EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO®

Instituto Tecnológico de Mexicali

Mexicali, Baja California, 18/Junio/2021
Oficio No.: DEPI-37/ 2021
ASUNTO: Autorización de Impresión

ISAAC ANTONIO CARDOZA LEYVA
PRESENTE.

El que suscribe, Jefe de la División de Estudios de Posgrado e Investigación, comunica a usted que para la obtención del grado de **Maestría en Sistemas Computacionales**, se ha **autorizado la impresión de su trabajo de Tesis**, cuyo título es:

"Optimización de hiperparámetros de redes neuronales convolucionales para la detección de armas de fuego en imágenes"

La autorización se emite con base a la revisión y dictamen emitido favorablemente y avalado con su rúbrica por los integrantes del Comité Tutorial, integrado por:

ARNOLDO DÍAZ RAMÍREZ
Presidente

HEBER SAMUEL HERNÁNDEZ TABARES
Secretario

JUAN FRANCISCO IBÁÑEZ SALAS
Vocal

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
La tecnología para el bien de la humanidad®

ARNOLDO DÍAZ RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN

Ccp. Archivo

EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE MEXICALI

Optimización de hiperparámetros de redes neuronales convolucionales para la detección de armas de fuego en imágenes

TESIS PARA LA OBTENCIÓN DEL GRADO MAESTRO EN INGENIERÍA EN SISTEMAS COMPUTACIONALES

Presenta

I.T.I Isaac Antonio Cardoza Leyva.

Directores de Tesis:

Dr. Arnoldo Díaz Ramírez.

Dr. Juan Pablo García Vázquez.

Mexicali, Baja California. Junio 2021.

Resumen

Optimización de hiperparámetros de redes neuronales convolucionales para la detección de armas de fuego en imágenes

por Isaac Antonio CARDOZA LEYVA

El aumento de la violencia relacionada con armas de fuego alrededor del mundo en años recientes se ha convertido en un serio problema que compromete la seguridad de cada individuo en alguna medida. Dicha situación ha motivado a los investigadores a encontrar nuevas formas de mejorar los actuales métodos del estado del arte para solucionar este problema en específico. Entre estos métodos se incluyen principalmente los sistemas de vigilancia automáticos para detectar y clasificar la presencia de armas de fuego dentro de una escena específica, sin los inconvenientes que presenta la necesidad de constante supervisión por parte de personal humano. Profundizando en la tarea de clasificación de imágenes, el desempeño de las técnicas de aprendizaje profundo (Deep Learning) sobresale, especialmente tratándose de redes neuronales convolucionales, ya que estas tienen la propiedad de comenzar a aprender directamente de los datos en bruto (Imágenes). Además de que su proceso de aprendizaje puede ser potenciado aún más mediante el uso de técnicas de transferencia de aprendizaje (Transfer Learning) y seleccionando valores óptimos para los diferentes tipos de hiperparámetros que las componen. Por lo tanto, esta investigación se centra en analizar los beneficios de optimizar los valores de los hiperparámetros que definen el algoritmo de entrenamiento con respecto a sus valores por defecto en el desempeño de un clasificador de armas de fuego (Pistolas y revólveres) en imágenes basado en redes neuronales convolucionales. Para ello, se evalúan los modelos generados por dos diferentes arquitecturas estándar, AlexNet e Inception V3. Así mismo, se evalúan también los efectos de emplear técnicas de transferencia de aprendizaje. Resultados experimentales sugieren una importante relación entre el conjunto de datos, específicas combinaciones de hiperparámetros y el desempeño final del clasificador generado. Ya que una vez aplicado el proceso de optimización de hiperparámetros se presentaron incrementos en la precisión de hasta el 10.33%.

Agradecimientos

A consciencia de que este logro no fue sólo mío, quiero agradecer primeramente a mis directores de tesis: Al Dr. Arnoldo Díaz Ramírez y al Dr. Juan Pablo García Vázquez, porque con su vasta experiencia compensaron la falta de la mía. De la misma manera, quiero agradecer a la maestra Verónica Quintero Rosas por brindarme orientación de principio a fin en todo este proceso de crecimiento académico, personal y profesional. Agradezco también a mis padres Armando Rafael Cardoza Urías y Elizabeth Leyva Cruz por todo su apoyo y paciencia. Finalmente, quiero agradecer también al Instituto Tecnológico de Mexicali y al Consejo Nacional de Ciencia y Tecnología por proporcionarme la beca que hizo posible que diera este importante paso adelante en mi formación.

Contenido

Resumen i

Agradecimientos iii

Índice de figuras vii

Índice de tablas ix

1 Introducción 1
- 1.1 Clasificación de delitos. 2
  - 1.1.1 Delitos contra las personas. 2
- 1.2 Estadísticas delictivas en México. 4
- 1.3 Pregunta de investigación 7
- 1.4 Objetivo general 7
- 1.5 Objetivos específicos 7

2 Conceptos básicos 9
- 2.1 Inteligencia artificial 9
- 2.2 Aprendizaje automático (Machine Learning) 9
- 2.3 Redes Neuronales Artificiales 9
- 2.4 Regresores y clasificadores 10
- 2.5 Transferencia de aprendizaje (Transfer Learning) 10

3 Redes Neuronales Convolucionales 11
- 3.1 Componentes básicos de una red neuronal convolucional. 11
  - 3.1.1 Capa de convolución 11
  - 3.1.2 Capa de pooling 11
  - 3.1.3 ReLU 12
  - 3.1.4 Capas completamente conectadas (Fully connected) 12
  - 3.1.5 Capa softmax 12
  - 3.1.6 Función de pérdida 12
- 3.2 Hiperparámetros 12
  - 3.2.1 Hiperparámetros que definen el algoritmo de entrenamiento 12

4 Materiales y Métodos 15
- 4.1 AlexNet 15
- 4.2 Inception V3 15
- 4.3 Preparación del entorno de trabajo 16
  - 4.3.1 Obtención de librerías de aceleración de hardware 16
  - 4.3.2 Instalación de Anaconda, creación de entorno e instalación de librerías 19
- 4.4 Valores por defecto de los optimizadores de Keras y rangos a considerar en los experimentos 20
  - 4.4.1 Valores por defecto de los optimizadores de Keras: 20

4.4.2 Rangos a considerar para cada hiperparámetro: 20
4.5 Metodología 20
4.6 Obtención del conjunto de datos 21
4.7 Especificaciones de hardware y software para la realización de los experimentos 22

5 Resultados y Discusión **25**
5.1 Resultados 25
5.2 Discusión 28

6 Conclusiones y trabajo futuro **31**

7 Anexos **33**
7.1 Implementación de AlexNet en Python 33
7.2 Implementación de Inception V3 en Python 37
7.3 Verificar versiones 41

Bibliografía **43**

Índice de figuras

1 Distribución estimada de homicidios intencionales a lo largo del mundo durante el año 2017 por mecanismo de perpetración. 1
2 Crecimiento anual de homicidios registrados en México desde 1990 hasta 2018. 2
3 Crímenes más recurrentes en México durante el año 2018. Esta distribución se mantiene en grandes ciudades y áreas metropolitanas. . . 5
4 Porcentaje de uso de armas en la perpetración de delitos en territorio mexicano desde el año 2012 hasta el 2018. 5

5 Arquitectura básica de una red neuronal convolucional. 11

6 Proceso de instalación de CUDA 11.0. 16
7 Contenido de la librería CuDNN. 17
8 Contenido de la carpeta de instalación de CUDA. 17
9 Ventana para acceder a las variables del entorno de Windows 10. . . . 18
10 Variables del entorno. En esta ventana podemos editar las variables del sistema y validar las referencias a las carpetas "bin" y "libnvvp" a través de la variable "Path". 18
11 Iniciando la instalación de Anaconda. 19
12 Ejemplos del contenido de cada clase del conjunto de imágenes elaborado. 22

13 Pérdida y precisión en el proceso de validación a través de las épocas para Inception V3 con hiperparámetros óptimos. 27
14 Pérdida y precisión en el proceso de validación a través de las épocas para AlexNet con hiperparámetros óptimos 27
15 Curva ROC para la red AlexNet con hiperparámetros optimizados. Su valor de área bajo la curva (AUC) es de 0.8709. 27
16 Curva ROC para la red Inception V3 con hiperparámetros optimizados. Su valor de área bajo la curva (AUC) es de 0.9812. 28

Índice de tablas

1 Resultados obtenidos al emplear valores por defecto para los hiperparámetros de los optimizadores de Keras. Tanto para AlexNet como para Inception V3. 25
2 Resultados obtenidos al emplear valores optimizados para los hiperparámetros de los optimizadores de Keras y su respectivo cambio en la precisión con respecto al uso de valores por defecto. Tanto para AlexNet como para Inception V3. 25
3 Matriz de confusión para Inception V3 con hiperparámetros óptimos. . 26
4 Matriz de confusión para AlexNet con hiperparámetros óptimos. . . . 26

1 Introducción

Las armas de fuego juegan un rol incuestionable como instrumento en la perpetración de numerosos crímenes violentos alrededor del mundo, incluyendo robos y homicidios. De acuerdo al Estudio global en homicidio realizado por la Oficina en drogas y crimen de las Naciones Unidas (UNODC) [1], de los 464,000 homicidios que se estima ocurrieron en 2017, el uso de armas de fuego estuvo involucrado en el 54% de los casos (Un total de 238,804 homicidios). Esta distribución puede observarse en la figura 1.

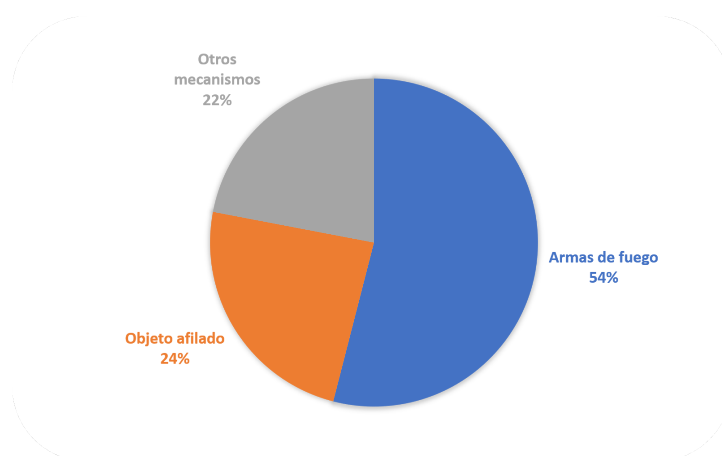


Fig. 1: Distribución estimada de homicidios intencionales a lo largo del mundo durante el año 2017 por mecanismo de perpetración.

Particularmente en el continente americano, los homicidios relacionados con armas de fuego alcanzan un porcentaje de aproximadamente el 75% con respecto a otros mecanismos. Porcentaje que es más alto que en cualquier otra región del mundo.

Otro aspecto fundamental en el estudio de la violencia relacionada con armas de fuego de alguna región en particular, es la tasa de decomisos en dicha región. Ya que el Estudio global en tráfico de armas de fuego 2020 [2] sugiere que un patrón puede discernirse entre el número de armas decomisadas en relación al número de homicidios registrados por región. Mostrando que, altas tasas de decomisos de armas se observan en países con bajos niveles de homicidio, mientras que tasas de decomiso bajas se presentan en países que experimentan altos niveles de homicidio. Esto a su vez sugiere que, niveles de control estrictos sobre las armas de fuego propician bajos niveles de homicidio. Situación que concuerda con el hecho de que incrementos en la tasa de posesión de armas de fuego corresponde también a un incremento en la tasa de homicidios. Este estudio también menciona que los principales tipos de armas de fuego decomisadas en América son las armas de mano, que incluyen pistolas y revólveres.

En el contexto nacional, es bien sabido que la violencia es un problema que afecta la vida diaria de todos los mexicanos en alguna medida, sin embargo, conocer las cifras reales evidencia la gravedad de este asunto. De acuerdo al Instituto Nacional de Estadística y Geografía (INEGI) [3], tan sólo durante el año 2018 se registraron un total de 36,685 homicidios. Lo que representa un crecimiento del 161% en comparación a la última década, tal como puede verse en la figura 2.

Fig. 2: Crecimiento anual de homicidios registrados en México desde 1990 hasta 2018.

Aunque el homicidio pueda ser considerado como la cúspide de todas las formas de violencia, esta no es la única clase de delito con una alta tasa de frecuencia en México que conlleva violencia y que puede o no, incluir el uso de armas de fuego. Por ello, resulta necesario clasificar aquellos delitos donde pueda presentarse violencia con armas de fuego para que así, sea posible medir el problema adecuadamente. Dicha clasificación se presenta en la siguiente subsección.

1.1 Clasificación de delitos.

En el año 2007, el INEGI convocó a diferentes instituciones del país para elaborar una clasificación que fuera de utilidad a un amplio número de usuarios y para todos los sectores de la sociedad interesados en la cuantificación y clasificación de los delitos ocurridos en el país para contribuir a ordenar las estadísticas del tema y reflejar el fenómeno delictivo. De acuerdo a la publicación “Clasificación estadística de delitos 2012” [4], podemos resumir el área de interés de nuestro trabajo en algunos de los elementos que conforman la sección “Delitos contra las personas”.

1.1.1 Delitos contra las personas.

De acuerdo a la descripción presentada en la sección “Delitos contra las personas”, dentro de este grupo se integran la comisión de todas y cada una de las conductas típicas, antijurídicas y culpables, en las cuales se afecta de manera directa alguna de las atribuciones de la persona y la familia individualmente consideradas. Es decir, los efectos de los delitos aquí considerados solamente conciernen a quienes se ven

afectados por los mismos, desligándose completamente de la idea de la seguridad colectiva o estatal.
Resulta importante mencionar que sólo se incluyó en el listado a aquellos delitos que presentan algún nivel de violencia potencialmente observable a través de sistemas de detección visual y que pudieran llegar a incluir el uso de armas de fuego para llevarse a cabo. Dichos delitos se separan en las siguientes 12 sub categorías:

1. Delitos contra la vida.
   (a) Homicidio.
   (b) Genocidio.
   (c) Feminicidio.
   (d) Homicidio en razón del parentesco o relación.
   (e) Incitación o ayuda al suicidio.
2. Delitos contra la integridad corporal o psíquica.
   (a) Ataque peligroso.
   (b) Disparo de arma de fuego.
   (c) Golpes y lesiones.
   (d) Afectaciones a la integridad y salud de menores.
   (e) Tortura y actos relacionados con la misma.
   (f) Actos de violencia cometidos por huelguistas.
3. Delitos contra la libertad física (Corporal).
   (a) Secuestro.
   (b) Secuestro exprés.
   (c) Robo de infante.
   (d) Privación de la libertad con propósitos sexuales.
   (e) Privación ilegal de la libertad.
   (f) Desaparición forzada de personas.
4. Delitos contra la libertad y la seguridad sexual o el normal desarrollo de la personalidad.
   (a) Violación.
   (b) Abuso sexual.
   (c) Acoso sexual.
5. Delitos contra las libertades de reunión, expresión y trabajo.
6. Delitos contra la seguridad individual, la privacidad y la confidencialidad de las personas.
   (a) Asalto.
   (b) Allanamiento de morada.
7. Delitos contra el patrimonio.

   (a) Robo a persona.
   (b) Robo a casa habitación.
   (c) Robo a transeúnte en vía pública.
   (d) Robo a transeúnte en espacio abierto al público.
   (e) Robo en transporte público individual.
   (f) Robo en transporte público colectivo.
   (g) Robo de vehículo.
   (h) Daño a la propiedad.

8. Contra la familia.
   (a) Violencia familiar.
9. Contra la dignidad o la reputación.
   (a) Violencia de género física o psíquica.
10. Contra la responsabilidad profesional.
11. Contra las normas de inhumación o exhumación.
12. Delitos contra las personas no considerados anteriormente.
   (a) Otros delitos contra las personas no considerados anteriormente.

Siguiendo esta clasificación, fue posible reducir el espectro de nuestra investigación. Simplificando el análisis estadístico al enfocarlo solamente en los delitos aquí propuestos.

1.2 Estadísticas delictivas en México.

De acuerdo a la ENVIPE (Encuesta Nacional de Victimización y Percepción sobre seguridad pública) 2019 [5], el crimen más recurrente en México durante el año 2018 fue "Robo o asalto en la calle o transporte público", sumando 9.4 millones de incidencias y abarcando un 28.5% del total de los crímenes cometidos ese año. Cuya distribución se muestra en la figura 3.

En la gráfica podemos observar que crímenes con alto grado de ocurrencia entran dentro de nuestra clasificación de delitos violentos que pueden (O no) involucrar el uso de armas de fuego, acumulando entre estos un 55% del total de los crímenes registrados.

Es importante mencionar que estos números no resultan exactos, debido a las llamadas "Cifras negras" o delitos desconocidos. Estas estimaciones se llevan a cabo dividiendo el número de delitos ocurridos entre la población de 18 años y más multiplicado por cien mil. Se maneja de esta manera porque aún no se implementa un método efectivo para llevar un control preciso sobre la incidencia delictiva a nivel nacional.

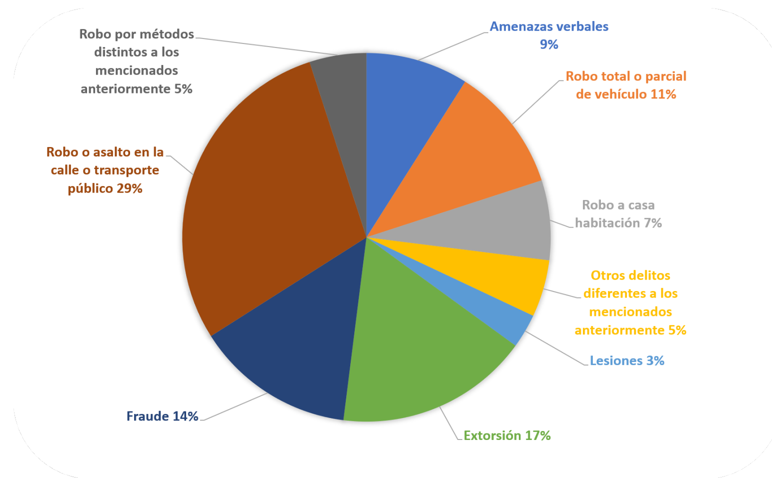


Fig. 3: Crímenes más recurrentes en México durante el año 2018. Esta distribución se mantiene en grandes ciudades y áreas metropolitanas.

Aun con estas cifras no podemos asegurar que exista una conexión entre la perpetración de estos crímenes y el uso de armas de fuego. Sin embargo, estadísticas oficiales del año 2018 [5] indican que de los 18.9 millones de crímenes estimados en donde la victima estuvo presente al momento del crimen, el, o los responsables poseían alguna clase de arma en el 45.6% de los casos. Tratándose de un arma de fuego en el 32.2% de los casos, y, como dato adicional, se tiene registro de que dicha arma (Blanca o de fuego) fue utilizada en el 9.1% de los casos. En la figura 4 se muestra el porcentaje de uso de armas en la perpetración de delitos en territorio mexicano desde el año 2012 hasta el 2018.

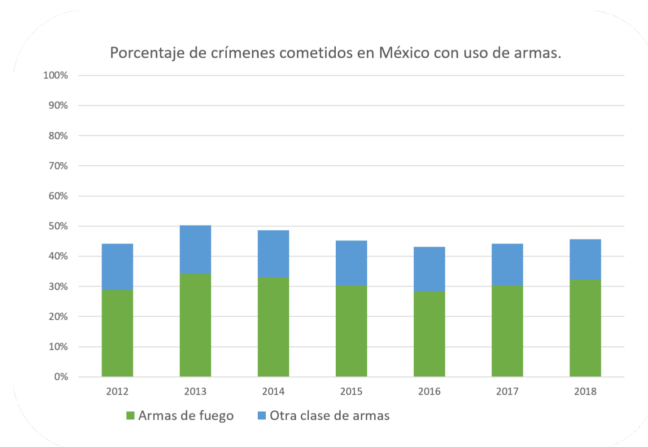


Fig. 4: Porcentaje de uso de armas en la perpetración de delitos en territorio mexicano desde el año 2012 hasta el 2018.

Con esta información es posible reducir el problema al mecanismo más utilizado en la ejecución de crímenes violentos, las armas de fuego. Después, a la clase de armas de fuego más común en la región, pistolas de mano. Y finalmente, a las áreas más susceptibles a este tipo de violencia, las grandes ciudades.

Como establecimos anteriormente, asegurar altos niveles de vigilancia sobre las armas de fuego es un factor crucial para mantener una baja tasa de homicidios en determinada región. Y entre las posibles soluciones para esta tarea contamos con la presencia de sistemas de vigilancia en áreas particularmente vulnerables a presentar casos de asalto a mano armada, como los son las grandes ciudades, ya que nuestra investigación determinó que las áreas metropolitanas son los principales puntos donde se presenta violencia con armas de fuego en México. Sin embargo, los sistemas de vigilancia comúnmente vienen con su propio set de limitaciones, que incluyen principalmente la necesidad de constante supervisión humana sobre inmensos volúmenes de datos. Hecho que hace que esta tarea resulte costosa y poco factible para implementarse a gran escala. Con esto en mente, y tras estudiar el estado del arte (Véase sección "Discusión") para este problema particular, se decidió seguir el enfoque de Aprendizaje Profundo (Deep Learning) con la intención de abordar este problema eficientemente.

En años recientes las redes neuronales convolucionales (CNN), que son una técnica de Deep Learning, han demostrado resultados sumamente prometedores en cuanto a clasificación automática de imágenes, superando inclusive a personas en algunas tareas de clasificación [6]. Específicamente, la clasificación de armas de fuego en imágenes usando CNN ya ha sido estudiada, sin embargo, la precisión de los modelos resultantes de dichos trabajos promedia 94.19%. Porcentaje que aún puede mejorarse si tomamos en cuenta uno de los aspectos mas fundamentales de las CNN: Los hiperparámetros y su proceso de optimización.
Diferentes conjuntos de imágenes usualmente requieren diferentes combinaciones de valores para sus hiperparámetros en pro de lograr realizar predicciones precisas, sin embargo, el gran número de hiperparámetros entre los cuales escoger dificulta la tarea de elegir a cuáles se les ha de dar prioridad a la hora de aplicar el proceso de optimización.

Dicho proceso, a grandes rasgos, consiste en poner a prueba qué valores de cada hiperparámetro producen los mejores resultados en pruebas mediante repetida experimentación. Lo que implica tener que volver a entrenar nuevamente toda la red por cada valor de cada hiperparámetro con el que se desee experimentar. Siendo este proceso exhaustivo, costoso en cuanto a tiempo y poder computacional, muchos desarrolladores pueden optar por utilizar solamente los valores por defecto para cada hiperparámetro al momento de entrenar sus modelos de clasificación, ya que estos valores teóricamente representan una aproximación al valor ideal para cada hiperparámetro dado un problema de características generales. Sin embargo, no existe una única respuesta para cuántas capas de red son lo ideal, ni cuantas neuronas por capa, o cuál optimizador funcionaría mejor para todos los conjuntos de datos y problemas que pudieran presentársenos. Es por ello que, el proceso de optimización resulta crucial a la hora de encontrar el conjunto de valores que nos permita construir el mejor modelo posible para un conjunto de datos y problema específicos.

Siendo conscientes de las limitaciones que la complejidad del proceso de optimización de hiperparámetros puede implicar para el desarrollo de modelos más precisos, se decidió enfocar la investigación en el análisis y exploración de los efectos de emplear valores para hiperparámetros que definen el algoritmos de entrenamiento distintos de aquellos por defecto en un esfuerzo por medir el impacto que puede tener cada uno de ellos, o todos en conjunto, en el desempeño de un clasificador de armas de fuego en imágenes basado en redes neuronales convolucionales.

1.3 Pregunta de investigación

Tras descubrir la relevancia que tienen las armas de fuego en los altos índices de violencia en regiones determinadas y profundizar en el estudio del estado del arte de los sistemas de detección de dichas armas, fue posible definir la siguiente pregunta de investigación para esta tesis:

¿Qué impacto tienen los hiperparámetros del algoritmo de entrenamiento en el desempeño final de un clasificador de armas de fuego en imágenes basado en redes neuronales convolucionales?

Así mismo, los objetivos general y específicos de esta investigación se mencionan en las siguientes secciones.

1.4 Objetivo general

Comprobar el peso que tienen los hiperparámetros del algoritmo de entrenamiento y su proceso de optimización en el desempeño de un clasificador de armas de fuego en imágenes basado en redes neuronales convolucionales.

1.5 Objetivos específicos

- Inquirir en el estado del arte en cuanto a clasificación de armas de fuego en imágenes por medio de redes neuronales convolucionales.
- Determinar el peso individual de cada hiperparámetro en el desempeño del modelo resultante y elaborar un listado de acuerdo a su importancia en el proceso de optimización.
- Evaluar el posible impacto del uso o ausencia de técnicas de Transfer Learning durante la fase de entrenamiento para este problema en particular.
- Determinar los valores óptimos para los hiperparámetros del algoritmo de entrenamiento para el problema de clasificación de presencia de armas de fuego en imágenes.
- Proponer trabajo futuro para mejorar el rendimiento del sistema de clasificación.

2 Conceptos básicos

2.1 Inteligencia artificial

La inteligencia artificial es una disciplina académica relacionada con la teoría de la computación, su objetivo es emular algunas de las facultades intelectuales humanas en sistemas artificiales [7]. Por lo tanto, para que una máquina inteligente pueda ser considerada como tal, debe poder percibir su entorno y tomar las acciones que maximicen las probabilidades de éxito en un escenario específico. Entre las aplicaciones más notables de la inteligencia artificial se encuentran el procesamiento del lenguaje natural, la demostración de teoremas, robótica, programación automática, análisis de grandes volúmenes de datos, marketing, entre otros.

2.2 Aprendizaje automático (Machine Learning)

Definiendo "Aprendizaje" como el proceso de convertir experiencia en conocimiento, al hablar de aprendizaje automático, la experiencia (Entrada) estaría representada por datos de entrenamiento y el conocimiento (Salida) por alguna capacidad adquirida. Usualmente en la forma de un programa computacional diseñado para realizar alguna tarea específica [8].
Dicho aprendizaje puede clasificarse principalmente en 2 tipos: Supervisado y no supervisado. Siendo lo que los diferencia el etiquetado de los datos de entrada para el aprendizaje supervisado, que provee al algoritmo o modelo con la información necesaria para el posterior etiquetado de nuevos datos que le sean desconocidos. Es por ello que este tipo de aprendizaje comúnmente se utiliza para entrenar clasificadores y regresores. Mientras que, por otro lado, en el aprendizaje no supervisado no existe distinción alguna entre los datos de entrenamiento y los datos de pruebas. Haciendo que su propósito sea el definir un comportamiento general o una versión comprimida de los datos. Lo cual resulta muy útil para tareas que requieran separar el conjunto de datos en subconjuntos con características similares (Clustering) o para problemas de reducción de dimensionalidad.

2.3 Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (Artificial Neural Networks) están inspiradas en las redes neuronales biológicas del cerebro humano. Están constituidas por elementos que básicamente se comportan de forma similar a la neurona biológica en sus funciones más comunes. Permitiéndoles aprender de la experiencia, generalizar de ejemplos previos a nuevos y abstraer las características principales de una serie de datos de entrada [9].

Su unidad más básica se llama neurona artificial y se encarga de recibir entradas con un determinado peso cada una. Después, ha de sumar dichas entradas por su respectivo peso individual asociado y si el resultado excede de determinado umbral,

su valor de salida será igual a uno, mientras que, si es menor, la salida será un cero. Es decir, cada una de estas neuronas trabaja como si de un pequeño procesador se tratase.

Al igual que las neuronas biológicas, las neuronas artificiales también están conectadas, lo que da lugar al concepto de "Capas". Las capas son el nivel que se le agrega a la red cuando la salida de una neurona está conectada a la entrada de otra. Situación que permite aumentar la complejidad de los problemas a resolver. Las primera y última capa de cada red reciben respectivamente los nombres de capas de entrada y de salida. Mientras que todas las capas intermedias se conocen como "Capas ocultas", ya que se desconoce los valores exactos de entrada y salida que poseen. El objetivo de las ANN es la resolución de problemas que resultan demasiado complejos para la programación tradicional.

2.4 Regresores y clasificadores

Se conoce como regresión al conjunto de técnicas utilizadas para predecir un valor numérico asociado a un dato de entrada. En palabras de [10], dado un conjunto D de n datos, formado por pares (xi, yi), i = 1, 2, ..., n, donde xi Rd y yi R, regresión es el o los métodos que han de aplicarse para encontrar la función desconocida f(x) que permita relacionar cada xi con su yi correspondiente con el mayor grado de precisión posible.

Mientras que, lo que diferencia a un regresor de un clasificador es la salida del sistema. Cuando la salida esperada es una lista de categorías, el sistema se denomina clasificador, y, cuando la salida es numérica, se denomina regresor.

2.5 Transferencia de aprendizaje (Transfer Learning)

Es una técnica que, enfocada a redes neuronales, consiste en tomar una red previamente entrenada para realizar cambios apropiados a los parámetros de sus diversas capas con la intención de adaptarla a otro problema y conjunto de datos de naturaleza similar [9]. El principal propósito de la transferencia de aprendizaje es sacar provecho del aprendizaje adquirido por modelos entrenados en conjuntos de datos ampliamente diversos capaces de identificar algunas características universales de los objetos, como orillas y curvas, que resultan relevantes para la mayoría de problemas de clasificación de objetos. Además, la transferencia de aprendizaje también resulta útil cuando no se cuenta con los suficientes datos de entrada para realizar un proceso de entrenamiento adecuado.

Sin embargo, la factibilidad de utilizar esta técnica estará definida por las semejanzas entre el conjunto de datos del que estamos transfiriendo información y aquel de nuestro caso de estudio. De diferir en gran medida, podría resultar contra producente el utilizar transferencia de aprendizaje, ya que no se estaría aportando conocimiento relevante al modelo. Por lo tanto, en esos casos se recomienda entrenar la red desde cero utilizando solamente nuestro conjunto de datos.

3 Redes Neuronales Convolucionales

Una red neuronal convolucional (Convolutional Neural Network) es un tipo de red neuronal profunda especialmente diseñada para reconocimiento de imágenes [9], donde cada imagen empleada en el proceso de aprendizaje es dividida en pequeñas porciones topológicas, las cuales serán individualmente procesadas por filtros dedicados a detectar patrones particulares. En una CNN, cada imagen de entrada es representada como una matriz de pixeles tridimensional, siendo estas tres dimensiones alto, ancho y color respectivamente. La figura 5 muestra la arquitectura básica de una CNN y sus componentes principales, mismos que se describen a continuación.

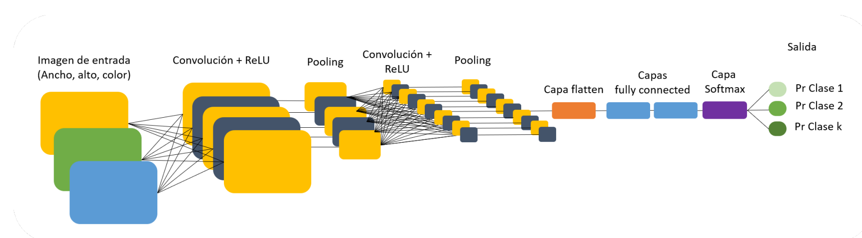


Fig. 5: Arquitectura básica de una red neuronal convolucional.

3.1 Componentes básicos de una red neuronal convolucional.

3.1.1 Capa de convolución

Es un tipo de capa en donde cada neurona está conectada a cierta región del área de entrada llamada "Campo receptivo" [9]. Las capas de convolución utilizan filtros de diferentes dimensiones dentro de los mismos campos receptivos para reconocer imágenes desde diferentes perspectivas. El grupo de neuronas que definen una misma propiedad forma un "Mapa de propiedades".

3.1.2 Capa de pooling

Las CNN también utilizan capas de pooling ubicadas justo después de las capas de convolución. Se encargan de dividir las regiones de convolución en sub regiones y seleccionar un único valor representativo para reducir el costo computacional de las capas subsecuentes e incrementar la robustez de la característica con respecto a su posición espacial [9].

3.1.3 ReLU

Es una función linear que convierte los valores de entrada en números positivos [11]. Es la función de activación por defecto para la mayoría de redes neuronales porque los modelos que la emplean son más sencillos de entrenar y usualmente alcanzan un mejor desempeño.

3.1.4 Capas completamente conectadas (Fully connected)

Conforman las últimas capas de una CNN [12]. Su entrada es la salida reducida de la última capa de pooling o capa convolucional y su función es realizar las operaciones matemáticas que, una vez enviadas a la capa softmax, determinarán la probabilidad para cada clase.

3.1.5 Capa softmax

Es una capa de salida para tareas de multi clasificación que trabaja en conjunto con la función de pérdida entropía cruzada (Cross entropy). Normaliza las salidas de la capa anterior, asegurando que sus sumas sean igual a uno. De esta manera, la salida representará la probabilidad para cada clase [13].

3.1.6 Función de pérdida

Es el error en predicción de una ANN, denotado por la diferencia entre la salida y el valor real de la entrada [14]. La función de pérdida a utilizar dependerá de la naturaleza del problema, siendo las más comunes error medio cuadrático, entropías cruzadas binarias, categóricas y categóricas dispersas.

3.2 Hiperparámetros

Los hiperparámetros habitualmente se definen como la configuración de la estructura de una red neuronal [15]. Por ejemplo, el número de capas de la red, el número de neuronas en cada capa, la función de activación que se emplea, etcétera. Sin embargo, cuando trabajamos con redes neuronales convolucionales, la estructura de la red no es el único factor configurable. Y es precisamente esta capacidad de configuración por parte del desarrollador lo que diferencia a un hiperparámetro de un parámetro. Por lo tanto, en la práctica podemos considerar también como hiperparámetros a aquellos parámetros configurables que definen el algoritmo de entrenamiento, como el optimizador, la tasa de aprendizaje, tamaño de batch y número de épocas. Así mismo, las capas de convolución también poseen sus propios parámetros configurables previo al entrenamiento, por ejemplo, los valores de padding y stride. Sin embargo, esta investigación se encuentra enfocada en el estudio de los hiperparámetros que definen el algoritmo de entrenamiento.

3.2.1 Hiperparámetros que definen el algoritmo de entrenamiento

Para nuestros experimentos fue necesario definir con antelación qué hiperparámetros habriamos de configurar para determinar el impacto de cada uno de ellos en el desempeño de nuestro clasificador. Aquellos seleccionados, y en orden de importancia, se presentan a continuación:

- Optimizador: El optimizador es responsable de actualizar la tasa de aprendizaje y los pesos de las neuronas dentro de una ANN con el propósito de encontrar la función de pérdida mínima [16].

- Tasa de aprendizaje: Representa la magnitud del avance que se da en cada iteración antes de actualizar los pesos de la red [11]. Un valor alto para la tasa de aprendizaje propicia que el modelo aprenda más rápidamente, bajo el riesgo de no encontrar la función de pérdida mínima. Por otro lado, un valor bajo tiene mayores probabilidades de encontrar la función de pérdida mínima, pero requerirá de un mayor número de épocas y, por consiguiente, más tiempo de entrenamiento.

- Tamaño de batch: Es el número de muestras que se le presentan a la red previo a la actualización de los pesos [17]. Un valor pequeño favorece a la velocidad del proceso de aprendizaje y requiere de un menor consumo de memoria, pero la variación en la precisión durante la validación del conjunto de datos será mayor. Por lo tanto, un tamaño más grande alentaría el proceso de aprendizaje y consumiría más memoria, pero la variación en la precisión durante la validación del conjunto de datos se vería reducida, tentativamente haciendo al modelo resultante más robusto ante nuevos datos desconocidos.

- Número de épocas: Es el número de veces que el conjunto de datos completo será mostrado a la red durante el proceso de entrenamiento. Una época significa que el conjunto de datos ha pasado a través de la red una vez, hacia adelante y hacia atrás [15]. Un número demasiado pequeño de épocas podría resultar en un modelo desajustado, debido a que la red no ha aprendido lo suficiente para resolver el problema eficientemente. Sin embargo, especificar demasiadas épocas podría resultar en un sobreajuste, caso en el que el modelo presenta buenos resultados durante el proceso de validación, pero no puede hacerlo con nuevos datos. Es debido a esto que el número de épocas debe de optimizarse para alcanzar el mejor resultado posible.

4 Materiales y Métodos

Para la realización de este trabajo, fue necesario definir de antemano qué arquitecturas de redes neuronales convolucionales habríamos de emplear para llevar a cabo los entrenamientos. Por lo tanto, con el propósito de facilitar la cuantificación de la efectividad de los modelos de clasificación resultantes se decidió, al igual que [6], utilizar AlexNet e Inception (Aunque en vez de su primera versión seleccionamos la tercera) para de esta manera tener una aproximado del desempeño que podríamos esperar por parte de los clasificadores. Ambas arquitecturas se describen a detalle en las siguientes secciones.

4.1 AlexNet

Es una de las primeras redes neuronales convolucionales en alcanzar verdadero éxito al ganar la competencia ILSVRC 2012 al obtener un desempeño sobresaliente en el complejo conjunto de datos de ImageNet utilizando la construcción estándar de las redes neuronales [6]. AlexNet cuenta con 8 capas con parámetros entrenables, de las cuales las primeras 5 son capas convolucionales consecutivas y las ultimas 3 son completamente conectadas (Fully connected). Cada capa de convolución puede estar sucedida por una capa ReLU y, opcionalmente, también por una capa de Max Pooling, cuya región de extensión es de 3 x 3 y su Step rate igual a 2. Requiere que las imágenes de entrada tengan una resolución de 227x227 pixeles.

Resumiendo, nuestra implementación de AlexNet cuenta con un total de 200,114,946 parámetros, de los cuales 200,112,194 son entrenables y 2,752 son no entrenables.

4.2 Inception V3

Es la tercera versión de la arquitectura ganadora de la competencia ILSVRC 2014, GoogLeNet. Está enfocada en reducir en el número de parámetros sin afectar la eficiencia de la red mediante la factorización de convoluciones [18]. Posee 42 capas de profundidad y en vez de utilizar un filtro de 5x5 emplea dos filtros de 3x3 en el primer módulo de incepción. Con este mismo mecanismo, los otros dos módulos también reducen su número de parámetros haciendo menos probable que se presente un escenario de sobre ajuste de la red [19] mientras que a la vez permite que esta crezca a mayor profundidad que sus primeras dos versiones sin descartar sus principales características. Requiere que las imágenes de entrada tengan una resolución de 229x229 pixeles.

Especialmente para este trabajo la estructura de Inception V3 ha sido modificada con la intención de aplicar transferencia de aprendizaje desde el conjunto de datos de ImageNet, añadiendo una capa entrenable extra en la penúltima posición. Por lo que, en resumen, nuestra implementación de Inception V3 cuenta con 43 capas y un total de 23,851,784 parámetros, de los cuales 23,817,352 son entrenables y 34,432 son no entrenables.

4.3 Preparación del entorno de trabajo

Para poder realizar los experimentos requeridos por la investigación hizo falta primero contar con un entorno de trabajo que permitiera que estos se llevaran a cabo de una manera correcta, clara y ordenada, y cuyos resultados fueran fácilmente legibles y presentables. Es por ello que se decidió hacer uso de la interfaz gráfica de Jupyter, en conjunto con las poderosas librerías Tensorflow y Keras para implementar las Redes Neuronales Convolucionales en un lenguaje de programación de alto nivel como lo es Python. En esta sección se detallan los métodos empleados para lograr esta configuración.

4.3.1 Obtención de librerías de aceleración de hardware

Aunque opcional, el uso de aceleración por hardware presenta enormes ventajas al momento de realizar entrenamientos. Mediante la paralelización, es posible utilizar Unidades de Procesamiento Gráfico (GPU) para ejecutar operaciones que normalmente se llevarían a cabo sólo en el procesador, ampliando los recursos disponibles para la ejecución de tareas y consecuentemente reduciendo el tiempo necesario para su compleción.
Dada nuestra configuración de hardware (Véase sección "Especificaciones de hardware y software para la realización de los experimentos") lo primero que hay que hacer es asegurarnos de que los controladores de nuestra GPU estén debidamente actualizados. Después debemos acceder a la página web de NVIDIA [20] y descargar la versión del toolkit de CUDA compatible con la tarjeta gráfica de la que dispongamos. Es importante mencionar que un requisito previo para poder instalar CUDA es que nuestro equipo debe contar con alguna instalación de Visual Studio. Si no es el caso, ingresaremos a la página web de Microsoft [21] para descargarlo. Ya que tengamos el instalador, lo correremos y seleccionaremos los componentes que deseemos instalar. Para nuestro propósito, una instalación básica será más que suficiente.
Regresando a CUDA, si ya concluimos con la descarga y si la instalación de Visual Studio se completó sin problemas, procederemos a instalar. La figura 6 muestra una captura de pantalla del proceso de instalación de CUDA.

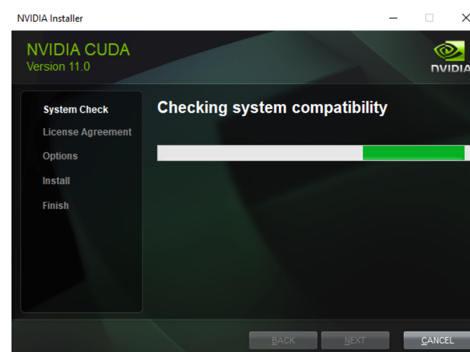
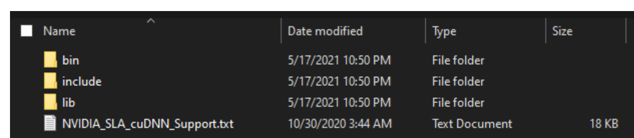


Fig. 6: Proceso de instalación de CUDA 11.0.

Ya que CUDA esté instalado en nuestro sistema, debemos conseguir la librería complementaria CuDNN, que será la que contenga las rutinas necesarias para poder trabajar con Deep Learning. Para ello, accedemos al sitio de NVIDIA [22] y, tras

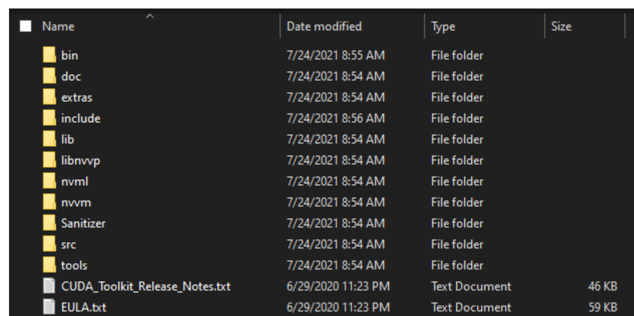
registrarnos en el programa de desarrolladores (Requisito importante) procedemos a descargar la versión de CuDNN compatible con la distribución de CUDA que hayamos instalado. Es importante mencionar que, si las versiones no coinciden, no podremos hacer uso de la aceleración por hardware.

Una vez adquirido el archivo, hay que descomprimirlo y buscar entre sus carpetas los archivos cudnn64_8.dll, ubicado en el folder "bin", cudnn.h, en "include" y cudnn.lib en "lib64". Después copiaremos dichos archivos en las subcarpetas con los mismos nombres dentro del folder de instalación de CUDA. En la figura 7 se puede apreciar el contenido de la librería CuDNN y en la figura 8 podemos ver las carpetas correspondientes en el folder de la instalación de CUDA.



Name	Date modified	Type	Size
bin	5/17/2021 10:50 PM	File folder	
include	5/17/2021 10:50 PM	File folder	
lib	5/17/2021 10:50 PM	File folder	
NVIDIA_SLA_cuDNN_Support.txt	10/30/2020 3:44 AM	Text Document	18 KB

Fig. 7: Contenido de la librería CuDNN.



Name	Date modified	Type	Size
bin	7/24/2021 8:55 AM	File folder	
doc	7/24/2021 8:54 AM	File folder	
extras	7/24/2021 8:54 AM	File folder	
include	7/24/2021 8:56 AM	File folder	
lib	7/24/2021 8:54 AM	File folder	
libnvvp	7/24/2021 8:54 AM	File folder	
nvml	7/24/2021 8:54 AM	File folder	
nvvm	7/24/2021 8:54 AM	File folder	
Sanitizer	7/24/2021 8:54 AM	File folder	
src	7/24/2021 8:54 AM	File folder	
tools	7/24/2021 8:54 AM	File folder	
CUDA_Toolkit_Release_Notes.txt	6/29/2020 11:23 PM	Text Document	46 KB
EULA.txt	6/29/2020 11:23 PM	Text Document	59 KB

Fig. 8: Contenido de la carpeta de instalación de CUDA.

Finalmente, procederemos a validar que se hayan de dado de alta correctamente las variables de entorno en nuestro sistema operativo [23]. Para ello, accederemos al Panel de Control, daremos clic en "Sistema y Seguridad", después "Sistema", y finalmente "Configuración avanzada del Sistema" en el menú lateral. Esto abrirá una ventana como la que se muestra en la figura 9. Una vez ahí, daremos clic en el botón "Environment variables" o "Variables del entorno" y en la siguiente ventana que se mostrará, buscaremos la variable denominada "Path" dentro del grupo "Variables del sistema". La seleccionaremos y daremos clic en el botón "Editar", lo que abrirá una ventana como la que se muestra en la figura 10. Será aquí donde validemos si se referenciaron correctamente las carpetas "bin" y "libnvvp" de la instalación de CUDA, en caso contrario hemos de añadirlas manualmente pulsando el botón "Nuevo".

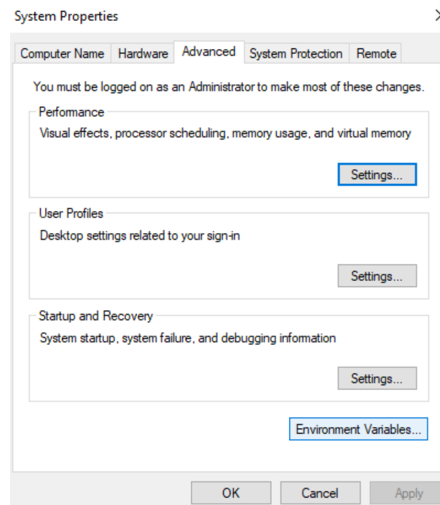


Fig. 9: Ventana para acceder a las variables del entorno de Windows 10.

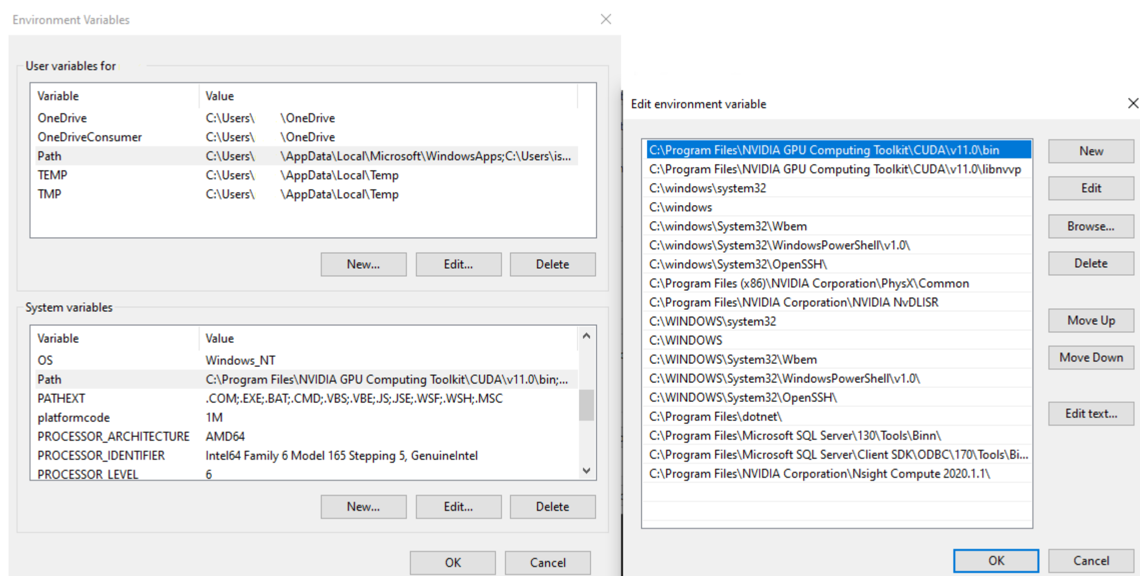


Fig. 10: Variables del entorno. En esta ventana podemos editar las variables del sistema y validar las referencias a las carpetas "bin" y "libnvvp" a través de la variable "Path".

Si completamos exitosamente estas instrucciones, podemos seguir con el segundo paso ya habiendo reunido los componentes necesarios para habilitar la aceleración por hardware.

4.3.2 Instalación de Anaconda, creación de entorno e instalación de librerías

En este paso lo primero que hay que hacer es instalar Anaconda, y para ello accederemos a su sitio web [24] y descargaremos el instalador.
Después, debemos ejecutar el archivo .exe que descargamos para iniciar el setup, como se muestra en la figura 11.

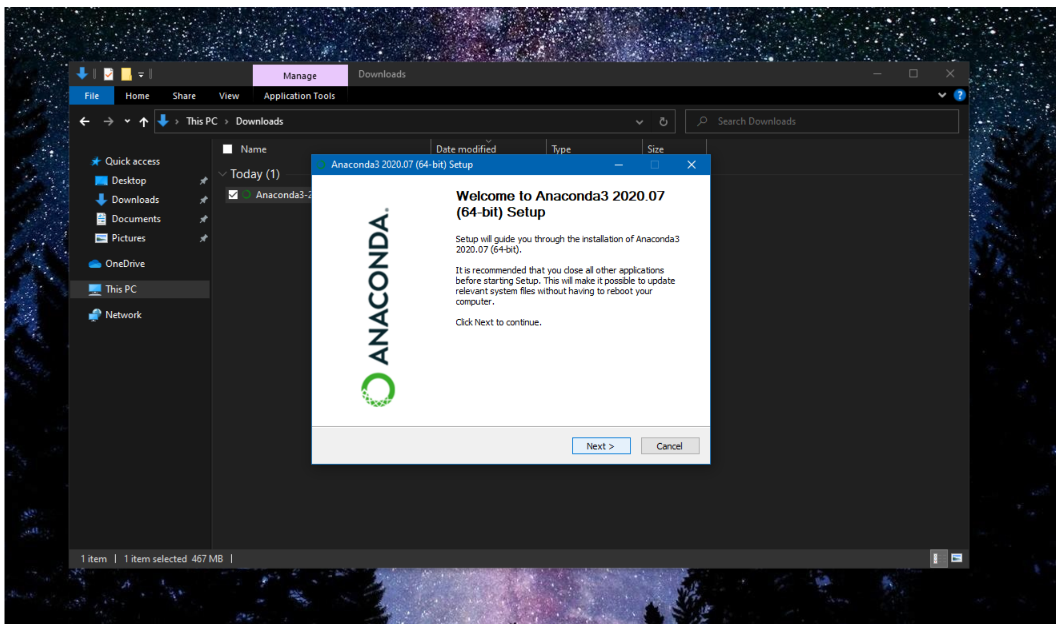


Fig. 11: Iniciando la instalación de Anaconda.

Ya con Anaconda en nuestro sistema, ejecutaremos Anaconda Powershell Prompt para inicializar la consola. Lo siguiente será verificar si existen actualizaciones disponibles de Anaconda, para lo cual emplearemos el comando:

conda update conda

Una vez hayamos verificado que nuestro sistema está debidamente actualizado, procedemos a crear y activar el entorno en el que trabajaremos con aceleración por hardware. Esto mediante los comandos:

conda create –name tf_gpu

conda activate tf_gpu

Con el entorno de trabajo creado y activado, ahora descargaremos e instalaremos las librerías necesarias para poder ejecutar nuestro código. Para ello, ejecutamos uno a uno los siguientes comandos y aceptamos los términos de instalación de cada paquete individual:

conda install -c conda-forge jupyterlab
conda install tensorflow-gpu
conda install pandas

*p*ip install matplotlib
*p*ip install sklearn

Y listo, ya podemos ejecutar el comando "Jupyter notebook" para comenzar a trabajar. El código de la implementación de ambas arquitecturas se encuentra disponible en la sección "Anexos" del documento.

4.4 Valores por defecto de los optimizadores de Keras y rangos a considerar en los experimentos

De acuerdo a la hipótesis de nuestra investigación, el emplear valores para los hiperparámetros que definen el algoritmo de entrenamiento en una red neuronal convolucional distintos a los que se presentan por defecto puede tener un efecto positivo en el desempeño de un modelo de clasificación desarrollado bajo este paradigma. Sin embargo, para poder argumentar en pro de esta hipótesis, es necesario conocer dichos valores por defecto, de manera que sea posible llevar a cabo una comparación precisa y cuantificable. Por lo tanto, estos valores, y los rangos a considerar durante el proceso de experimentación se presentan a continuación.

4.4.1 Valores por defecto de los optimizadores de Keras:

Comenzando con la tasa de aprendizaje, todos los optimizadores de Keras [25] manejan un valor por defecto de 0.001 a excepción de Stochastic Gradient Descent (SGD), cuyo valor por defecto es 0.01. Por otro lado, Keras emplea un estándar para el valor del tamaño de batch de 32 independientemente del optimizador a utilizar. Sin embargo, debido a que no existe un valor por defecto concerniente al número de épocas, decidimos definir un valor estándar de 10 épocas. Esto a razón del gran número de configuraciones que se han de poner a prueba a lo largo de todo el proceso de experimentación.

4.4.2 Rangos a considerar para cada hiperparámetro:

- Optimizador: Adadelta, Adagrad, Adam, Adamax, Ftrl, Nadam, RMSprop, SGD.
- Tamaño de batch: 32, 64, 128, 256.
- Tasa de aprendizaje: 0.01, 0.001, 0.0001.
- Número de épocas: 10, 200.

4.5 Metodología

Previo a la realización de los experimentos fue necesario definir cuál sería la forma de proceder para poder conseguir resultados que aportaran suficiente información relevante a la investigación. Por lo tanto, la metodología empleada durante la fase de experimentación se llevó a cabo tal como se especifica a continuación.

Primero, se hizo uso de los valores por defecto para cada optimizador con ambas arquitecturas para obtener el margen de desempeño a superar con los experimentos de optimización. Luego, se descartaron los optimizadores cuyo desempeño fue inferior

al promedio (Adam, Follow The Regularized Leader y Root Mean Squared Propagation) y después, para medir el peso de la tasa de aprendizaje y el tamaño de batch en el desempeño final del modelo, incrementamos diez veces el valor por defecto de la tasa de aprendizaje y probamos cada optimizador con tamaños de batch de 32, 64, 128 y 256 respectivamente. Después, experimentamos con la tasa de aprendizaje por defecto para los diferentes tamaños de batch y optimizadores. Y al final, hicimos lo propio con un valor para la tasa de aprendizaje diez veces menor que el valor por defecto. Esto concluyó con la primera fase de nuestros experimentos. Cabe mencionar que, durante esta fase, el número de épocas de todas las configuraciones fue igual a 10. Esto con el fin de reducir el tiempo empleado en cada experimento. Además de que cada configuración de hiperparámetros fue probada tanto en AlexNet como en Inception V3.

En la segunda fase, analizamos nuestros resultados para descartar todas aquellas configuraciones de hiperparámetros cuyo desempeño no superó el margen definido durante la primera fase y después, incrementamos el número de épocas a 200. De la misma forma, en esta fase incrementamos a 200 el número de épocas para los valores por defecto para cada optimizador con la intención de generar un nuevo y más preciso margen de desempeño, así, de presentarse alguna mejora tendríamos la certeza de que se debió a nuestro proceso de optimización y no sólo al aumento en el número de épocas.

Ya que tuvimos todos los resultados, la tercera fase consistió en seleccionar las configuraciones que presentaron mayor precisión por cada optimizador y presentarlas en una tabla para comparación. Una vez obtenida la precisión más alta de entre todas las configuraciones, se optimizó el número de épocas para dicha configuración mediante pruebas de aproximación. Dando como resultado el modelo de clasificación óptimo para nuestro conjunto de datos y concluyendo con el proceso de experimentación de la investigación.

4.6 Obtención del conjunto de datos

Otro aspecto clave en la construcción de un modelo de clasificación de alto desempeño basado en redes neuronales convolucionales es el conjunto de imágenes que se empleará para entrenar, validar y realizar pruebas. Para nuestro caso de estudio, era preciso el tomar en cuenta no solo imágenes de pistolas estáticas, si no también escenarios de la vida real en los que dichas pistolas estuvieran siendo empuñadas por personas en diferentes posturas.

Por ello, buscando un balance entre consistencia y utilidad en un ambiente de producción se tomó la decisión de formar un conjunto de datos personalizado con solamente algunas imágenes de otros conjuntos ya existentes. Entre ellos se incluye el estándar de Internet "Weapon-Detection" [26], del cual tomamos 3,000 imágenes para la clase verdadero y otras 2,973 del conjunto “Object-Detection/Pistols” de Roboflow.com [27]. Una vez reunidas las imágenes, estas fueron verificadas una por una para validar que estas no provinieran de videojuegos o dibujos animados, que no contuvieran marcas de agua, filtros, pistolas de juguete, marcos gruesos o cualquier otro tipo de contaminación visual, en un esfuerzo por mantener el conjunto de datos lo más limpio y útil como fuera posible. Este proceso de filtrado

descartó más del 50% de las imágenes que habían sido colectadas, así que, para alcanzar la cantidad previamente planificada (Un total de 3,000 imágenes) se realizó la búsqueda "gun pointed" en el sitio istockphoto.com [28].

Para la clase Falso resultó necesario incluir gran variedad de escenas para proporcionar al modelo la capacidad de generalizar el fondo de la imagen, y además, suficientes imágenes de personas sosteniendo toda clase de objetos en sus manos, para que de esta manera el modelo no aprendiera erróneamente que el atributo principal de la clase verdadero fueran las personas en lugar de las pistolas. Bajo ese criterio, fueron seleccionadas aproximadamente 2,400 imágenes del conjunto de datos COCO [29], perteneciente a Microsoft. Mientras que el resto de imágenes fue obtenido a través de búsquedas manuales en Google Imágenes.
Ejemplos del contenido de ambas clases del conjunto de datos pueden observarse en la figura 12.

Fig. 12: Ejemplos del contenido de cada clase del conjunto de imágenes elaborado.

Resumiendo, nuestro conjunto de datos consiste en un total de 6,000 imágenes. De las cuales, 3,000 pertenecen a la clase "Verdadero" y las otras 3,000 a la clase "Falso". El 70% de estas imágenes corresponden a datos de entrenamiento, mientras que otro 15% a validación y el 15% restante a pruebas.

El formato de archivo de todas las imágenes es ".jpg" y vienen en una variedad de tamaños que va desde 15 Kb hasta 5,000 Kb.

4.7 Especificaciones de hardware y software para la realización de los experimentos

Todos los experimentos se llevaron a cabo utilizando una computadora personal con las siguientes especificaciones de hardware: CPU: Intel(R) Core(TM) i7-10700F CPU

@ 2.90GHz; GPU: NVIDIA GeForce GTX 1660 SUPER GPU @ 1.785GHz, 6.00 GB; RAM: 31.9 GB Utilizables y Disco duro 476 GB Utilizables.

En cuanto a software, se empleó: Windows 10 edición Home como sistema operativo; NVIDIA CUDA versión 11.0.2_451.48 en conjunto con la librería cuDNN 8.0.5 como aceleradores de hardware; Anaconda3 como entorno de desarrollo y administrador de paquetes; Jupyter como frontend, con Keras 2.4.0 y Tensorflow 2.4.0 como backend.

Todo el código fue escrito en lenguaje Python, versión 3.8.8.

5 Resultados y Discusión

5.1 Resultados

Una vez concluidos los experimentos, tuvimos suficiente evidencia para respaldar nuestra hipótesis de que los valores por defecto de los hiperparámetros no siempre proveen la mejor respuesta para un problema en específico. Y que cada uno de ellos, además de tener un peso individual en el desempeño que tendrá un modelo de clasificación posee también una correlación con por lo menos con algún otro hiperparámetro. En la tabla 1 podemos observar los resultados obtenidos tras realizar pruebas em ambas arquitecturas con los valores por defecto para 200 épocas. Mientras que en la tabla 2 se observan los resultados obtenidos al emplear valores optimizados para los hiperparámetros con su respectiva ganancia en precisión en donde aplique.

Tabla 1: Resultados obtenidos al emplear valores por defecto para los hiperparámetros de los optimizadores de Keras. Tanto para AlexNet como para Inception V3.

CNN	Optimizador	Tamaño de batch	Tasa de aprendizaje	Épocas	VP	VN	Precisión
AlexNet	Adadelta	32	0.001	200	240 / 450	418 / 450	73.11%
″	Adagrad	32	0.001	200	255 / 450	409 / 450	73.78%
″	Adamax	32	0.001	200	285 / 450	353 / 450	70.89%
″	Nadam	32	0.001	200	238 / 450	402 / 450	71.11%
″	SGD	32	0.01	200	278 / 450	402 / 450	75.56%
Inception V3	Adadelta	32	0.001	200	377 / 450	443 / 450	91.11%
″	Adagrad	32	0.001	200	375 / 450	444 / 450	91.00%
″	Adamax	32	0.001	200	304 / 450	444 / 450	83.11%
″	Nadam	32	0.001	200	299 / 450	415 / 450	79.33%
″	SGD	32	0.01	200	394 / 450	442 / 450	92.89%

Tabla 2: Resultados obtenidos al emplear valores optimizados para los hiperparámetros de los optimizadores de Keras y su respectivo cambio en la precisión con respecto al uso de valores por defecto. Tanto para AlexNet como para Inception V3.

CNN	Optimizador	Tamaño de batch	Tasa de aprendizaje	Épocas	VP	VN	Precisión	Cambio
AlexNet	Adadelta	32	0.01	200	235 / 450	421 / 450	72.89%	-0.22%
″	Adagrad	128	0.001	200	239 / 450	413 / 450	72.44%	-1.34%
″	Adamax	64	0.0001	200	276 / 450	409 / 450	76.11%	+5.22%
″	Nadam	64	0.0001	200	282 / 450	411 / 450	77.00%	+5.89%
″	SGD	128	0.01	200	272 / 450	409 / 450	75.67%	+0.11
Inception V3	Adadelta	64	0.01	200	386 / 450	443 / 450	92.11%	+1.00%
″	Adagrad	64	0.01	200	397 / 450	444 / 450	93.44%	+2.44%
″	Adamax	128	0.0001	200	394 / 450	444 / 450	93.11%	+10.00%
″	Nadam	256	0.0001	200	389 / 450	444 / 450	92.56%	+13.23%
″	SGD	128	0.01	200	383 / 450	441 / 450	91.56%	-1.33%

A través de los experimentos fuimos capaces de confirmar que, para el problema de clasificación de pistolas en imágenes, el proceso de optimización de los hiperparámetros resulta indispensable, ya que el mejor resultado fue obtenido sin seguir las asunciones y técnicas habituales de la literatura, si no mediante la experimentación exhaustiva. Y, de hecho, al obtener este resultado fue posible notar que 1) El optimizador no suele ser comúnmente utilizado ni es especialmente recomendado por la literatura de Redes Neuronales. 2) El valor de la tasa de aprendizaje no

fue aquel sugerido por defecto y presuntamente el mejor para dicho optimizador. Y 3) El número de épocas resultó ser mucho menor que el empleado habitualmente por otras configuraciones para obtener resultados similares.

De esto podemos concluir que es la combinación de los valores de los hiperparámetros lo que propicia el desempeño óptimo de un modelo de clasificación. Y, por lo tanto, no es posible obviar una sola "Formula ganadora" para cualquier problema y conjunto de datos.

La combinación con mejor desempeño a lo largo de todo el proceso de optimización de hiperparámetros resultó de entrenar bajo la arquitectura Inception V3 con uso de transferencia de aprendizaje desde el conjunto de datos de ImageNet, con Nadam por optimizador, una tasa de aprendizaje de 0.0001, un tamaño de batch de 256 y un total de 13 épocas. La matriz de confusión resultante se presenta en la tabla 3

Tabla 3: Matriz de confusión para Inception V3 con hiperparámetros óptimos.

Real / Predicción	**Positivo**	**Negativo**
Positivo	406	44
Negativo	9	441

Esto significa que el máximo desempeño obtenido en pruebas a través de los experimentos alcanzó una precisión del 94.11%, lo que representa un crecimiento del 10.33% con respecto a los resultados obtenidos al emplear valores por defecto para el optimizador Nadam. Porcentaje bastante significativo.

Por otra parte, para la Red AlexNet (Sin uso de transferencia de aprendizaje) obtuvimos que el optimizador Nadam, con una tasa de aprendizaje de 0.0001, un tamaño de batch de 64 y 200 épocas proporcionaron el mayor desempeño de entre todas las combinaciones. Alcanzando una precisión de 77% durante las pruebas y cuyo margen de mejora respecto a los valores por defecto fue del 5.89%. La matriz de confusión para esta configuración de hiperparámetros se presenta en la tabla 4.

Tabla 4: Matriz de confusión para AlexNet con hiperparámetros óptimos.

Real / Predicción	**Positivo**	**Negativo**
Positivo	282	168
Negativo	39	411

En las figuras 13 y 14 se muestra la reducción en la pérdida durante el proceso de validación y el aumento en la precisión de cada red con su respectiva óptima configuración de hiperparámetros. Mientras que en las figuras 14 y 15 se muestran las curvas ROC correspondientes.

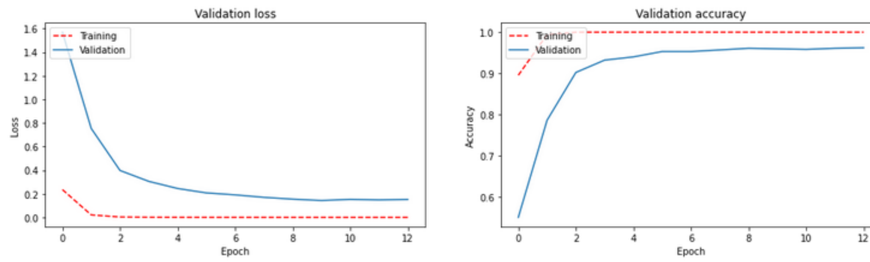


Fig. 13: Pérdida y precisión en el proceso de validación a través de las épocas para Inception V3 con hiperparámetros óptimos.

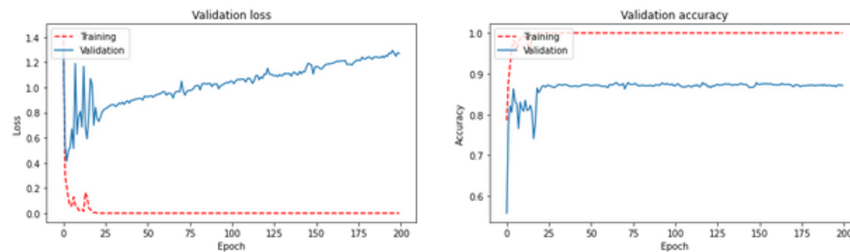


Fig. 14: Pérdida y precisión en el proceso de validación a través de las épocas para AlexNet con hiperparámetros óptimos

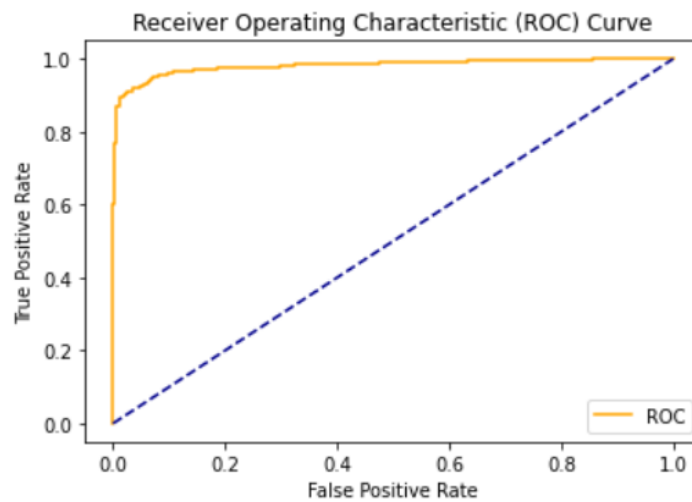


Fig. 15: Curva ROC para la red AlexNet con hiperparámetros optimizados. Su valor de área bajo la curva (AUC) es de 0.8709.

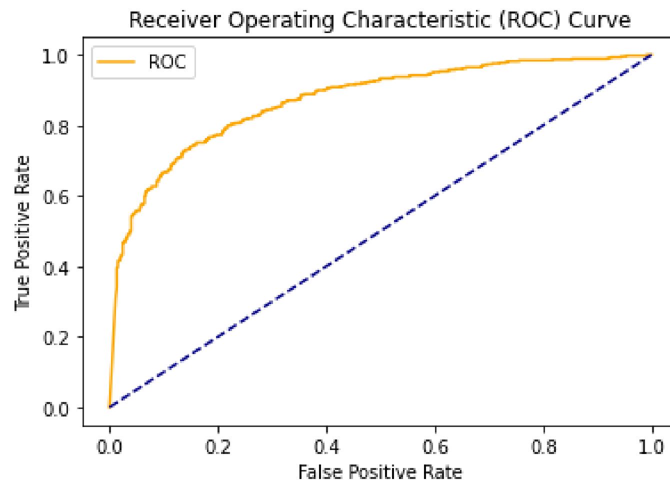


Fig. 16: Curva ROC para la red Inception V3 con hiperparámetros optimizados. Su valor de área bajo la curva (AUC) es de 0.9812.

5.2 Discusión

Con los resultados obtenidos en los experimentos demostramos que los valores por defecto de los optimizadores no siempre proveen el mejor resultado posible para un problema determinado. Tal como sucedió al reducir diez veces el valor de la tasa de aprendizaje por defecto para el optimizador Nadam, lo que ocasionó que pasáramos de tener un modelo cuyo desempeño en pruebas era inferior al promedio, con 67.67% de precisión, a uno de alto rendimiento con 94.11% de precisión durante la fase de pruebas.

Para nuestro caso de estudio particular, la detección de armas de fuego en imágenes con Redes Neuronales Convolucionales, logramos identificar que, descontando al optimizador, el factor más importante en la construcción de un modelo de clasificación óptimo es precisamente elegir la arquitectura de Red Neuronal Convolucional adecuada. Esto debido a que Inception V3 presentó mejores resultados desde el inicio, promediando una precisión del 79.81% por encima del 64.88% alcanzado por AlexNet durante los entrenamientos realizados con los valores por defecto de los optimizadores. Hecho atribuible tanto al uso de Transferencia de Aprendizaje por parte del conjunto de datos Imagenet como al mayor número de capas utilizado por Inception V3. El siguiente hiperparámetro en nuestra lista de importancia es un empate entre la tasa de aprendizaje y el tamaño del Batch, ya que una combinación de un valor pequeño para la tasa de aprendizaje (0.0001) y un valor significantemente alto para el tamaño del Batch (256) presentó los mejores resultados en las pruebas. Sin embargo, es imperativo mencionar que tasas de aprendizaje bajas tienden a alentar el proceso de entrenamiento, y, de la misma manera, un tamaño de Batch grande consumirá mayor cantida de memoria que uno pequeño. Lo que hará el proceso de aprendizaje más computacionalmente costoso.

Al final de nuestra lista encontramos al número de épocas porque, gracias a los resultados de los experimentos podemos concluir que especificar un número grande de épocas no garantiza mejores resultados que uno pequeño debido a la posibilidad

del escenario de sobreajuste. Esto usualmente sucede porque presentar el dataset muchas veces a la red puede ayudar a mejorar la precisión durante el entrenamiento, sin embargo, hace que al modelo le resulte más difícil clasificar correctamente datos que no ha visto antes.

A través de las diferentes fases de los experimentos fue posible discernir algunos patrones entre el desempeño de los optimizadores empleados con uno o varios hiperparámetros específicos. Por ejemplo, Adadelta y Adagrad demostraron efectos positivos al incrementarse su tasa de Aprendizaje de 0.001 a 0.01 independientemente del tamaño de batch empleado. Por otro lado, Adamax y Nadam mostraron mejoría al reducirse su tasa de aprendizaje de 0.001 a 0.0001, y esta mejoría se fue haciendo más notoria conforme crecía también el valor del tamaño del batch. Ambos fenómenos se presentaron tanto en una arquitectura como en la otra, por lo tanto, podemos deducir que estas correlaciones están determinadas en su mayoría por el conjunto de datos.

Otro descubrimiento notable fue que tamaños grandes de Batch suelen funcionar bastante bien con pocas épocas, ya que los experimentos demostraron que la precisión de validación deja de crecer antes cuando especificamos valores grandes para el tamaño del Batch en comparación con valores pequeños. E inclusive, en ocasiones la precisión de validación empieza a reducirse después de determinado número de épocas. Nuevamente, este fenómeno se presentó en los modelos generados bajo ambas arquitecturas.

Sin embargo, la clasificación de armas de fuego en imágenes ya se ha abordado anteriormente en otros trabajos tales como [30], que utiliza un clasificador entrenado bajo la arquitectura VGG-16 especialmente diseñada para minimizar pérdidas en predicción. VGG-16 tiene muchos más parámetros en comparación con AlexNet e Inception V3, bajo el inconveniente de que esto la hace computacionalmente más costosa de implementar. Dicho trabajo alcanza una máxima precisión del 84.21% y el modelo resultante es entonces analizado como un posible sistema de alarma automático. De la misma manera, [31] emplea una arquitectura basada en [32] como clasificador, alcanzando una precisión de 96.26% al realizar pruebas en el conjunto de datos IMFDb. Otro ejemplo se presenta en [6], donde se propone el uso de técnicas de tranferencia de aprendizaje a las ya conocidas arquitecturas de Redes Neuronales Convolucionales AlexNet y GogLeNet (Inception V1). Esta propuesta sobrepasó exitosamente a sus predecesores llegando a alcanzar un 99.2% de precisión para AlexNet y 99.5% para Inception V1 respectivamente durante pruebas realizadas en el conjunto de datos IMFDb.

Estos resultados son realmente alentadores, sin embargo, cuando lidiamos con un tema tan importante y delicado como lo es la seguridad pública siempre hay lugar para mejorar. Por ello, buscando la manera de entender y mejorar los métodos actuales, nuestra propuesta incursiona a un nivel más profundo en los cimientos de las Redes Neuronales Convolucionales, enfocándose en el estudio y análisis de los hiperparámetros que definen el algoritmo de entrenamiento, ya que los trabajos anteriormente mencionados no especifican los valores que fueron empleados ni sus efectos en el desempeño del modelo de clasificación resultante. A su vez, nuestra propuesta considera el uso y ausencia de técnicas de transferencia de aprendizaje y la cantidad de capas en las redes, por ejemplo, AlexNet cuenta con 8 capas mientras que Inception V3 cuenta con 42.

Entre las limitaciones que encontramos al llevar a cabo nuestros experimentos contamos principalmente con los procesos internos de las Redes Neuronales Convolucionales, que, al incluir cierto nivel de aleatoriedad producen un modelo ligeramente distinto en cada ocasión que se ejecuta el entrenamiento aun empleando el mismo conjunto de datos y las mismas configuraciones de hiperparámetros. Lo que hace diferir los resultados de las predicciones de un experimento a otro. La solución más efectiva para esta problemática sería repetir los experimentos varias veces para cada configuración hasta obtener un comportamiento general para dicha configuración calculado a través de una función estadística. Sin embargo, el desarrollo de un modelo óptimo ya requiere un número considerablemente alto de experimentos para probar las diferentes combinaciones de valores para el optimizador, tasa de aprendizaje, tamaño de batch y número de épocas. Por lo tanto, un incremento en la cantidad de experimentos de tal magnitud aumentaría el tiempo y costo computacional requerido por la investigación al punto de hacer dicho intento de normalización inviable.

Finalmente, debido a que los experimentos demostraron que para nuestro conjunto de datos la precisión del modelo durante la fase de entrenamiento dejó de aumentar alrededor de 50 épocas (e inclusive en algunos casos se redujo), valores mayores a 200 épocas no fueron explorados.

6 Conclusiones y trabajo futuro

Concluyendo con la investigación podemos asegurar que, para el caso de detección de armas de fuego en imágenes, el enfoque de clasificación por Redes Neuronales Convolucionales resulta especialmente prometedor. Ya que el modelo final, resultado de los experimentos de optimización, alcanzó una precisión máxima del 94.11% en pruebas. Y esto, existiendo todavía un amplio margen de mejora dada la variedad de posibles aspectos que podrían integrarse a la solución.

Así mismo, se demostró que el proceso de optimización de hiperparámetros resulta de vital importancia en el desarrollo de un modelo de clasificación basado en Redes Neuronales Convolucionales. Puesto que, elegir uno u otro optimizador puede ser la diferencia entre un modelo de alta precisión y un modelo "Tonto". Es decir, un clasificador que carece de la capacidad de identificar alguna de las dos clases. Y eso hablando de un solo hiperparámetro.

Finalmente, como trabajo a futuro en lo que a hiperparámetros se refiere, se propone que, una vez definido el mejor optimizador para nuestro problema en específico (Que en este caso fue Nadam), consideremos también optimizar sus parámetros internos como lo son el Valor Epsilon, la Tasa de descarte y el Momento para explorar sus efectos en el desempeño del modelo de clasificación resultante. A su vez, podría ampliarse el espectro de la investigación mediante la inclusión de los hiperparámetros que definen la estructura de la red, como el Número de capas, las Funciones de activación y la Tasa de descarte y no sólo aquellos que definen el algoritmo de entrenamiento. Esto con el fin de determinar si existe alguna correlación entre ambos tipos de hiperparámetros y el máximo desempeño que un modelo de clasificación basado en Redes Neuronales Convolucionales puede alcanzar.
Mientras que fuera del tema de los hiperparámetros, las posibles direcciones para la continuación del proyecto incluyen:

- La incorporación de sensores (Por ejemplo, detectores de metal) para complementar la solución e incrementar la precisión de las detecciones.
- Elaborar manualmente un conjunto de datos con imágenes de la escena específica en donde se piense implementar el sistema de detección (Por ejemplo, la entrada principal de una escuela o una calle particularmente vulnerable a atentados con armas de fuego). Presentando en él ambas clases con una amplia variedad de contextos, para reducir las variaciones en el fondo imagen a imagen y facilitar la identificación por parte del sistema de las características fundamentales de cada clase durante el proceso de entrenamiento. Contribuyendo así a la diferenciación entre ellas y, por lo tanto, aumentando la precisión.

- Realizar un estudio acerca de cómo afecta el formato de las imágenes del conjunto de datos al desempeño del modelo de clasificación generado por una Red Neuronal Convolucional.

7 Anexos

Se anexa el código Python para la implementación de AlexNet e Inception V3. Así como un script para verificar las versiones tanto de Python como de Keras y Tensorflow. Además de validar la disponibilidad de CUDA en el entorno de Tensorflow para permitir el uso de aceleración por GPU y listar las unidades habilitadas.

7.1 Implementación de AlexNet en Python

```
#Deshabilitamos el uso de aceleración por GPU (Opcional).
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
#Importamos las librerías necesarias.
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pathlib
import datetime
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
#Definimos el número de clases.
output_class_units = 2
#Inicializamos los generadores de datos que contendrán las imágenes de entrenamiento,
validación y pruebas.
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
test_datagen_true = tf.keras.preprocessing.image.ImageDataGenerator()
#Definimos los generadores de entrenamiento y validación con sus respectivos parámet-
ros.
train_generator = train_datagen.flow_from_directory(
directory=r"./Train/", #Especificamos el directorio que contiene las imágenes de en-
trenamiento.
target_size=(227, 227), #Especificamos tamaño de imagen.
color_mode="rgb",
batch_size=128, #Especificamos Batch size.
class_mode="categorical",
shuffle=True, #Habilitamos lectura aleatoria del conjunto de imágenes.
seed=42)
valid_generator = valid_datagen.flow_from_directory(
directory=r"./Validation/",
target_size=(227, 227),
```

```
color_mode="rgb",
batch_size=128,
class_mode="categorical",
shuffle=True,
seed=42)
#Recuperamos el valor de Batch size de los generadores.
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
#Definimos la arquitectura de la red neuronal convolucional.
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(96, (11,11),strides=(4,4), activation='relu', input_shape=(227,
227, 3)),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.MaxPooling2D(2, strides=(2,2)),
tf.keras.layers.Conv2D(256, (11,11),strides=(1,1), activation='relu',padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(384, (3,3),strides=(1,1), activation='relu',padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(384, (3,3),strides=(1,1), activation='relu',padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(256, (3, 3), strides=(1, 1), activation='relu',padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.MaxPooling2D(2, strides=(2, 2)),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(4096, activation='relu'),
tf.keras.layers.Dense(4096, activation='relu'),
tf.keras.layers.Dense(output_class_units, activation='softmax')
])
#Especificamos el optimizador a utilizar con su respectivo valor de Learning Rate:
#opt = keras.optimizers.Adadelta(learning_rate=0.001)
opt = keras.optimizers.Adagrad(learning_rate=0.001)
#opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.Adamax(learning_rate=0.001)
#opt = keras.optimizers.Ftrl(learning_rate=0.001)
#opt = keras.optimizers.Nadam(learning_rate=0.001)
#opt = keras.optimizers.RMSprop(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.01)
#Compilamos el modelo.
model.compile(optimizer=opt, loss="categorical_crossentropy", metrics=['accuracy'])
#Ejecutamos el entrenamiento.
history = model.fit(train_generator,
step_per_epoch=STEP_SIZE_TRAIN,
validation_data=valid_generator,
validation_steps=STEP_SIZE_VALID,
epochs=200 #Especificamos número de épocas.
)
#Inicializamos el proceso de validación.
model.evaluate(valid_generator,
steps=STEP_SIZE_VALID)
#Graficamos la pérdida y la precisión del modelo durante las fases de entrenamiento
y validación.
```

```
fig = plt.figure(figsize=(16,4))
plt1 = fig.add_subplot(121)
plt1.plot(history.history['loss'],color='red', linestyle="–")
plt1.plot(history.history['val_loss'])
plt1.set_title('Validation loss')
plt1.set_ylabel('Loss')
plt1.set_xlabel('Epoch')
plt1.legend(['Training', 'Validation'], loc='upper left')
plt2 = fig.add_subplot(122)
plt2.plot(history.history['accuracy'],color='red', linestyle="–")
plt2.plot(history.history['val_accuracy'])
plt2.set_title('Validation accuracy')
plt2.set_ylabel('Accuracy')
plt2.set_xlabel('Epoch')
plt2.legend(['Training', 'Validation'], loc='upper left')
plt.show()
#Guardamos el modelo.
model.save('AlexNet_Model.h5')
#Cargamos el modelo desde su directorio.
model = tf.keras.models.load_model("AlexNet_Model.h5")
#Definimos generador de pruebas para la clase "False" con sus respectivos parámet-
ros.
test_generator = test_datagen.flow_from_directory(
directory=r"./Test_F/", #Especificamos el directorio que contiene las imágenes de
prueba de la clase "False".
target_size=(227, 227),
color_mode="rgb",
batch_size=1,
class_mode=None,
shuffle=False,
seed=42
)
#Definimos generador de pruebas para la clase "True" con sus respectivos parámet-
ros.
test_generator_true = test_datagen_true.flow_from_directory(
directory=r"./Test_T/", #Especificamos el directorio que contiene las imágenes de
prueba de la clase "True".
target_size=(227, 227),
color_mode="rgb",
batch_size=1,
class_mode=None,
shuffle=False,
seed=42
)
#Recuperamos el valor de Batch size de los generadores de prueba.
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
#Reiniciamos generador.
test_generator.reset()
#Iniciamos proceso de predicción de clase para las imágenes de prueba de la clase
"False".
pred=model.predict(test_generator,
```

```
steps=STEP_SIZE_TEST,
verbose=1)
#Extraemos y damos formato a los resultados de las predicciones del modelo para las imágenes de prueba de la clase "False".
predicted_class_indices=np.argmax(pred,axis=1)
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]
#Generamos Dataframe de Pandas con los resultados.
filenames=test_generator.filenames
results=pd.DataFrame("Filename":filenames,
"Predictions":predictions)
STEP_SIZE_TEST=test_generator_true.n//test_generator_true.batch_size
test_generator_true.reset()
pred_true=model.predict(test_generator_true,
steps=STEP_SIZE_TEST,
verbose=1)
predicted_class_indices_true=np.argmax(pred_true,axis=1)
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions_true = [labels[k] for k in predicted_class_indices_true]
filenames_true=test_generator_true.filenames
results_true=pd.DataFrame("Filename":filenames_true,
"Predictions":predictions_true)
#Quitamos restricciones a Pandas para poder mostrar todos los datos.
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
#Imprimimos resultados de las predicciones hechas a la clase "False" del conjunto de pruebas.
results
#Imprimimos resultados de las predicciones hechas a la clase "True" del conjunto de pruebas.
results_true
#Contamos e imprimimos verdaderos positivos (True Positives).
tp = predictions_true.count('True')
tp
#Contamos e imprimimos verdaderos negativos (True Negatives).
tn = predictions.count('False')
tn
#Contamos e imprimimos falsos positivos (False Positives).
fp = predictions.count('True')
fp
#Contamos e imprimimos falsos negativos (False Negatives).
fn = predictions_true.count('False')
fn
#Definimos curva ROC, gráficamos y calculamos valor de AUC (Area Under the Curve).
def plot_roc_curve(fpr, tpr):
plt.plot(fpr, tpr, color='orange', label='ROC')
```

```
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='–')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
y_false = np.full((450,1), [0])
y_true = np.full((450,1), [1])
actual_values = np.concatenate((y_false, y_true))
actual_predictions = np.concatenate((pred, pred_true))
probs = actual_predictions[:, 1]
auc = roc_auc_score(actual_values, probs)
print('AUC: %.4f' % auc)
```

7.2 Implementación de Inception V3 en Python

```
#Deshabilitamos el uso de aceleración por GPU (Opcional).
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
#Importamos las librerías necesarias.
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pathlib
import datetime
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import random
from matplotlib.pyplot import imshow
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.imagenet_utils import preprocess_input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Activation
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Model
#Definimos el número de clases.
output_class_units = 2
#Inicializamos los generadores de datos que contendrán las imágenes de entrenamiento,
validación y pruebas.
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
test_datagen_true = tf.keras.preprocessing.image.ImageDataGenerator()
#Definimos los generadores de entrenamiento y validación con sus respectivos parámet-
ros.
```

```
train_generator = train_datagen.flow_from_directory(
directory=r"./Train/", #Especificamos el directorio que contiene las imágenes de entrenamiento.
target_size=(229, 229), #Especificamos tamaño de imagen.
color_mode="rgb",
batch_size=256, #Especificamos Batch size.
class_mode="categorical",
shuffle=True, #Habilitamos lectura aleatoria del conjunto de imágenes.
seed=42)
valid_generator = valid_datagen.flow_from_directory(
directory=r"./Validation/",
target_size=(229, 229),
color_mode="rgb",
batch_size=256,
class_mode="categorical",
shuffle=True,
seed=42)
#Descargamos la arquitectura Inception V3 pre entrenada con el dataset de ImageNet desde Keras y la almacenamos en una variable para su posterior modificación.
vgg = tf.keras.applications.InceptionV3(
include_top=True,
weights="imagenet",
input_tensor=None,
input_shape=None,
pooling=None,
classes=1000,
classifier_activation="softmax",
)
#Referenciamos la capa de entrada de la red neuronal convolucional.
inp = vgg.input
#Creamos una nueva capa softmax con el número de clases que asignamos a nuestro problema.
new_classification_layer = Dense(output_class_units, activation='softmax')
#Conectamos nuestra nueva capa a la penúltima capa de la arquitectura y la referenciamos.
out = new_classification_layer(vgg.layers[-2].output)
#Creamos una nueva red entre las variables entrada y salida para que esta contenga la capa softmax que agregamos.
model = Model(inp, out)
#Congelamos los pesos de todas las capas excepto de la última (Opcional).
#for l, layer in enumerate(model.layers[:-1]):
layer.trainable = False
#Nos aseguramos de que la última capa sea entrenable (No congelada).
for l, layer in enumerate(model.layers[-1:]):
layer.trainable = True
#Recuperamos el valor de Batch size de los generadores.
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
#Especificamos el optimizador a utilizar con su respectivo valor de Learning Rate:
#opt = keras.optimizers.Adadelta(learning_rate=0.001)
#opt = keras.optimizers.Adagrad(learning_rate=0.001)
```

```
#opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.Adamax(learning_rate=0.001)
#opt = keras.optimizers.Ftrl(learning_rate=0.001)
opt = keras.optimizers.Nadam(learning_rate=0.001)
#opt = keras.optimizers.RMSprop(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.01)
#Compilamos el modelo.
model.compile(optimizer=opt, loss="categorical_crossentropy", metrics=['accuracy'])
#Ejecutamos el entrenamiento.
history = model.fit(train_generator,
step_per_epoch=STEP_SIZE_TRAIN,
validation_data=valid_generator,
validation_steps=STEP_SIZE_VALID,
epochs=13 #Especificamos número de épocas.
)
#Inicializamos el proceso de validación.
model.evaluate(valid_generator,
steps=STEP_SIZE_VALID)
#Graficamos la pérdida y la precisión del modelo durante las fases de entrenamiento
y validación.
fig = plt.figure(figsize=(16,4))
plt1 = fig.add_subplot(121)
plt1.plot(history.history['loss'],color='red', linestyle="–")
plt1.plot(history.history['val_loss'])
plt1.set_title('Validation loss')
plt1.set_ylabel('Loss')
plt1.set_xlabel('Epoch')
plt1.legend(['Training', 'Validation'], loc='upper left')
plt2 = fig.add_subplot(122)
plt2.plot(history.history['accuracy'],color='red', linestyle="–")
plt2.plot(history.history['val_accuracy'])
plt2.set_title('Validation accuracy')
plt2.set_ylabel('Accuracy')
plt2.set_xlabel('Epoch')
plt2.legend(['Training', 'Validation'], loc='upper left')
plt.show()
#Guardamos el modelo.
model.save('GoogLeNet_Model.h5')
#Cargamos el modelo desde su directorio.
model = tf.keras.models.load_model('GoogLeNet_Model.h5')
#Definimos generador de pruebas para la clase "False" con sus respectivos parámet-
ros.
test_generator = test_datagen.flow_from_directory(
directory=r"./Test_F/", #Especificamos el directorio que contiene las imágenes de
prueba de la clase "False".
target_size=(229, 229),
color_mode="rgb",
batch_size=1,
class_mode=None,
shuffle=False,
seed=42
```

```
)
#Definimos generador de pruebas para la clase "True" con sus respectivos parámetros.
test_generator_true = test_datagen_true.flow_from_directory(
directory=r"./Test_T/", #Especificamos el directorio que contiene las imágenes de prueba de la clase "True".
target_size=(229, 229),
color_mode="rgb",
batch_size=1,
class_mode=None,
shuffle=False,
seed=42
)
#Recuperamos el valor de Batch size de los generadores de prueba.
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
#Reiniciamos generador.
test_generator.reset()
#Iniciamos proceso de predicción de clase para las imágenes de prueba de la clase "False".
pred=model.predict(test_generator,
steps=STEP_SIZE_TEST,
verbose=1)
#Extraemos y damos formato a los resultados de las predicciones del modelo para las imágenes de prueba de la clase "False".
predicted_class_indices=np.argmax(pred,axis=1)
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]
#Generamos Dataframe de Pandas con los resultados.
filenames=test_generator.filenames
results=pd.DataFrame("Filename":filenames,
"Predictions":predictions)
STEP_SIZE_TEST=test_generator_true.n//test_generator_true.batch_size
test_generator_true.reset()
pred_true=model.predict(test_generator_true,
steps=STEP_SIZE_TEST,
verbose=1)
predicted_class_indices_true=np.argmax(pred_true,axis=1)
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions_true = [labels[k] for k in predicted_class_indices_true]
filenames_true=test_generator_true.filenames
results_true=pd.DataFrame("Filename":filenames_true,
"Predictions":predictions_true)
#Quitamos restricciones a Pandas para poder mostrar todos los datos.
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
#Imprimimos resultados de las predicciones hechas a la clase "False" del conjunto de pruebas.
```

```
results
#Imprimimos resultados de las predicciones hechas a la clase "True" del conjunto de pruebas.
results_true
#Contamos e imprimimos verdaderos positivos (True Positives).
tp = predictions_true.count('True')
tp
#Contamos e imprimimos verdaderos negativos (True Negatives).
tn = predictions.count('False')
tn
#Contamos e imprimimos falsos positivos (False Positives).
fp = predictions.count('True')
fp
#Contamos e imprimimos falsos negativos (False Negatives).
fn = predictions_true.count('False')
fn
#Definimos curva ROC, gráficamos y calculamos valor de AUC (Area Under the Curve).
def plot_roc_curve(fpr, tpr):
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='–')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
y_false = np.full((450,1), [0])
y_true = np.full((450,1), [1])
actual_values = np.concatenate((y_false, y_true))
actual_predictions = np.concatenate((pred, pred_true))
probs = actual_predictions[:, 1]
auc = roc_auc_score(actual_values, probs)
print('AUC: %.4f' % auc)
```

7.3 Verificar versiones

```
import tensorflow as tf #Importamos Tensorflow.
print(tf.__version__) #Imprimimos nuestra versión actual de Tensorflow.
from tensorflow import keras as k #Importamos Keras desde nuestra distribución de Tensorflow.
print(k.__version__) #Imprimimos nuestra versión actual de Keras.
tf.test.is_built_with_cuda() #Validamos que la librería CUDA esté incorporada a Tensorflow.
tf.test.is_gpu_available(cuda_only=False, min_cuda_compute_capability=None) #Revisamos si hay GPUs disponibles.
```

```
from platform import python_version #Importamos nuestra versión de Python.
print(python_version()) #Imprimimos.
```

Bibliografía

[1] UNODC. *Global Study on Homicide 2019*. July 2019. URL: https://www.unodc.org/unodc/en/data-and-analysis/global-study-on-homicide.html.

[2] UNODC. *Global Study on Firearms Trafficking 2020*. Mar. 2020. URL: https://www.unodc.org/unodc/en/firearms-protocol/firearms-study.html.

[3] INEGI. *Mortalidad. Conjunto de datos: Defunciones por homicidios*. Oct. 2019. URL: https://www.inegi.org.mx/sistemas/olap/proyectos/bd/continuas/mortalidad/defuncioneshom.asp?s=est.

[4] INEGI. *Clasificación estadística de delitos 2012*. 2013. URL: https://www.inegi.org.mx/contenidos/productos/prod_serv/contenidos/espanol/bvinegi/productos/clasificadores/clasi_delito_12/CED_2012.pdf.

[5] INEGI. *ENVIPE 2019*. Sept. 2019. URL: https://www.inegi.org.mx/programas/envipe/2019/.

[6] Mai Mai Mohamed Galab, Ahmed Taha, and Hala Zayed. "Automatic Gun Detection Approach for Video Surveillance". In: *International Journal of Sociotechnology and Knowledge Development* 12 (Jan. 2020), pp. 49–66. DOI: 10.4018/IJSKD.2020010103.

[7] Raúl Benítez et al. *Inteligencia artificial avanzada*. Editorial UOC, 2014.

[8] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[9] G. Zaccone and M.R. Karim. *Deep Learning with TensorFlow: Explore neural networks and build intelligent systems with Python, 2nd Edition*. Packt Publishing, 2018. ISBN: 9781788831833. URL: https://books.google.com.mx/books?id=zZ1UDwAAQBAJ.

[10] Carlos Pardo Aguilar et al. "Minería de Datos y combinación de regresores". In: (2016).

[11] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (2021), p. 53. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8. URL: https://doi.org/10.1186/s40537-021-00444-8.

[12] Hamed Habibi Aghdam and Elnaz Jahani Heravi. "Guide to convolutional neural networks". In: *New York, NY: Springer* 10.978-973 (2017), p. 51.

[13] N KETKAR. "Deep learning with python, a hands-on introduction, apress edition, 160p". In: (2017).

[14] Hang Zhao et al. "Loss functions for image restoration with neural networks". In: *IEEE Transactions on computational imaging* 3.1 (2016), pp. 47–57.

[15] Erik Bochinski, Tobias Senst, and Thomas Sikora. "Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms". In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 3924–3928. DOI: 10.1109/ICIP.2017.8297018.

[16] Zewen Li et al. "A survey of convolutional neural networks: analysis, applications, and prospects". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[17] Matthias Feurer and Frank Hutter. "Hyperparameter optimization". In: *Automated machine learning*. Springer, Cham, 2019, pp. 3–33.

[18] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[19] Leslie N Smith. "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay". In: *arXiv preprint arXiv:1803.09820* (2018).

[20] *CUDA Toolkit*. 2021. URL: https://developer.nvidia.com/cuda-toolkit.

[21] *Microsoft Visual Studio*. 2021. URL: https://visualstudio.microsoft.com/es/free-developer-offers/.

[22] *NVIDIA cuDNN*. 2021. URL: https://developer.nvidia.com/cudnn.

[23] *Installing Tensorflow with CUDA, cuDNN and GPU support on Windows 10*. 2021. URL: https://towardsdatascience.com/installing-tensorflow-with-cuda-cudnn-and-gpu-support-on-windows-10-60693e46e781.

[24] *Anaconda Individual Edition*. 2021. URL: https://www.anaconda.com/products/individual.

[25] Keras API. *Optimizers*. URL: https://keras.io/api/optimizers/.

[26] *weapon$_d$etection$_d$ataset*. 2019. URL: https://www.kaggle.com/abhishek4273/gun-detection-dataset.

[27] *Pistols Dataset*. 2020. URL: https://public.roboflow.com/object-detection/pistols/1.

[28] *gun pointed*. 2021. URL: https://www.istockphoto.com/es/search/2/image?page=3&phrase=gun%20pointed.

[29] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[30] Roberto Olmos, Siham Tabik, and Francisco Herrera. "Automatic Handgun Detection Alarm in Videos Using Deep Learning". In: *CoRR* abs/1702.05147 (2017). arXiv: 1702.05147. URL: http://arxiv.org/abs/1702.05147.

[31] Rodrigo Kanehisa and Areolino Neto. "Firearm Detection using Convolutional Neural Networks". In: *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*. Jan. 2019, pp. 707–714. DOI: 10.5220/0007397707070714.

[32] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.