se produce sobreajuste en algún momento.

4.2 Análisis de resultados

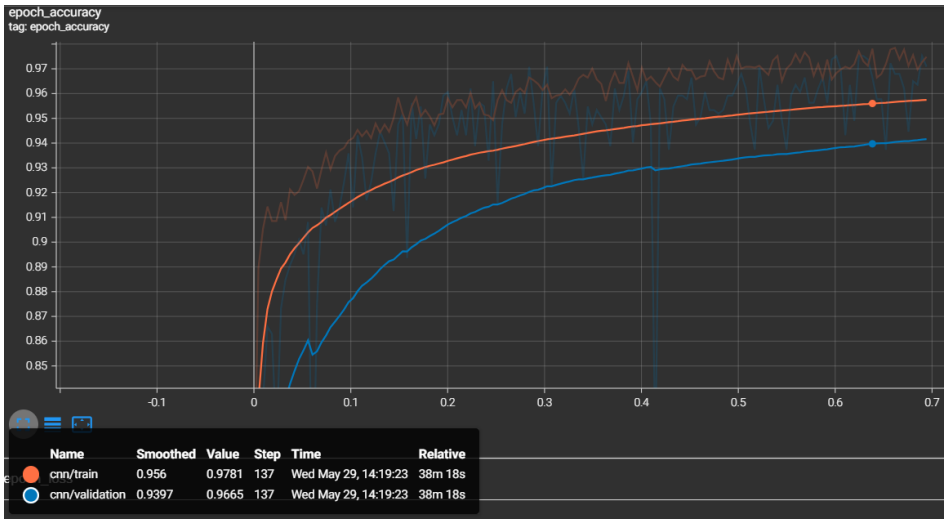


Ilustración 16: Precisión por Épocas

El gráfico curvas que representan la precisión (accuracy) en el conjunto de entrenamiento (cnn/train, en naranja) y en el conjunto de validación (cnn/validation, en azul). En el eje Y se muestra la precisión, que varía de 0.85 a 0.97, mientras que en el eje X se representa el tiempo relativo en horas desde el inicio del entrenamiento, cubriendo aproximadamente 42 minutos.

Al finalizar el entrenamiento, la precisión en el conjunto de entrenamiento es de 0.9781 y en el conjunto de validación es de 0.9665. El gráfico también incluye curvas suavizadas para ofrecer

una idea más clara de la tendencia general en los datos de precisión durante el entrenamiento. Ambas curvas muestran un aumento inicial rápido en la precisión, que se va estabilizando a medida que avanza el entrenamiento. La precisión en el conjunto de validación sigue de cerca a la precisión en el conjunto de entrenamiento, lo que sugiere que el modelo no está sobreajustando significativamente.

Una precisión de 0.9781 en entrenamiento y 0.9665 en validación indica que el modelo tiene un rendimiento muy bueno. La pequeña diferencia entre las precisiones de entrenamiento y validación sugiere que el modelo generaliza bien y no está sobreajustado. Las curvas de precisión muestran que el modelo ha convergido adecuadamente, ya que la curva de entrenamiento ha alcanzado un plató y la curva de validación se ha estabilizado, indicando que el modelo no está mejorando significativamente con más entrenamiento.

Dado que hay alrededor de 2000 imágenes de cuchillos y 1000 de manoplas, el modelo podría estar sesgado hacia la clase con más ejemplos (cuchillos). Sin embargo, el alto rendimiento en el conjunto de validación sugiere que el modelo está manejando bien el desequilibrio de clases.

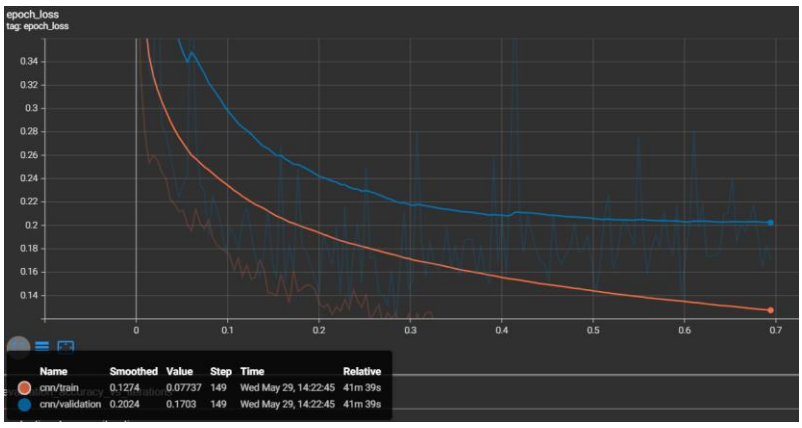


Ilustración 17: Pérdida por Épocas

El gráfico muestra el resultado del entrenamiento centrado en la pérdida (loss) a lo largo del tiempo tanto para el conjunto de entrenamiento (cnn/train, en naranja) como para el conjunto de validación (cnn/validation, en azul). En el eje Y se presenta la pérdida, que varía de 0.0 a 0.34, mientras que en el eje X se muestra el tiempo relativo en horas desde el inicio del entrenamiento, cubriendo aproximadamente 42 minutos.

Al finalizar el entrenamiento, la pérdida en el conjunto de entrenamiento es de 0.07737 y en el conjunto de validación es de 0.1703. Al igual que en el gráfico de precisión, se incluyen curvas suavizadas para proporcionar una visión más clara de la tendencia general en los datos de pérdida durante el entrenamiento. Ambas curvas muestran una disminución rápida inicial en la pérdida, que se va estabilizando a medida que avanza el entrenamiento. La curva de pérdida en el conjunto de validación sigue una tendencia similar a la del conjunto de entrenamiento, aunque con un valor de pérdida ligeramente mayor, lo que es esperado.

Una pérdida final de 0.07737 en entrenamiento y 0.1703 en validación indica que el modelo está ajustando bien los datos de entrenamiento y generalizando adecuadamente. La pequeña diferencia entre las pérdidas de entrenamiento y validación sugiere que el modelo no está sobreajustado y mantiene su capacidad de generalización. Las curvas de pérdida muestran que el modelo ha convergido correctamente, ya que ambas curvas se han estabilizado, indicando que el modelo no está mejorando significativamente con más entrenamiento.

Ilustración 18: Resultados Conjunto de Prueba

El modelo parece tener una precisión aceptable, ya que la mayoría de las predicciones coinciden con las etiquetas verdaderas. Sin embargo, hay un número significativo de errores que no se deben ignorar. Los errores en la clasificación parecen ocurrir más frecuentemente en casos donde la manopla y el cuchillo podrían tener características visuales similares o cuando las imágenes son de baja calidad o tienen mucho ruido. La distribución de los errores muestra un claro patrón que sugiera un sesgo específico hacia una clase en particular, indicando que el modelo está sesgado hacia las manoplas, probablemente por la diferencia en la cantidad de datos.

	precision	recall	f1-score	support
Clase 0 (Cuchillo)	1.00	0.62	0.77	16
Clase 1 (Manopla)	0.67	1.00	0.80	12
accuracy			0.79	28
macro avg	0.83	0.81	0.78	28
weighted avg	0.86	0.79	0.78	28

Ilustración 19: Métricas de desempeño

El modelo CNN muestra un rendimiento aceptable en la clasificación de cuchillos y manoplas, con una precisión general del 79%

Clase 1 (Cuchillo)

- Precisión: La precisión para la clase de cuchillos es del 100%. Esto significa que todas las instancias que el modelo predijo como cuchillos efectivamente eran cuchillos.
- Recall: El recall es del 62%, lo que indica que el modelo solo identificó correctamente el 62% de los cuchillos presentes en las imágenes.

- F1-Score: El f1-score, que es la media armónica de la precisión y el recall, es del 77%. Este valor es un compromiso entre la precisión y el recall, indicando un rendimiento moderado en la detección de cuchillos.

Clase 0 (Manopla)

- Precisión: La precisión para la clase de manoplas es del 67%. Esto significa que el 67% de las instancias que el modelo predijo como manoplas eran correctas.
- Recall: El recall para las manoplas es del 100%, indicando que el modelo identificó correctamente todas las manoplas presentes en las imágenes.
- F1-Score: El f1-score para las manoplas es del 80%, lo que refleja un buen equilibrio entre la precisión y el recall para esta clase.

Métricas Generales

- Accuracy: La precisión global del modelo es del 79%, lo que indica que el modelo clasifica correctamente el 79% de todas las imágenes.
- Macro Avg: La media aritmética de la precisión, el recall y el f1-score entre las dos clases son 0.83, 0.81 y 0.78 respectivamente. Estas métricas indican que, en promedio, el modelo tiene un buen rendimiento en ambas clases, aunque con ciertas limitaciones.
- Weighted Avg: La media ponderada de la precisión, el recall y el f1-score son 0.86, 0.79 y 0.78 respectivamente. Estas métricas ponderan cada clase por el número de instancias en esa clase, lo que ofrece una visión más representativa del rendimiento general del modelo.

Comentado [RRJP1]:

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

El modelo de Red Neuronal Convolucional desarrollado ha mostrado una precisión general del 79%, con una precisión del 100% para la clase de cuchillos y del 67% para la clase de manoplas. Esto indica que el modelo es capaz de identificar cuchillos con alta precisión, pero tiene un desempeño moderado en la identificación de manoplas.

Se observó que la calidad y la resolución de las imágenes afectan significativamente el rendimiento del modelo. Las imágenes de alta calidad y resolución proporcionaron mejores resultados en términos de precisión, pues al realizarse las pruebas, la clasificación de manoplas no es perfecta justamente con imágenes que no cuentan con estas características.

El modelo mostró variaciones en su desempeño bajo diferentes condiciones de iluminación, lo que sugiere la presencia de sesgos debido a la iluminación variable. Aunque las imágenes se pusieron en blanco y negro para mitigar las condiciones de iluminación desfavorables, las imágenes de prueba de manoplas presentaron errores de clasificación, especialmente en aquellas con poca iluminación.

El sistema desarrollado tiene el potencial de ser una herramienta eficaz para la detección de armas blancas, mostrando una alta precisión en la identificación de cuchillos y un buen equilibrio general en la detección de otras armas .

5.2 Recomendaciones

Aunque ya se han aplicado técnicas de aumento de datos como la rotación, desplazamientos horizontales y verticales, el corte, el zoom y los volteos, se puede mejorar aún más la precisión en la clase de manoplas mediante las siguientes acciones: aumentar el número de imágenes de

manoplas en el conjunto de datos de entrenamiento, lo que puede implicar la recolección de más imágenes de diferentes fuentes o la generación de imágenes sintéticas utilizando técnicas de modelado 3D; continuar utilizando las técnicas actuales de aumento de datos, pero también considerar otras técnicas específicas como el ajuste de brillo y contraste, filtros de color, y técnicas de corte y pegado (cutout y cutmix) que pueden introducir variaciones adicionales; realizar una búsqueda más exhaustiva de hiperparámetros utilizando métodos como la búsqueda en cuadrícula (grid search) o la optimización bayesiana para encontrar la configuración óptima que mejore el rendimiento del modelo en la detección de manoplas.

Es crucial utilizar imágenes de alta calidad y resolución en el conjunto de datos de entrenamiento y validación. Además, implementar técnicas de preprocesamiento para mejorar la calidad de las imágenes, como el ajuste de contraste y la reducción de ruido, puede ayudar a mejorar la precisión del modelo.

Se recomienda incorporar un conjunto de datos de entrenamiento que cubra una amplia gama de condiciones de iluminación, pues esto ayudará al modelo a generalizar mejor y reducir los sesgos. Utilizar técnicas de ajuste de brillo y contraste y asegurarse de que el conjunto de datos de entrenamiento tenga una representación adecuada de imágenes con diferentes condiciones de iluminación puede mejorar la precisión del modelo en la clasificación de manoplas.

Se debería integrar el modelo con sistemas de vigilancia existentes y realizar pruebas en entornos reales. Además, es importante establecer un protocolo de respuesta inmediata basado en las alertas generadas para maximizar la efectividad del sistema en la prevención de incidentes.

6.- BIBLIOGRAFÍA

Download All Images. (2024). Obtenido de Download All Images: https://download-all-images.mobilefirst.me/.

Google. (2024). *Research*. Obtenido de https://research.google.com/colaboratory/intl/es/faq.html#:~:text=Colab%20es%20un%20servicio%20alojado,de%20datos%20y%20la%20educaci%C3%B3n.

Kaggle. (2020). *About Us: Kaggle*. Obtenido de Kaggle: https://www.kaggle.com/

Keita, Z. (Enero de 2024). *Explicación de la detección de objetos YOLO: Datacamp*. Obtenido de Datacamp: https://www.datacamp.com/es/blog/yolo-object-detection-explained

Lia. (24 de Julio de 2023). *Introducción a Machine Learning, Modelos y Aplicaciones: Hexagon Data*. Obtenido de Hexagon Data: https://www.hexagondata.com/es/uncategorized-es/machine-learning-modelos-aplicaciones/#:~:text=El%20Machine%20Learning%20se%20basa,abordar%20distintos%20problemas%20y%20situaciones.

MathWorks. (2023). *¿Qué son las redes neuronales convolucionales?: MathWorks*. Obtenido de MathWorks: https://la.mathworks.com/discovery/convolutional-neural-network.html

MathWorks. (2024). *Reconocimiento de imágenes: MathWorks*. Obtenido de MathWorks: https://la.mathworks.com/discovery/image-recognition-matlab.html

Microsoft. (2024). *Detección de objetos mediante R-CNN más rápida: Microsoft*. Obtenido de Detección de objetos mediante R-CNN más rápida: https://learn.microsoft.com/es-es/cognitive-toolkit/object-detection-using-faster-r-cnn

Microsoft. (2024). *Redes neuronal convolucionales: Microsoft* . Obtenido de Microsoft: https://learn.microsoft.com/es-es/training/modules/train-evaluate-deep-learn-models/4-convolutional-neural-networks

P, P., & M, V. (2018). *Facultad de Matemática, Astronomía, Física y Computación - Universidad de Córdova: Fundamentos básicos del procesamiento de imágenes*. Obtenido de Facultad de Matemática, Astronomía, Física y Computación - Universidad de Córdova: https://www.famaf.unc.edu.ar/~pperez1/manuales/cim/cap2.html#fundamentos-basicos-del-procesamiento-de-imagenes

RoboFlow. (2020). *About Us: Roboflow*. Obtenido de Roboflow: https://roboflow.com/about

TensorFlow. (2024). *Image Classification: TensorFlow*. Obtenido de https://www.tensorflow.org/tutorials/images/classification

TensorFlow. (2024). *Image Segmentation: Tensorflow*. Obtenido de https://www.tensorflow.org/tutorials/images/segmentation

TensorFlow. (2024). *TensorFlow*. Obtenido de https://www.tensorflow.org/tutorials/keras