Universidad Autónoma de Ciudad Juárez

Instituto de Ingeniería y Tecnología
Departamento de Ingeniería Industrial y Manufactura
Maestría en Ingeniería en Manufactura

“Evaluación de un algoritmo “Faster R-CNN” entrenado sobre un conjunto de datos locales para la detección de señales de tráfico”

Proyecto para obtener el grado de Maestro en Ingeniería en Manufactura

Juan Manuel Bobadilla Ramos

“Becado (a) por el Consejo Nacional de Ciencia y Tecnología”

Bajo la Dirección del
Dr. Angel Israel Soto Marrufo

Ciudad Juárez, Chih
Noviembre 2021

Índice general

Índice de tablas

Índice de figuras

1. Introducción

1.1 Antecedentes

La inteligencia artificial y sus diversas técnicas forman un campo amplio con aplicaciones en áreas como el reconocimiento de imágenes, de patrones y de voz [2]; particularmente, la aplicación de técnicas y algoritmos de reconocimiento de imágenes en el área automotriz [3] ha permitido automatizar tareas entre las que destacan: los sistemas auxiliares de cambio y conservación de línea, los de detección de objetos en carretera o de peatones [4] y los de asistencia de aparcamiento [5].

Debido a esta variedad de aplicaciones, la inclusión cada vez mayor de funciones automatizadas en los vehículos se ha vuelto una constante en el portafolio de los productos de los grandes fabricantes automotrices [6]; la cual, ha provocado un aumento en el número de sensores y cámaras y, derivado de esto, la generación de información que se ha recopilado en grandes volúmenes de datos, cuyo acceso ha ido, también, en aumento. La disponibilidad de esta gran cantidad de datos (*big data*) ha permitido la aplicación de un tipo particular de método de inteligencia artificial conocido como *deep learning* [1]. En consecuencia, la cada vez más constante utilización de las aplicaciones del *deep learning* motivó a las instituciones académicas y compañias tecnológicas como plantean en [1] a invertir grandes cantidades de recursos técnicos y económicos en investigación y desarrollo. Esto impulsó la generación de algoritmos y métodos a partir de conjuntos de datos locales, es decir específicos del lugar donde se generaron.

Sin embargo, los métodos utilizados tradicionalmente no resultan ser los mejores fuera de las áreas donde se recopilan los datos, como plantean Kulkarni, Dhavalikar y Bangar, [1]. Es por ello, que surge el interés de recabar información en conjuntos de entrenamiento y validación con los que entrenar un algoritmo, considerando los recursos de información (luces de tráfico, señalizaciones viales) disponibles en Ciudad Juárez.

El *deep learning* comprende la clase de técnicas de *machine learning* que explotan varias capas de procesamiento de información no lineal para la extracción y transformación de características con o sin supervisión, además de su análisis de patrones y clasificación [7, 8].

La característica que diferencía al *deep learning* es que está basado en aprender a través de varios niveles de representación los cuales corresponden a una jerarquía de características o factores, en donde los conceptos de alto nivel están definidos a partir de los de bajo nivel.

1.2 Descripción o Planteamiento del Problema

El *deep learning* ha impulsado un avance sustancial en el área de la visión computarizada [9]. Estos avances en el desempeño de los algoritmos de reconocimiento visual han llegado a superar el desempeño humano en algunas tareas como identificación y clasificación de imágenes [10, 11]. Este alto desempeño y su ejecución a una velocidad mayor a la humana posibilita la aplicación de estas tecnologías para lidiar con situaciones que exceden las capacidades de reacción humanas [12].

Un caso particular se encuentra en los accidentes de tráfico, de acuerdo con cifras de INEGI en México, [13], durante el año 2019 hubo 362,586 accidente de tránsito terrestre causantes de 4,125 muertos y 91,713 heridos.

Existen muchos factores con un efecto en el número de accidentes, sin embargo y a pesar de que existan señalizaciones adecuadas, una tercera parte de ellos ocurren en las intersecciones dentro de las ciudades [14, 15]. La cantidad de accidentes puede reducirse utilizando técnicas basadas en el *deep learning* como con algoritmos basados en *CNN* [16, 17], sin embargo, se requiere que estas técnicas estén entrenadas de manera que el error entre las predicciones realizadas y las imágenes reales (señalizaciones) sea el mínimo posible.

De acuerdo con los resultados de [1], al utilizar una conjunto de entrenamiento conformado por datos locales se logró alcanzar un desempeño (función de perdida) en torno al orden de 0.01 % en contraste con el umbral comúnmente aceptado de 0.05 %.

1.3 Objetivos

1.3 Objetivo general

Evaluar un algoritmo de *deep learning* comparando conjuntos de entrenamiento conformados por datos locales en contraste con datos generados en otras partes del mundo para medir su desempeño entre las dos bases de datos en igualdad de condiciones.

1.3 Objetivos particulares

- Recopilar imágenes de los distintos tipos de señales viales utilizados en Ciudad Juárez.
- Construir un conjunto de entrenamiento y uno de validación compuesto por los datos locales.
- Entrenar los algoritmos de aprendizaje profundo con los conjuntos formados.
- Evaluar los resultados obtenidos al ejecutar los algoritmos utilizando los conjuntos de datos tradicionales y los conjuntos de datos locales.

1.4 Hipótesis

Un algoritmo basado en *deep learning* entrenado y validado con datos de Ciudad Juárez reporta un mejor desempeño que el mismo entrenado con un conjunto de datos recopilados en otra parte del mundo.

1.5 Justificación

La omisión de las señalizaciones viales representa un serio peligro para los tripulantes de un vehículo y para las personas en las inmediaciones. La utilización de un algoritmo basado en *deep learning* puede contribuir en la reducción del número de accidentes ocasionados por las omisiones o reacciones tardías a las señalizaciones viales.

De ahí que resulta de interés contar con un algoritmo de identificación de señales viales optimizado en función de la localidad donde el auto se conduzca, esto de acuerdo con lo reportado por Kulkarni, Dhavalikar y Bangar, [1].

1.6 Alcance

Se conformará una base de datos con distintas señales viales de la localidad, lo que permitirá poder realizar la evaluación del algoritmo en función de su base de datos. No se considera en el alcance de este proyecto la construcción de un algoritmo propio sino la evaluación de algoritmos existentes.

Tampoco se realizará la integración ni evaluación en vehículos en movimiento. Las evaluaciones se realizarán directamente en el software correspondiente.

2. Revisión de Literatura

2.1 *Machine Learning*

El aprendizaje máquina, normalmente referido por el anglicismo *machine learning* también nombrado como aprendizaje automatizado o redes neuronales artificiales (debido a su perspectiva inicial neuro-científica [18, 19]) comprende las ramas de la ciencia cognitiva y la inteligencia artificial [20], encargadas de los algoritmos computacionales designados para emular la inteligencia humana por medio del aprendizaje obtenido a partir del medio circundante [21], de la experiencia [22,23] o derivado de datos [24] para poder realizar predicciones.

Algunos términos comunes en la literatura relacionada al *machine learning* se definen a continuación:

Ejemplo de entrenamiento: son los datos aplicados al algoritmo de aprendizaje para entrenarlo [25].

Entrenamiento: Es el ajuste del modelo [24, 26], el proceso en el cual se identifican los patrones de los ejemplos de entrenamiento (datos).

Característica (*feature*): es una propiedad de cada instancia del conjunto de datos [25], las entradas utilizadas para realizar la predicción o clasificación [27]

Objetivo (*target*): también conocido como salida, es el atributo que se desea predecir.

Función de perdida (o función de costo): es una función que evalúa la diferencia entre las predicciones realizadas por el algoritmo y los valores reales de las observaciones utilizadas durante el entrenamiento [28].

mAP: por sus siglas en inglés *Mean Average Precision*, se define primero a la precisión como el cociente:

$$\text{Precisión} = \frac{TP}{TP+FP}$$

donde TP es un acierto (en inglés *true positive*) y FP es un falso positivo.

Mientras que la razón positiva de "positivos" verdaderos, TPR, está definida por:

$$TPR = \frac{TP}{TP+FN}$$

Una compensación entre la precisión y la razón positiva de positivos reales debe realizarse de acuerdo a las necesidades propias, ajustando el umbral de la función softmax de la última capa del modelo. Estos dependerán del valor IoU, usualmente se define como verdadero positivo

a aquellos con IoU mayor a 0.5 y falso positivo en caso de que el IoU fuera menor a 0.5.

Los falsos positivos, cuando no hay detección de ningún objeto o tiene clasificación errónea.

Por lo tanto el mAP para la clasificación de objetos es la media de la precisión promedio calculada para todas las clases.

2.2 Tipos principales de aprendizaje

De manera general, el *machine learning* puede ser categorizado de acuerdo con varios enfoques, por ejemplo, en función de la optimización de las predicciones obtenidas con los datos de entrenamiento [29]. Sin embargo, es posible definir a partir de los métodos de *machine learning* tres categorías principales: **aprendizaje supervisado**, **no supervisado** y **reforzado**.

- El aprendizaje supervisado consiste en las técnicas de construcción de modelos predictivos que *aprenden* a partir de una cantidad grande de ejemplos de entrenamiento [30].
- Por el contrario, cuando se habla del aprendizaje no supervisado los datos no se encuentran etiquetados y el propio algoritmo trata de clasificar la información, por si mismo obtener representaciones [31].
- El aprendizaje reforzado consiste en una sucesión de interacciones entre un *agente* de aprendizaje y su entorno, que aprenderá a través de un proceso de recompensas-castigos resultantes de sus acciones [32, 33].

2.3 Clasificación por modelo

Las redes neuronales artificiales consisten en una representación matemática de unidades de procesamiento conectadas llamadas neuronas artificiales [34]. Este modelo surge de la imitación del funcionamiento del cerebro humano [35, 36].

Este modelo define la neurona como una unidad de procesamiento central, estas neuronas forman una capa de entrada a la que arriban un conjunto de señales de entrada posteriormente multiplicadas por un parámetro de ponderación y, en una capa oculta, ser sumadas entre sí agregando un término de sesgo para, posteriormente, activar una función de transferencia que produce la salida de la neurona en la última capa [37, 38]. Esta red de neuronas y conexiones, con una capa de entrada, una capa oculta y una capa de salida donde las neuronas y las capas adyacentes están complementarte conectadas es conocida como perceptrón sencillo. En caso de que exista más de una capa oculta se define a esta red como un perceptrón multicapa [39].

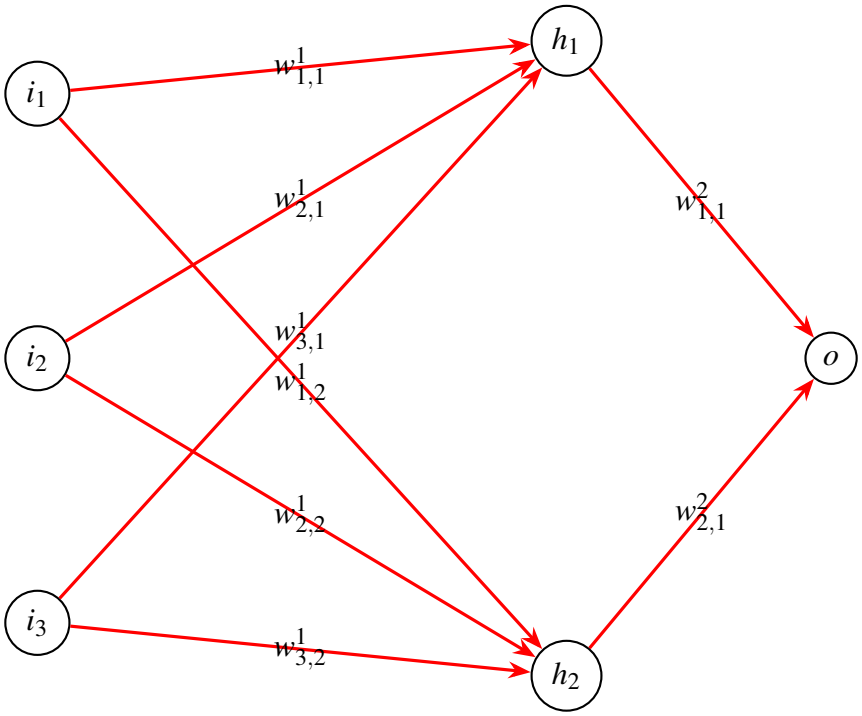


Figura 2.1: Ejemplo de perceptrón de una sola capa, con tres entradas i_j y una salida o.

Se conoce como *deep learning* a la red neuronal cuyo número de capas ocultas es mayor o igual a dos y estas trabajan en conjunto con arquitecturas y algoritmos altamente optimizados [40, 41].

Cuando una red neuronal contiene capas convolucionales se dice que es convolucional, CNN por sus siglas en inglés [42]. Las capas de convolución pueden visualizarse como la expresión:

$$o(t) = (x \times w)(t) = \int x(a)w(t-a)da$$

donde $x(t)$ es la entrada, $w(t)$ la ponderación, núcleo o kernel, que funge como un escáner [43], y $o(t)$ la salida resultante por la operación de la convolución [44], este operador combina de manera puntual un conjunto de datos con otro [45].

2.4 Aplicación en clasificación de imágenes

Aunque las redes neuronales convolucionales han sido, tradicionalmente, identificadas como de gran desempeño en las tareas de clasificación de imágenes con una sola etiqueta [46,47], sin embargo, la naturaleza de muchos tipos de aplicaciones para detección de objetos está orientada hacia las imágenes con multi-etiquetas [48].

Alternativas para solucionar el problema, como la utilización de perceptrones multicapas en redes neuronales profundas implementados en la API de Google, Tensorflow [49, 50].

Entre estas aplicaciones se encuentra el reconocimiento del estado de los semáforos y otras señales de tráfico y su clasificación [51–53].

2.4 Arquitectura "Faster R-CNN"

La arquitectura Faster R-CNN [54] contiene dos redes principales [55] y una red adicional extractora de características, en la imagen 2.2 puede observarse la arquitectura correspondiente con este modelo:

1. Red de características
2. Red de región propuesta (RPN por sus siglas en inglés)
3. Red de detección de objetos

Red de características

Normalmente consiste en una red clasificadora de imágenes previamente entrenada que extrae las principales características (*features*) de la imagen, que luego serán utilizadas en el la red de regiones propuestas.

RPN

En esta arquitectura, se utilizan redes de regiones propuestas que comparten capas convolucionales con redes que, en el momento en que se propuso la arquitectura, consistentes en redes de detección de objetos del estado del arte.

Añadiendo unas capas convolucionales a la arquitectura existente "Fast RCNN" que retornan simultáneamente cotas para las regiones y puntuaciones de cada locación en una cuadrícula regular.

Se realiza un ajuste fino de la tarea de propuesta de regiones y de la tarea de detección de objetos es realizado de manera alternada.

En este paso se realiza un reescalado de la imagen, que no excederá 1000 pixeles en su lado mayor.

El concepto de "ancla", que consiste en una región rectangular [56], es utilizado asignándole un valor de etiqueta positivo en caso de las anclas, o anclajes, con la intersección sobre unión (IoU, por sus siglas en inglés) más alta se superpongan con una caja (con valor verdadero o un positivo real [54]) o cuando un ancla tiene un IoU mayor que 0.7 con cualquier caja positivo real. En cambio, un ancla tiene valor negativo si la calificación de IoU está debajo de 0.3.

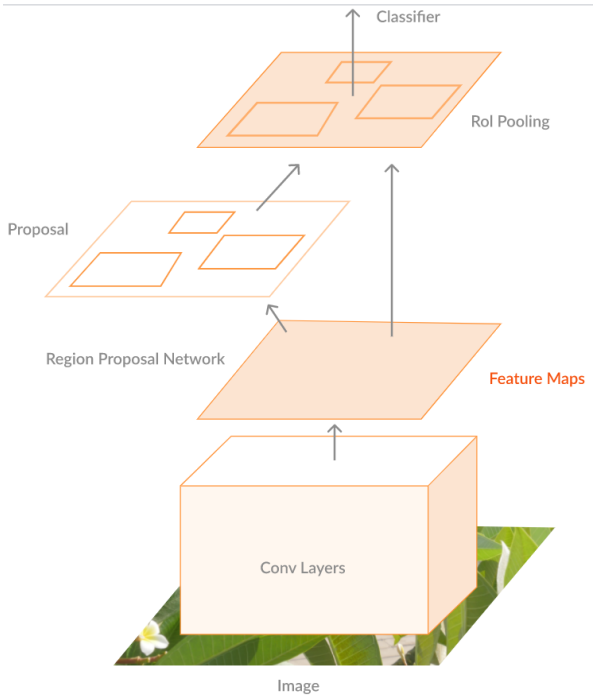


Figura 2.2: Arquitectura del modelo *Faster R-CNN*.

Red de detección de objetos

Utilizando los resultados obtenidos por la red de regiones propuestas en una capa de agrupación, que básicamente realiza una toma de regiones correspondientes con la propuesta generada por la red neuronal convolucional extractora de características, luego, divide la región en secciones (en un número fijo) de subventanas. Después se realiza un re-agrupamiento máximo en las subventanas para obtener un tamaño determinado para las salidas. Estas tienen un tamaño estándar de (n, 7, 7, 512), donde n es el número de propuestas generadas por la RPN, [54].

3. Metodología

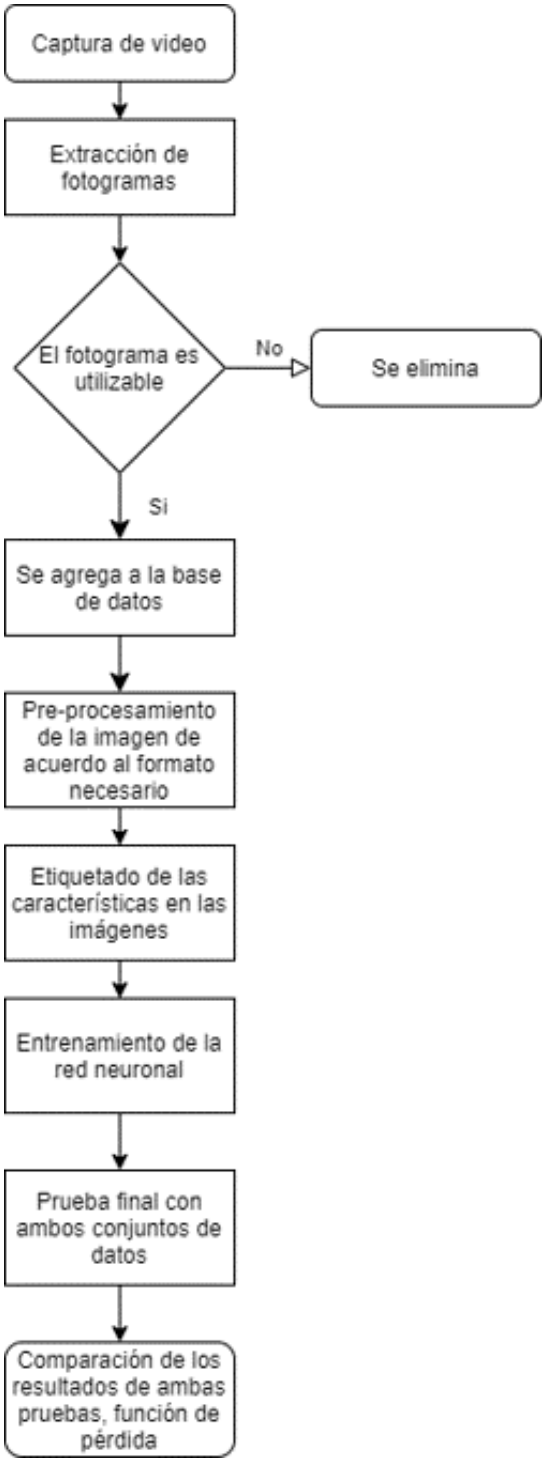


Figura 3.1: Diagrama de flujo de la metodología seguida.

3.1 Materiales

La conformación del conjunto de datos se hace con un equipo conformado por una cámara de vídeo Nakamichi ND37 con una resolución de 1920x1080 pixeles y una una tasa de captura de video de 30 cuadros por segundo, esta configuración fue utilizada debido a la limitación de la cámara Nakamichi.

-Para evitar comprometer la metodología del proyecto, se optó por utilizar únicamente este equipo para la captura de video, con la tasa de 30 *fps* (cuadros por segundo).

La arquitectura utilizada, con el objetivo de contar con la velocidad y presición necesaria para la aplicación, es la conocida como *Faster R-CNN-Inception-V2*. El modelo es entrenado con una GPU NVIDIA GEFORCE RTX 2070 utilizando TensorFlow.

Equipo	**CPU**	**GPU**	**RAM**
ASUS ROG Strix	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz	Nvidia Geforce RTX 2070	16 GB

Tabla 3.1: Características del equipo de computo utilizado.

3.2 Métodos

Para la realización de este proyecto se propuso un diagrama de flujo como el mostrado en la figura 3.1, el cual consiste en las actividades siguientes: captura de video, extracción de fotogramas, selección de los fotogramas de señales viales, pre-procesamiento de las imágenes capturadas, etiquetado de las imágenes, entrenamiento de la red neuronal convolucional para la detección y reconocimiento de las señales de tráfico con ambos conjuntos de datos y, finalmente, comparación de los valores de las funciones de pérdida entre las dos corridas de prueba.

3.2 Captura de video

Se hizo una revisión sobre las bases de datos de señales víales con acceso abierto, de las cuales se seleccionaron dos utilizadas normalmente como bases de datos de referencia:

- el conjunto GTSRB, por las siglas en inglés *German Traffic Sign Recognition Benchmark*
- el conjunto LISA de luces de tráfico.

debido a que no se encontró una base de datos con imágenes y contenido de México, se realizó la captura de video utilizando la cámara Nakamichi ND37. Se recorrieron algunas de las principales avenidas de la ciudad, entre las que se encuentran:

1. de las Torres

2. Hermanos Escobar
3. de los Insurgentes
4. de la Raza
5. Tecnológico
6. Santiago Troncoso

se complementó con información de otras calles ubicadas en zonas aledañas a las citadas avenidas.

La lista completa de calles y avenidas recorridas está conformada por: Ignacio de la Peña, Panamá, Ignacio Mejía, Hermanos Escobar, López Mateos, de los Insurgentes, José Borunda, Bolivia, Uruguay, Plutarco Elías Calles, de las Américas, Henri Dunant, del Charro, Tecnológico, Ejercito Nacional, Municipio Libre, Francisco Villarreal-de las Torres, Santiago Troncoso, Durango, de los Cedros, Intermex, Alamogordo, día del maestro, fénix, Montemayor, Oscar Flores Sánchez, Valentín Fuentes, del Acero, de la Vid, del Ciruelo, Teófilo Borunda, Paseo de la Victoria, Vicente Guerrero.

Los recorridos fueron realizados durante distintos horarios, estos incluyeron viajes de mañana, de tarde y de noche. Puede observarse con mayor detalle en la tabla 3.2 un desglose de los horarios habituales de captura de video para cada circuito completado.

Circuito	Recorrido	Horario
A	Colombia, Ignacio de la Peña, Bolivia, Hermanos Escobar, López Mateos, de los Insurgentes, Panamá, José Borunda, Colombia	7:00 - 9:00 13:00-17:00 20:00-22:00
B	Colombia, Ignacio de la Peña, Panamá, Hermanos Escobar, López Mateos, de los Insurgentes, Panamá, José Borunda, Colombia	7:00 - 9:00 13:00-17:00 20:00-22:00
C	Colombia, Ignacio de la Peña, Guillermo Solís, Vicente Guerrero, Plutarco Elías Calles, Hermanos Escobar, López Mateos, de los Insurgentes, Panamá, José Borunda, Colombia	7:00 - 9:00 13:00-17:00 20:00-22:00
D	Colombia, Ignacio de la Peña, Guillermo Solís, Vicente Guerrero, López Mateos, Ejercito Nacional, Villarreal-Torres, Santiago Troncoso, día del maestro, Alamogordo, Fenix, Intermex, de las Torres, Ejercito Nacional, Municipio Libre, Montemayor, de los Insurgentes, Panamá, José Borunda, Colombia	4:00 - 7:00 13:00-15:00
E	Colombia, Ignacio de la Peña, Guillermo Solís, Vicente Guerrero, López Mateos, Oscar Flores, del Acero, de la vid, del Ciruelo, del Acero, Oscar Flores, Gral Monterde, Municipio Libre, Montemayor, de los Insurgentes, Panamá, José Borunda, Colombia	13:00-18:00
F	Colombia, Ignacio de la Peña, Guillermo Solís, Vicente Guerrero, López Mateos, Oscar Flores, Teófilo Borunda, Paseo de la Victoria, de las Torres, Santiago Troncoso, día del maestro, Alamogordo, Fénix, Intermex, de las Torres, Ejercito Nacional, Municipio Libre, Montemayor, de los Insurgentes, Panamá, José Borunda, Colombia	5:00-7:00
G	Colombia, Ignacio de la Peña, Costa Rica, Vicente Guerrero, del Charro, Paseo Triunfo de la República, Gral Monterde, José Borunda, Colombia	15:00-17:00
H	Colombia, Ignacio de la Peña, Costa Rica, de los Insurgentes, de la Raza, Tecnológico, de la Raza, de los Insurgentes, Panamá, José Borunda, Colombia	11:00-20:00

Tabla 3.2: Circuitos recorridos, en la columna horario puede consultarse la hora a la que habitualmente se desarrolló la captura de video.

Las condiciones en las que se realizó la captura de video varió tanto por las condiciones de las vialidades, como por el horario en que se realizó y también las condiciones climatológicas del día. Algunas de estas condiciones pueden observarse en las imágenes de las figuras 3.2 y 3.3.

Imagen capturada durante un recorrido a las 4:00 horas.

Imagen capturada durante un recorrido realizado a las 8:30 horas.

Figura 3.2: Ambas imágenes, con deslumbramientos, presentan condiciones que dificultan la identificación de objetos.

Imagen capturada durante un recorrido a las 23:00 horas.

Imagen capturada durante un recorrido realizado a las 6:30 horas.

Figura 3.3: Ambas imágenes muestran condiciones de luminosidad, contraste y detalle completamente distintos.

3.2 Extracción de fotogramas

Se desarrolló un programa en Python que extrae cada fotograma del video. El código puede observarse en el anexo A. Para la manipulación de los videos se optó por la utilización de la librería OpenCV. Una vez ejecutado el código cada cuadro se iba guardando con su número consecutivo correspondiente, posteriormente se archivaba en una carpeta nombrada por el video de origen.

A cada uno de los videos obtenidos se le aplicó el extractor de fotogramas, con lo que se obtuvo una colección de imágenes consistentes en los recorridos realizados.

3.2 Selección de fotogramas

Con la colección de imágenes que se extrajeron de los distintos videos se procedió a realizar una selección de aquellos fotogramas que resultaran útiles para el proyecto. Es decir, aquellos que contuvieran alguna de las señales viales ubicadas en intersecciones. Para seleccionar estos fotogramas realizaron dos tareas principales.

1. Se utilizó la herramienta CVAT, *Computer Vision Annotation Tool*, que auxilió en la identificación de señales de alto e identificación de las luces de semáforo en algunos de los fotogramas.

2. Para complementar la labor del CVAT se realizó una inspección manual de cada fotograma.

En este último paso se realiza un etiquetado inicial de los fotogramas durante la revisión manual. Por otro lado, debido a la gran cantidad de datos de los conjuntos GTSRB y LISA, se desarrolló un código en Python, se puede observar en el anexo B, para extraer un subconjunto "aleatorio" de imágenes y de sus anotaciones correspondientes.

El objetivo de la creación de este subconjunto es evitar la sobrerrepresentación de alguna clase por sobre las demás y que el conjunto de datos extranjero tuviera mayor cantidad de datos que el conjunto de datos locales.

Una vez ejecutado el código se obtiene un archivo .csv con sólo las anotaciones del subconjunto aleatorio, por lo que es necesario eliminar las imágenes que no se corresponden con ninguna de esas etiquetas. Para ello se ejecuta el código de Python del anexo C. De esta manera se evita saturar el disco duro de la computadora con imágenes que no se utilizarán, lo que también contribuye con la optimización de espacio ocupado por la base de datos creada.

Video	fotogramas	Fotogramas útiles
MOV_0113	17875	6062
MOV_0114	7235	2410
MOV_0115	17948	5719
MOV_0116	3729	743
MOV_0117	8691	261
MOV_0118	4901	138
MOV_0119	17948	4400
MOV_0120	7532	1430

Tabla 3.3 continuación de la página anterior

Video	Fotogramas	Fotogramas Útiles
MOV_0121	17948	4979
MOV_0122	5017	101
MOV_0123	17949	2565
MOV_0124	410	0
MOV_0125	0	0
MOV_0126	17949	222
MOV_0127	2204	146
MOV_0128	17845	4715
MOV_0129	17948	1945
MOV_0130	16905	533
MOV_0090	1769	84
MOV_0092	1769	85
MOV_0093	1769	92
MOV_0094	1769	71
MOV_0168	1769	85
Total	208879	36786

Tabla 3.3: De la totalidad de fotogramas disponibles con la captura de video, 208 879, 36 786 contienen señales de alto o luces de semáforo.

3.2 Pre-procesamiento de las imágenes

Se realizó un escalado de la imagen cuyas dimensiones pasan de 1920x1080 a 416x416 pixeles en el caso de las capturadas con la cámara Nakamichi. En el caso de las imágenes del conjunto GTSRB (cuyas dimensiones varían de imagen en imagen) y del LISA cuyas imágenes tienen una dimensión de 1280x960, de igual manera en ambos casos se realizó un escalado a 416x416 pixeles.

Se optó por no realizar incrementos en el número de imágenes por medio de reflexiones horizontales o verticales, ninguna clase de rotación, de corte de imágenes ni aplicación de ruido o distorsión gaussiana. De manera general, no se aplicó ninguna técnica o método para aumentar la cantidad de fotogramas disponibles.

3.2 Etiquetado final de las imágenes

Durante este paso se eliminaron las etiquetas de falsos positivos obtenidas por la herramienta CVAT, se especificó que tipo de luz de semáforo correspondía con las imágenes

correctamente identificadas como tales y se verificó de manera manual cada imagen con sus etiquetas después del escalado.

Este etiquetado se realizó de manera manual, realizando una revisión visual de cada uno de los fotogramas incluidos en los conjuntos de datos. De acuerdo a la revisión realizada, se etiquetó o se cambió la etiqueta respecto a lo correspondiente en la imagen.

Conjunto	**Extranjero**	**Local**
Imagenes	2108	2409
anotaciones	3671	3784
Clases	6	6
alto	1222	1446
stop	864	765
go	701	663
stopLeft	338	601
warning	324	174
goLeft	222	175

Tabla 3.4: Conformación de los conjuntos de datos.

3.2 Entrenamiento de la red neuronal

De la totalidad de imágenes capturadas, después de la selección aleatoria de los fotogramas los conjuntos de datos quedaron conformados como puede observarse en la tabla 3.4.

Se utilizó el mismo conjunto de validación para realizar las pruebas, siendo este conjunto el conformado por imágenes locales, de esta manera los resultados de la función de pérdida generados durante la evaluación del algoritmo con cada base de datos permitirían observar el desempeño de los *datasets* local y extranjero.

Se utiliza un algoritmo pre-entrenado en la base de datos COCO (MSCOCO), con 100,000 iteraciones, de manera similar a lo propuesto en [1]. Dentro de los parámetros modificados en el algoritmo, cuya configuración se encuentra en el anexo D ,se encuentran, el número de clases fijo en 6, el redimensionador de imágenes seleccionando la opción “fixed_shape_resizer” que no deforma ni corta (sesga) las imágenes, con el extractor de características establecido en “faster_rcnn_inception_v2” con un tamaño para la primera etapa del extractor de características definido en 16. La arquitectura del Faster RCNN solo permite dos opciones en este parámetro un tamaño o *zancada* de 8 y de 16, optando por el de 16 para reducir la complejidad computacional del algoritmo [57].

Las escalas y razones de aspecto utilizadas para la generación del mallado de las anclas, [54], obedece la siguiente estructura con un polígono de 0.25, 0.5, 1 y 2, con razones de

aspecto de 0.5, 1 y 2; respectivamente. Además se consideran *zancadas* (la diferencia entre las anclas consecutivas en la imagen) de 16 pixeles tanto en forma vertical como horizontal.

La configuración de entrenamiento contiene el tamaño del lote "batch_size" definida en 12, debido a las limitaciones de la tarjeta gráfica y el tamaño de las imágenes. Mientras que el optimizador está definido como un optimizador de momento, una versión modificada del gradiente descendiente estocástico [58].

De acuerdo a lo mencionado en la subsección de preprocesamiento de imágenes no se aplicó ningún aumento a los datos de entrenamiento, por lo tanto la configuración de "data_aumentation_options" no contiene parámetros adicionales.

Con estos ajustes en el archivo de configuración se procedió a ejecutar el entrenamiento del algoritmo, el código puede observarse en el anexo E.

4. Resultados

Al finalizar ambos entrenamiento se verificaron las gráficas resultantes en Tensorboard.

4.1 Conjunto de datos locales

La primera gráfica en verificarse es la contenida en la figura 4.1 de la cual se extrae la siguiente información: se consideran cuatro casos, el primero consiste en el desempeño de la precisión promedio de las cajas de detección durante toda la ejecución del algoritmo. El segundo caso consiste en el comportamiento a partir de la iteración 25 mil, el tercer caso a partir de la 50 mil y el último caso a partir de la número 75 mil.

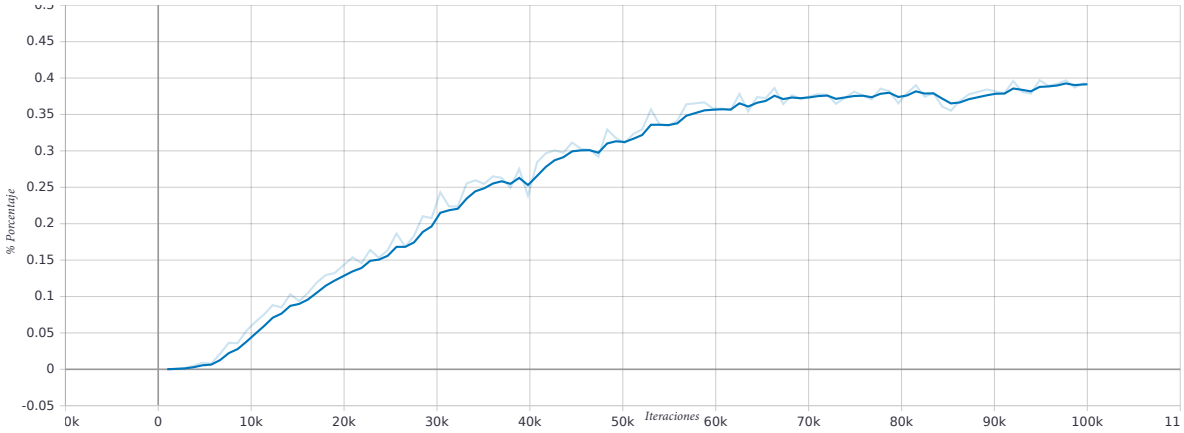


Figura 4.1: mAP, precisión de las cajas de detección.

Al considerar el primer caso se determinan las cotas como 0.39 % y 0 %. El segundo caso permite establecer cotas superior e inferior en los valores 0.39 % y 0.17 %, respectivamente. El tercer caso puede acotarse con los valores 0.39 % y 0.32 % y por último el cuarto caso tiene cota superior en 0.39 % e inferior en 0.37 %.

Con esta información se construye la tabla 4.1 que consiste en una propuesta de vecindades con un centro y un radio específico cuya área se extiende de acuerdo a las cotas determinadas en cada uno de los casos establecidos con anterioridad.

Datos locales	Vecindades de precisión	
mAP	**Centro %**	**Épsilon %**
Desde la primera iteración	0.195	0.195
Después de 25 mil iteraciones	0.280	0.110
Después de 50 mil iteraciones	0.355	0.035
Después de 75 mil iteraciones	0.380	0.010

Tabla 4.1: Vecindades de precisión de acuerdo a los resultados mostrados en la gráfica de promedio de precisión de las cajas de detección después de realizar cierto número de iteraciones del entrenamiento.

Con los resultados observables en la gráfica de la figura 4.2, el análisis de los valores de precisión de las cajas de detección permitió acotar las cuatro regiones de la forma siguiente: desde la primera iteración la cota superior es 0.77 % y la inferior 0 %; después de la iteración 25 mil la cota superior es 0.77 % y la inferior es 0.35 %; después de la iteración 50 mil las cotas son 0.77 % y 0.57 %; y después de las 75 mil iteraciones las cotas son 0.77 % y 0.59 %.

Conociendo estas cotas fue posible determinar vecindades de precisión de las cajas de detección para las imágenes de tamaño mediano. Estas vecindades pueden consultarse en la tabla 4.2, aquí se puede observar que el radio épsilon disminuye conforme avanzan las iteraciones durante el entrenamiento, desde 0.385 % hasta 0.09 %, además de centrarse en torno a 0.68 %.

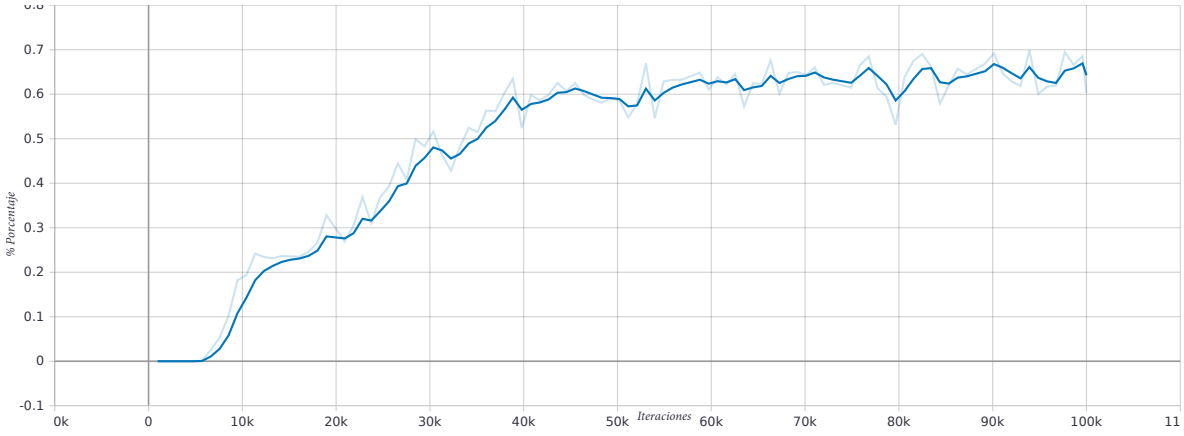


Figura 4.2: mAP, precisión de las cajas de detección. (Imágenes tamaño medio).

Datos locales	Vecindades de precisión	
mAP (imágenes medianas)	**Centro %**	**Épsilon %**
Desde la primera iteración	0.385	0.385
Después de 25 mil iteraciones	0.560	0.210
Después de 50 mil iteraciones	0.670	0.100
Después de 75 mil iteraciones	0.680	0.090

Tabla 4.2: Vecindades de valores mAP (imágenes de tamaño medio), resultantes durante el entrenamiento.

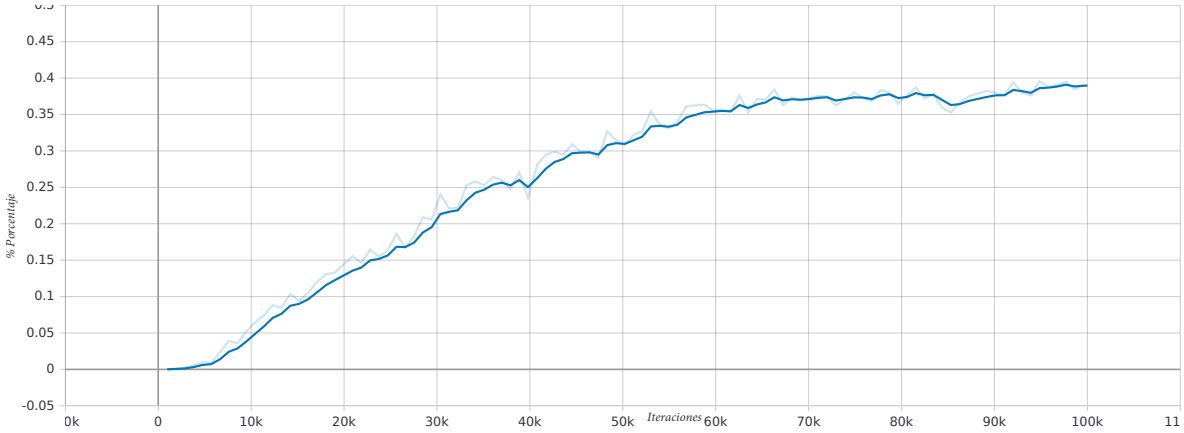


Figura 4.3: mAP, precisión de las cajas de detección. (Imágenes tamaño chico).

En el caso de la gráfica de valores de precisión de las cajas de detección para las imágenes chicas, contenida en la figura 4.3, los valores que permiten acotar el mAP en cada uno de los casos analizados, considerando que el valor de 0.39 % es el máximo de la función y se alcanza durante el final de la evaluación esta cota superior es válida para los cuatro casos. El primer par de cotas, superior e inferior es, 0.39 % y 0 %; después de 25 mil iteraciones el par es 0.39 % y 0.17 %; después de las 50 mil 0.39 % y 0.31 %; y durante las últimas 25 mil iteraciones 0.39 % y 0.36 %.

Con las cotas establecidas se puede construir la tabla 4.3, las cuales permiten observar la disminución de los radios épsilon al avanzar el entrenamiento y el desplazamiento del centro de la región de precisión de las cajas de detección hasta culminar en 0.375 %.

Datos locales	Vecindades de precisión	
mAP (imágenes chicas)	**Centro %**	**Épsilon %**
Desde la primera iteración	0.195	0.195
Después de 25 mil iteraciones	0.280	0.110
Después de 50 mil iteraciones	0.350	0.040
Después de 75 mil iteraciones	0.375	0.015

Tabla 4.3: Vecindades de valores mAP (imágenes de tamaño chico), resultantes durante el entrenamiento.

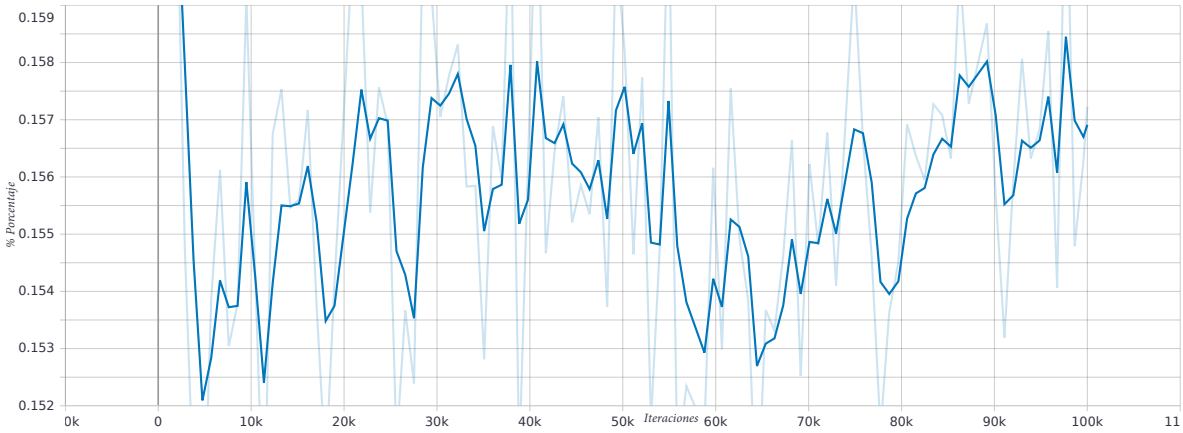


Figura 4.4: Función de pérdida del algoritmo durante la validación.

Para la función de pérdida generada durante la validación se tuvo un comportamiento no monótono con intervalos de crecimiento alternados con intervalos de decrecimiento. Realizando el mismo análisis que el realizado a los valores del promedio de precisión de cajas de detección se determinan las cotas superiores e inferiores (respectivamente) siguientes: desde la primera iteración se tiene 0.1589 % y 0.1527 %; en el segundo caso, después de las 25 mil iteraciones, 0.1589 % y 0.1527 %; a partir de las 50 mil iteraciones las cotas son 0.1589 % y 0.1523 %; y después de 75 mil iteraciones las cotas quedan definidas en 0.1589 % y 0.1539 %. Con estos datos se conformó la tabla 4.4.

Datos locales	Vecindades de precisión	
Función de pérdida en la evaluación	**Centro %**	**Épsilon %**
Desde la primera iteración	0.15555	0.00335
Después de 25 mil iteraciones	0.15580	0.00310
Después de 50 mil iteraciones	0.15580	0.00310
Después de 75 mil iteraciones	0.15640	0.00250

Tabla 4.4: Regiones acotadas para los valores de la función de pérdida durante la evaluación después de determinado número de iteraciones.

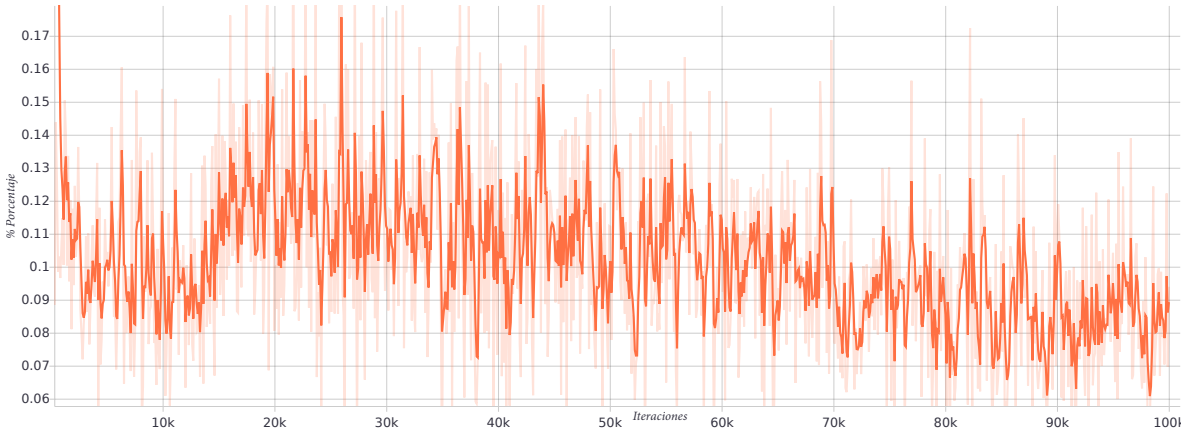


Figura 4.5: Función de pérdida del algoritmo durante el entrenamiento.

Al considerar la función de perdida del algoritmo al realizarse la evaluación del algoritmo se consideran las cotas superiores de 0.175 %, 0.175 %, 0.137 % y 0.128 %, con una cota inferior en común a todas las regiones 0.061 %.

Con estos valores se construyó la tabla 4.5, lo cual permite confirmar lo observado visualmente, la disminución del porcentaje de pérdida hacia un valor centrado en 0.945 % con una tolerancia de $\pm$ 0.0335 %.

Datos locales	Vecindades de precisión	
Función de pérdida en la evaluación	**Centro %**	**Épsilon %**
Desde la primera iteración	0.1180	0.0570
Después de 25 mil iteraciones	0.1180	0.0570
Después de 50 mil iteraciones	0.0990	0.0380
Después de 75 mil iteraciones	0.0945	0.0335

Tabla 4.5: Regiones acotadas para los valores de la función de pérdida durante la evaluación después de determinado número de iteraciones.

4.2 Conjunto de datos extranjeros

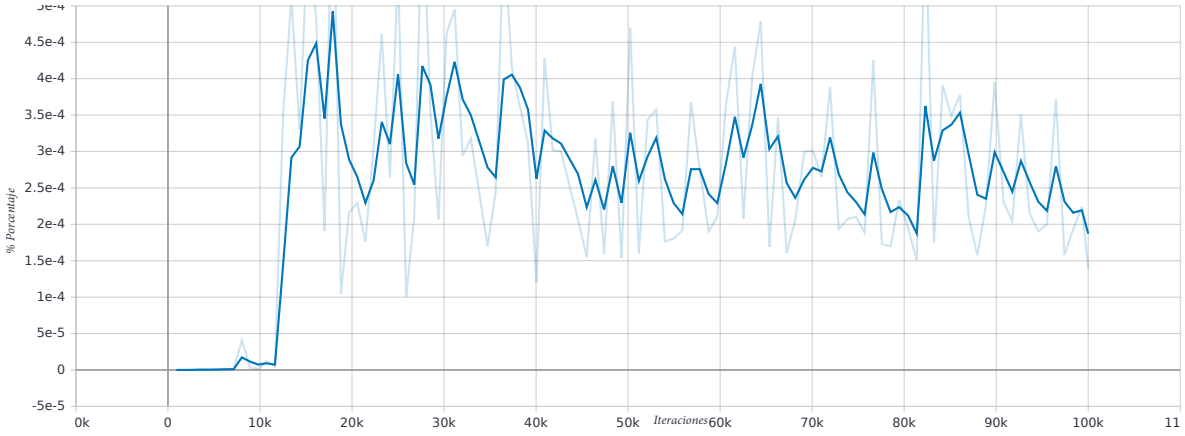


Figura 4.6: mAP, precisión de las cajas de detección.

La gráfica contenida en la figura 4.6 permite observar el acotamiento de los resultados después de la iteración 12 mil por encima de los valores de 0.00017 %. Del mismo modo, puede observarse un acotamiento en los valores por debajo de la cota de 0.00049 %. A partir de esta información se puede establecer una región o vecindad de precisión de las cajas de detección centrada en 0.00033 % con una tolerancia o un radio (épsilon) de ±0.00016 % a partir de la iteración 12 mil.

Haciendo extensivo el análisis a la ejecución posterior del algoritmo, después de las 25 mil iteraciones se acotan los valores de la precisión promedio de las cajas de detección superior e inferiormente por las cotas 0.00044 % y de 0.00017 %, superior e inferior respectivamente. A partir de la iteración 50 mil, los valores quedan dentro de la vecindad 0.00028 % ± 0.00011 % y después de las 75 mil iteraciones acotado superiormente por 0.00036 % e inferiormente por 0.00017 %. Lo cuál conforma una vecindad centrada en 0.00027 % con un radio de ± 0.0001 %. Se puede observar la información de estas vecindades en la tabla 4.6.

Datos extranjeros	**Vecindades de precisión**	
mAP	**Centro %**	**Épsilon %**
Desde la primera iteración	0.00033	0.00016
Después de 25 mil iteraciones	0.00031	0.00014
Después de 50 mil iteraciones	0.00028	0.00011
Después de 75 mil iteraciones	0.00027	0.00010

Tabla 4.6: Vecindades de precisión de acuerdo a los resultados mostrados en la gráfica de promedio de precisión de las cajas de detección después de realizar cierto número de iteraciones del entrenamiento.

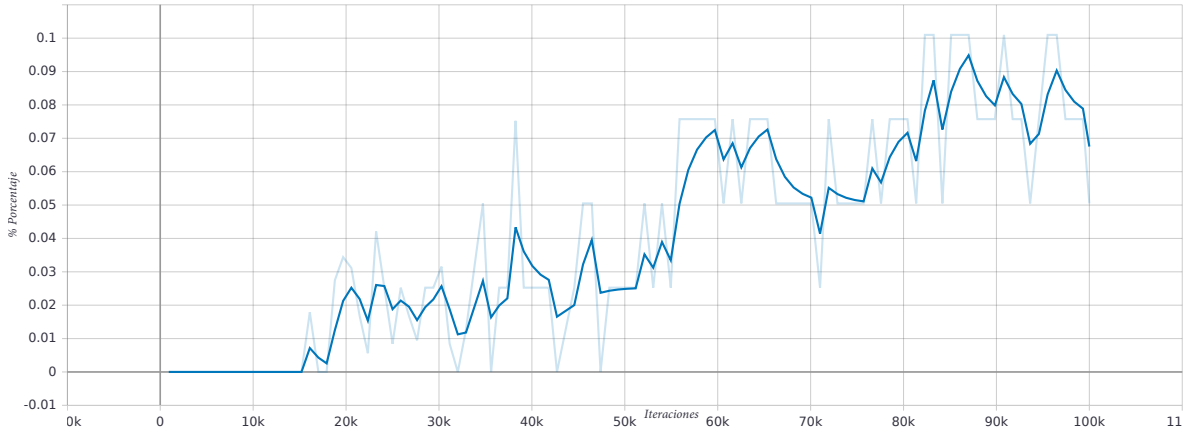


Figura 4.7: mAP, precisión de las cajas de detección. (Imágenes tamaño medio).

En el caso de la gráfica de la figura 4.7 que solamente considera las imágenes o fotogramas de tamaño mediano en el caso del conjunto conformado por datos del extranjero se observa un comportamiento creciente en la gráfica, si bien el mismo no es monótono en todo momento el mejor desempeño en la precisión promedio de las cajas de detección es alcanzado hacia el final del entrenamiento. Se observa que a partir del inicio del entrenamiento los valores de la función mAP quedan acotados superior e inferiormente por los valores 0.094 %y 0.003 %. A partir de las 25 mil iteraciones las cotas observables son 0.094 % y 0.011 %; a partir de 50 mil iteraciones 0.094 % y 0.0026 % son las cotas que se pueden identificar al ejecutar el resto del entrenamiento y durante las últimas 25 mil iteraciones la cota superior se establece en 0.094 % y 0.058 %. Con estos datos se puede elaborar la tabla 4.7 que establece valores donde formar conjuntos abiertos, vecindades o regiones de precisión de las cajas de detección.

Datos extranjeros	**Vecindades de precisión**	
mAP (imágenes medianas)	**Centro %**	**Épsilon %**
Desde la primera iteración	0.0485	0.0455
Después de 25 mil iteraciones	0.0525	0.0415
Después de 50 mil iteraciones	0.0600	0.0340
Después de 75 mil iteraciones	0.0760	0.0180

Tabla 4.7: Vecindades de valores mAP (imágenes de tamaño medio), resultantes durante el entrenamiento.

En el caso del mAP para imágenes de tamaño chico, se obtuvo como resultado la gráfica contenida en la figura 4.8. En la tabla 4.8 se observa que los valores de precisión de las cajas de detección en el caso de las imágenes chicas presenta su mejor desempeño desde las

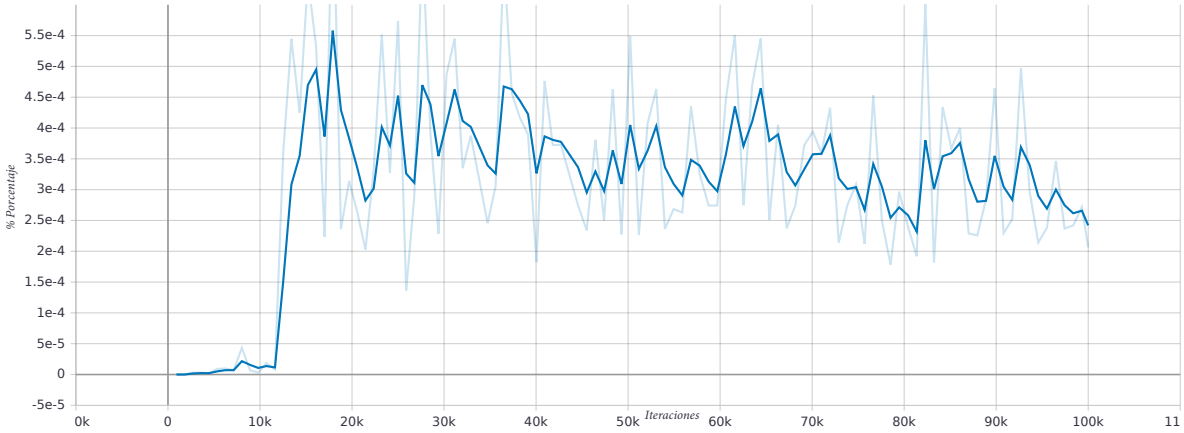


Figura 4.8: mAP, precisión de las cajas de detección. (Imágenes tamaño chico).

15 hasta las 18 mil iteraciones y el peor desempeño después de alcanzar este valor *supremo* ocurre después de las 80 mil iteraciones. Considerando esta particularidad, la cota inferior de *mAP* de 0.00023 % es constante en cada periodo analizado. Considerando el *mAP* desde el inicio del entrenamiento se observa la cota superior 0.00056 %, en el análisis posterior a las 25 mil iteraciones la cota superior se establece en 0.00047 %, después de las 50 mil iteraciones queda en 0.00046 % y durante las últimas 25 mil es 0.000375 %. Se observa que la gráfica tiende a centrarse, con el transcurso de las iteraciones, en la región de 0.00030 %, disminuyendo también la variación entre los valores obtenidos como precisión promedio de las cajas de detección con cada iteración.

Datos extranjeros	**Vecindades de precisión**	
mAP (imágenes chicas)	**Centro %**	**Épsilon %**
Desde la primera iteración	0.00040	0.00017
Después de 25 mil iteraciones	0.00035	0.00012
Después de 50 mil iteraciones	0.00035	0.00012
Después de 75 mil iteraciones	0.00030	0.00007

Tabla 4.8: Vecindades de valores mAP (imágenes de tamaño chico), resultantes durante el entrenamiento.

De la función de pérdida del algoritmo durante la validación también se realiza un análisis similar.

Con una cota superior para los valores de la función de pérdida de 0.408 % en cada uno de los casos analizados completa el análisis la determinación de cotas inferiores de 0.342 % para el entrenamiento a partir de la primera iteración; de 0.352 % después de 25 mil iteraciones; de 0.387 % durante la segunda mitad del entrenamiento y durante la ejecución de las últimas

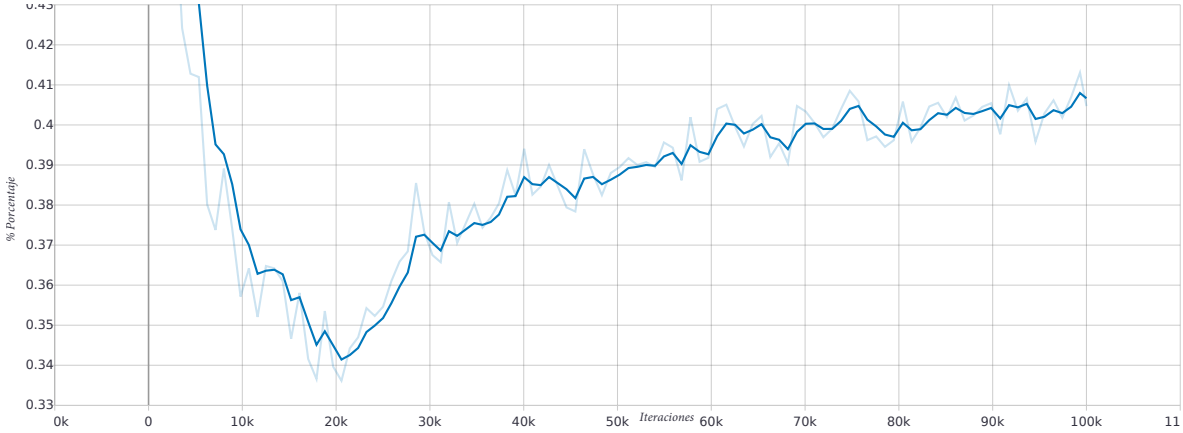


Figura 4.9: Función de pérdida del algoritmo durante la validación.

25 mil iteraciones en 0.397 %.

El comportamiento creciente, después de una caída pronunciada durante las primeras 21 mil iteraciones muestra una tendencia hacia el valor de 0.405.

Datos extranjeros	**Vecindades de precisión**	
Función de pérdida en la validación	**Centro %**	**Épsilon %**
Desde la primera iteración	0.3750	0.0330
Después de 25 mil iteraciones	0.3800	0.0280
Después de 50 mil iteraciones	0.3975	0.0105
Después de 75 mil iteraciones	0.4025	0.0055

Tabla 4.9: Regiones acotadas para los valores de la función de pérdida durante la validación después de determinado número de iteraciones.

A partir de los valores generados por la función de pérdida durante el entrenamiento, mostrado en la gráfica de la figura 4.10 , se determinan valores de cotas superiores e inferiores para los siguientes escenarios: después de la primera iteración son 0.15 % y 0.056 %; después de 25 mil iteraciones son 0.15 % y 0.056 %; posterior a las 50 mil iteraciones 0.142 % y 0.056 % y desde las 75 hasta las 100 mil iteraciones 0.124 % y 0.058 %. Con estos resultados es posible determinar la información de la tabla 4.10.

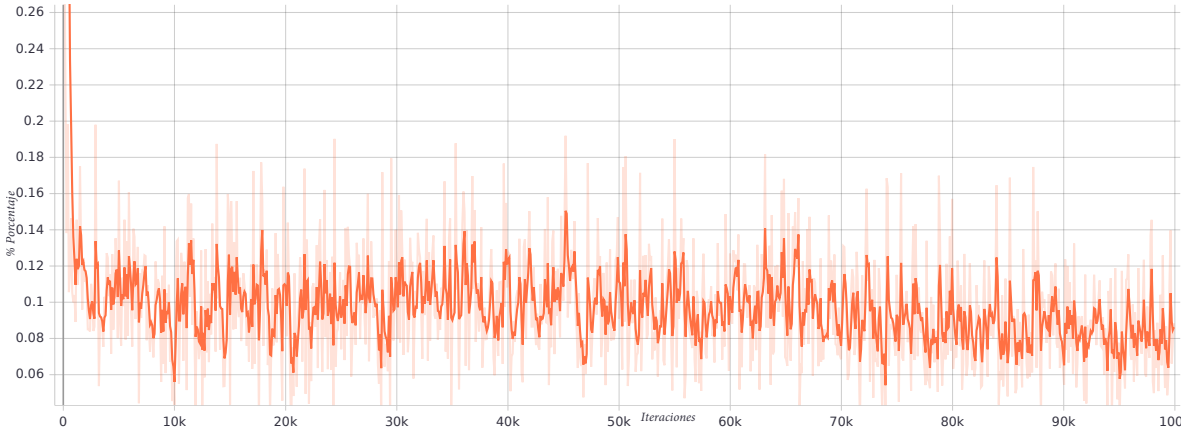


Figura 4.10: Función de pérdida del algoritmo durante el entrenamiento.

Datos extranjeros	**Vecindades de precisión**	
Función de pérdida en la evaluación	**Centro %**	**Épsilon %**
Desde la primera iteración	0.103	0.047
Después de 25 mil iteraciones	0.103	0.047
Después de 50 mil iteraciones	0.099	0.043
Después de 75 mil iteraciones	0.091	0.033

Tabla 4.10: Regiones acotadas para los valores de la función de pérdida durante la evaluación después de determinado número de iteraciones.

4.3 Imágenes evaluadas por el algoritmo, datos extranjeros y datos locales

Tensorboard permitió observar algunas de las imágenes durante el proceso de entrenamiento del algoritmo. Esto tanto para imágenes del conjunto de validación y el de entrenamiento.

Para el caso de los datos extranjeros se tienen las siguientes imágenes en las figuras: 4.11, 4.12, 4.13, 4.14 y 4.15. Mientras que para el conjunto de datos locales se obtuvieron las figuras: 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22 , 4.23, 4.24 y, por último, 4.25.

En el caso de la imagen de la figura 4.11 el algoritmo de detección de objetos ya al finalizar el entrenamiento muestra una caja de detección con una calificación de detección del 100 %. No existen objetos a detectar adicionales.

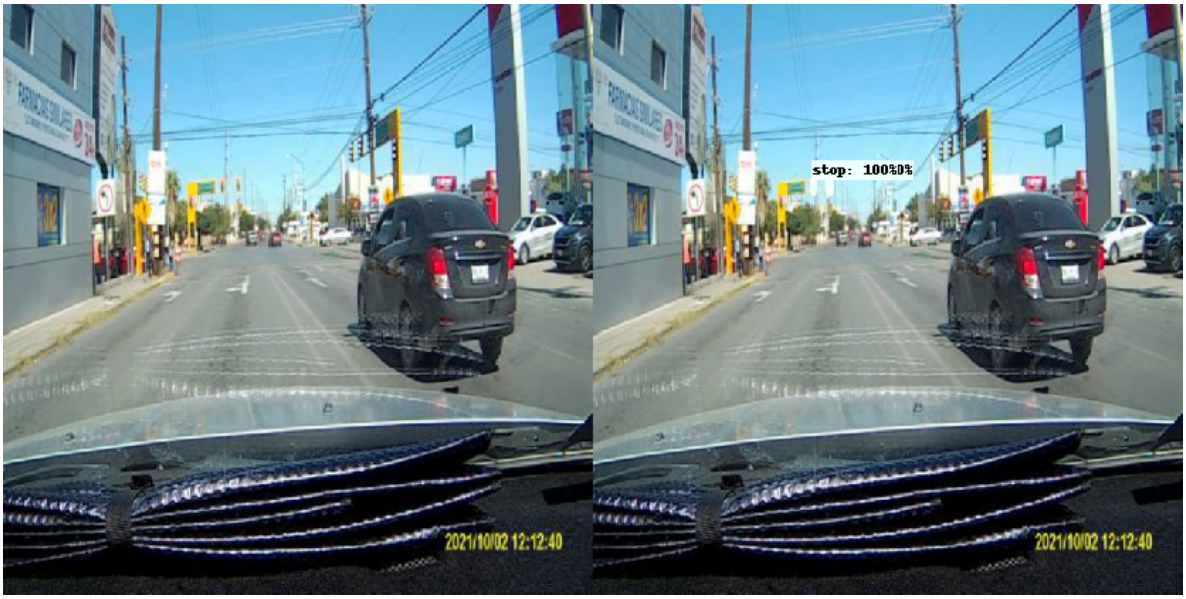


Figura 4.11: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenidas Paseo Triunfo de la República y Plutarco Elías Calles.

Figura 4.12: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenidas de la Raza y Adolfo López Mateos.

Al observar la imagen de la figura 4.12 se puede verificar que el algoritmo de detección de objetos, al finalizar el entrenamiento, muestra una caja de detección con una calificación de detección del 100 % identificando correctamente un semáforo de vuelta a la izquierda. Sin embargo, no detecta la luz de semáforo verde que se encuentra a la derecha.

Figura 4.13: Comparación de imagen antes y después de ser evaluada, señal de alto ubicada en calles Montemayor y Plan de Guadalupe.

Las figuras 4.13 , cuyas condiciones lumínicas no son las ideales, muestra la correcta identificación de una señal de alto.

Figura 4.14: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenidas de la Raza y Adolfo de la Huerta.

La figura 4.14 muestra un caso distinto al segundo, aunque identifica dos luces de semáforo, en lugar de una sola, no logra identificar correctamente la luz de semáforo de la izquierda que consiste en una vuelta a la izquierda.

Figura 4.15: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenida Ejercito Nacional.

En el caso de la imágen de la figura 4.15 se utilizan dos niveles distintos del entrenamiento, al llevar 10 mil iteraciones y al finalizar. La primera imagen muestra un falso positivo, identificando una luz como un semáforo verde, mientras que la segunda imagen deja de cometer este error de falso positivo e identifica las tres luces de semáforo amarillo que se encuentran en el fotograma. Todas las imágenes anteriores fueron identificadas por el algoritmo entrenado con la base de datos extranjera.

Figura 4.16: Comparación de imagen antes y después de ser evaluada, alto ubicado en calles Montemayor y 21 de Marzo.

Para el caso del entrenamiento realizado con la base de datos local, a partir de la figura 4.16, se tiene una correcta identificación de una señal de alto.

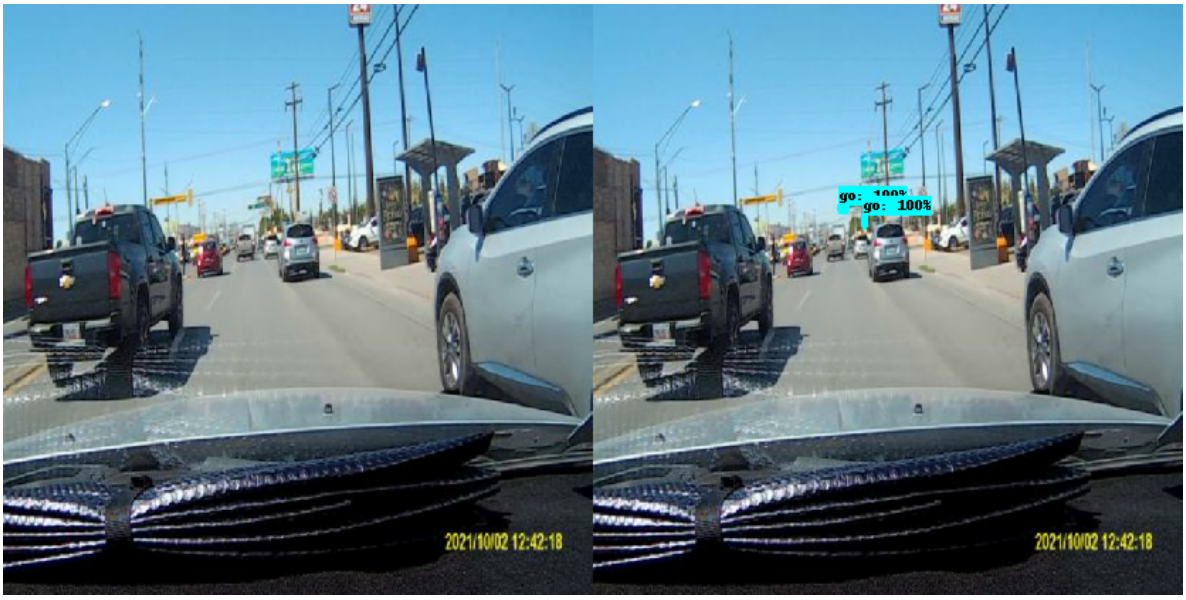


Figura 4.17: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenidas de la Raza y Plutarco Elías Calles.

En el caso de los fotogramas de la figura 4.17 la correcta identificación de dos luces de semáforo verdes. Correspondientes a todos los objetos detectables e identificables en la imagen.

Figura 4.18: Comparación de imagen antes y después de ser evaluada, señal de alto ubicada en calles Municipio Libre y General Monterde.

En el caso de los fotogramas de la figura 4.18 la correcta identificación de dos luces de semáforo verdes. Correspondientes a todos los objetos detectables e identificables en la imagen.

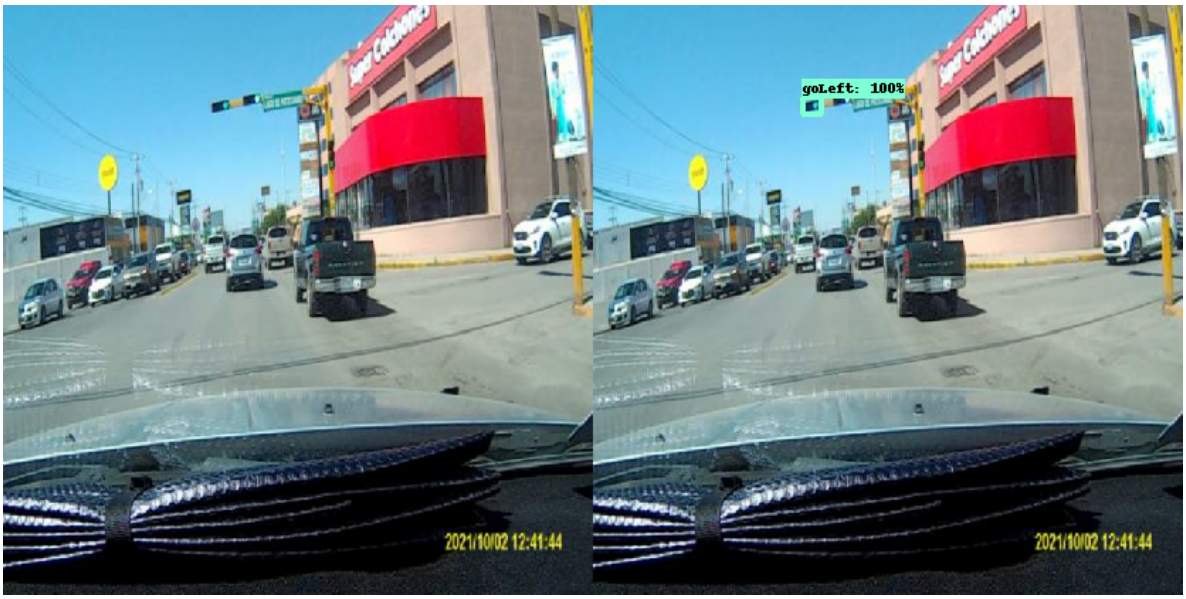


Figura 4.19: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenidas de la Raza y Lago de Pátzcuaro.

Para los fotogramas contenidos en las figuras 4.19, 4.21 la detección de objetos no es la correcta, pues no logra localizar una luz de semáforo en cada una de ellas.

Figura 4.20: Comparación de imagen antes y después de ser evaluada, alto ubicado en calle Panamá.

Las imágenes de las figuras 4.20, 4.23 y 4.24 identifican correctamente todos los objetos contenidos en ellas, tres señales de alto, tres luces de semáforo amarillo y dos luces de semáforo verde.

Figura 4.21: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenida Ejercito Nacional.

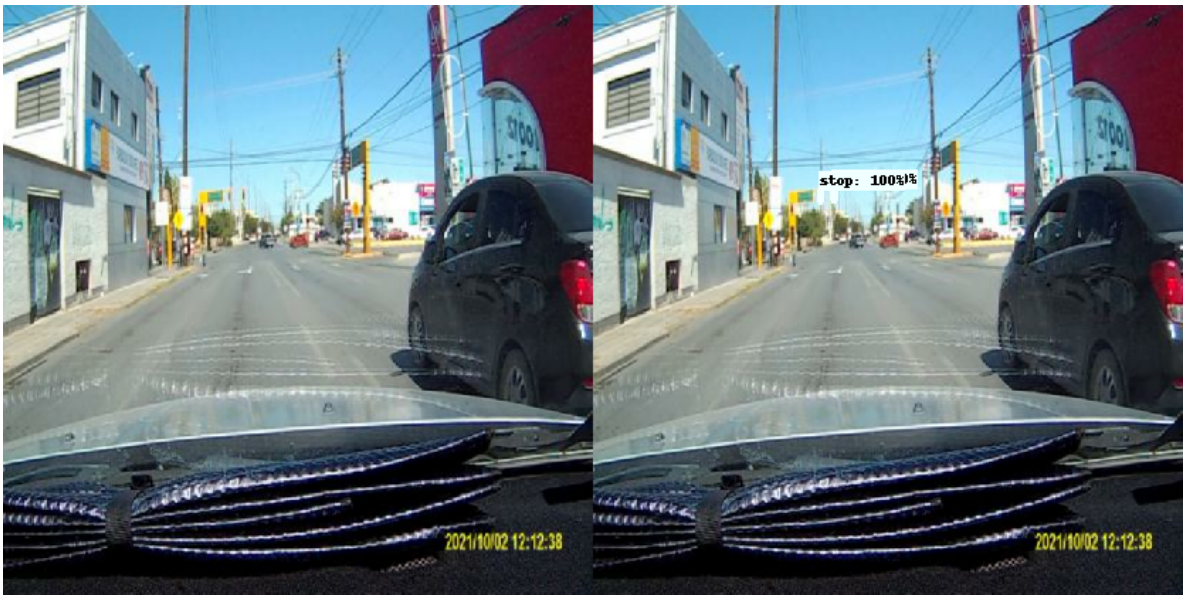


Figura 4.22: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenida Plutarco Elías Calles.

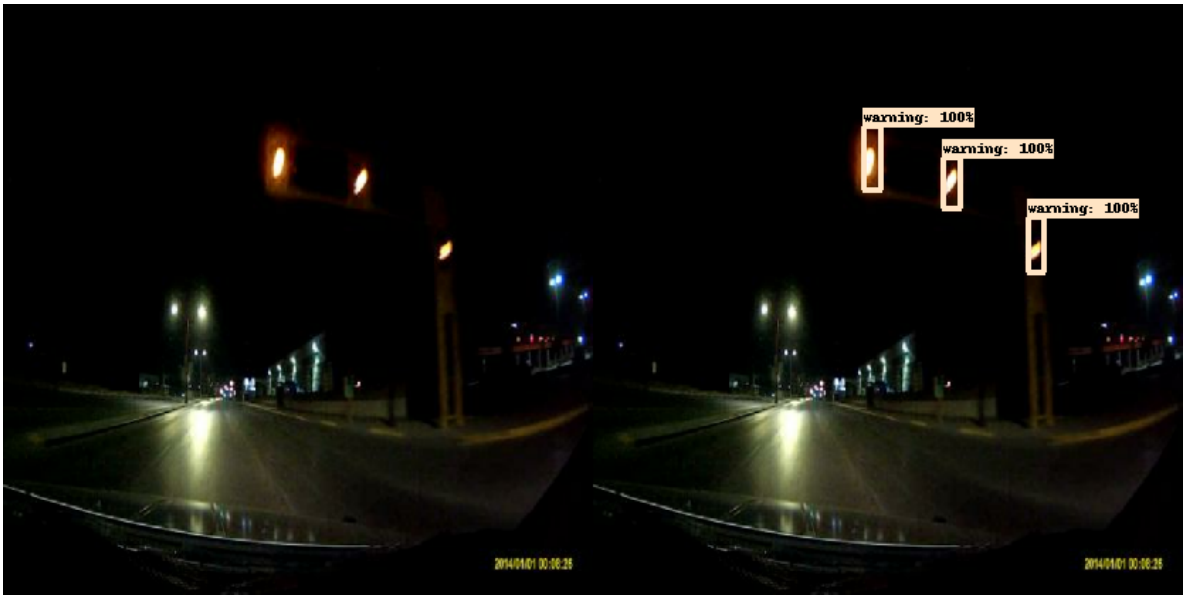


Figura 4.23: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenida Ejercito Nacional.

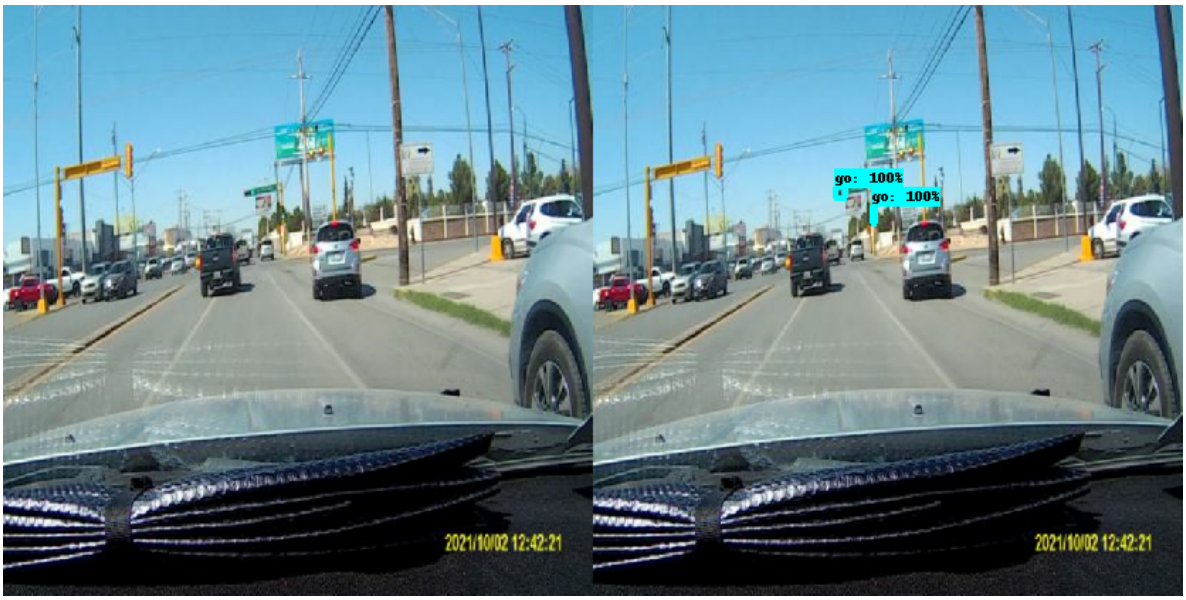


Figura 4.24: Comparación de imagen antes y después de ser evaluada, semáforo ubicado en avenida de la Raza.

Figura 4.25: Comparación de imagen antes y después de ser evaluada, alto ubicado en calle Montemayor.

Por último tanto las imágenes de la figura 4.22 como de la figura 4.25 detectan correctamente (e identifican) la luz de semáforo roja y la señal de alto.

5. Conclusiones

Por medio de los resultados obtenidos, de cada una de las funciones tanto de las de la media de la precisión de las cajas de detección como de las funciones de pérdida tanto durante el entrenamiento como la evaluación y consultando las tablas de las regiones o vecindades de los valores resultantes se observa que en cada comparativa, salvo por la de la función de pérdida durante el entrenamiento el algoritmo entrenado con datos locales muestra un mejor desempeño.

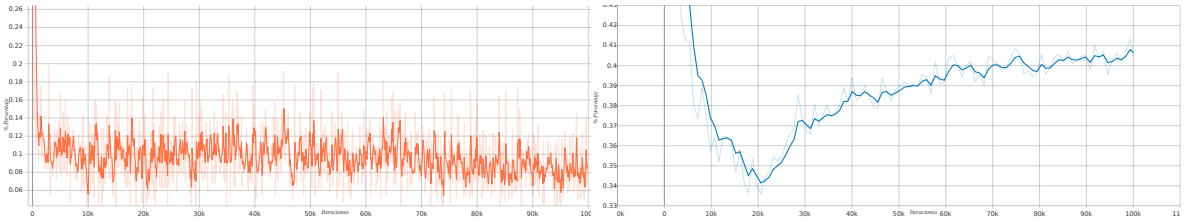
En el caso restante, el desempeño de los algoritmos muestra valores del mismo orden. Ambas quedan acotadas, aproximadamente dentro de los valores de 0.17 % y 0.06 % en el caso del entrenamiento con datos locales y de 0.15 % y 0.058 % para el entrenamiento con datos extranjeros, lo cual puede observarse en las figuras 4.10 y 4.5 y en las tablas 4.10 y 4.5.

Mientras que para la función de pérdida durante la validación, se mostraron comportamientos diferentes. Por un lado, el algoritmo al ser entrenado con el conjunto de datos extranjero mostró un comportamiento con una menor variación proporcional y, además, presentando convergencia hacia el valor porcentual de pérdida de, aproximadamente, 0.41 %, como se puede observar en la gráfica de la figura 4.9 y en la tabla 4.9.

Al ser entrenado con el conjunto de datos locales presenta un orden de los valores de pérdida menores, en comparación con la otra base de datos, con un valor estimado de aproximadamente 0.15 % de pérdida.

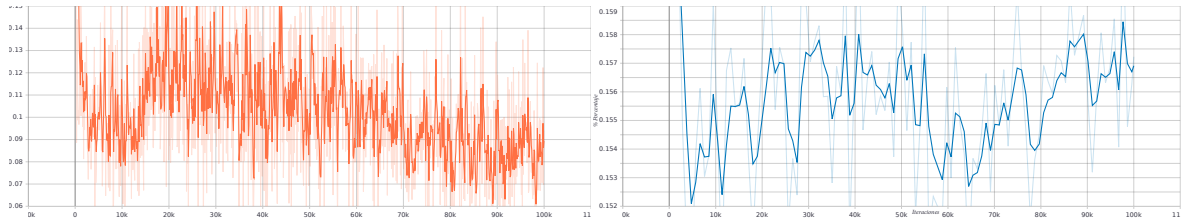
Tomando en cuenta, tanto los valores de la función de pérdida durante el entrenamiento como de los generados durante la validación del algoritmo de detección de objetos, y comparando estos resultados con los obtenidos por [1], cuya gráfica de pérdida durante la validación se muestra en la figura 5.3, se puede concluir que el desempeño obtenido durante las pruebas realizadas no alcanza el nivel de los resultados presentados por el equipo de la India.

Sin embargo, una reducción a un tercio, aproximadamente, de la función de pérdida entre el cambio de conjunto de datos es, considerando que se trata de un tema de seguridad, suficiente para replantear la necesidad de evaluar más a fondo la utilización de bases de datos locales para el entrenamiento. De aquí que se proponga evaluar con un distinto ajuste de los híper parámetros del algoritmo ambos conjuntos de entrenamiento, aumentando el número de iteraciones a realizar e incluso utilizando técnicas de clasificación de objetos más recientes como Yolo v5 o Mask RCNN.



Función de pérdida durante el entrenamiento. **Función de pérdida durante la validación.**

Figura 5.1: Funciones de pérdida resultantes de la evaluación del algoritmo con el conjunto de datos extranjeros.



Función de pérdida durante el entrenamiento. **Función de pérdida durante la validación.**

Figura 5.2: Funciones de pérdida resultantes de la evaluación del algoritmo con el conjunto de datos locales.

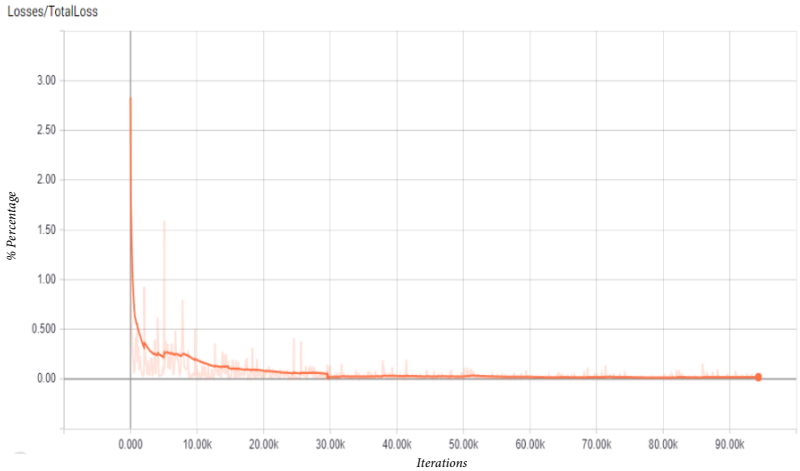


Fig. 3. Loss graph of model

Figura 5.3: Resultados de la función de pérdida total obtenida en [1].

Bibliografía

[1] R. Kulkarni, S. Dhavalikar, and S. Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning," in *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*. Institute of Electrical and Electronics Engineers Inc., 2018.

[2] C. Vallon, Z. Ercan, A. Carvalho, and F. Borrelli, "A machine learning approach for personalized autonomous lane change initiation and control," in *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv. IEEE, 2017, pp. 1590–1595.

[3] A. Uçar, Y. Demir, and C. Güzeliş, "Object recognition and detection with deep learning for autonomous driving applications," *SIMULATION: Transactions ofthe Society for Modeling*, vol. 93, no. 9, pp. 759–769, sep 2017. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0037549717709932

[4] J. Cao, Y. Pang, and X. Li, "Pedestrian Detection Inspired by Appearance Constancy and Shape Symmetry," *IEEE Transactions on Image Processing*, vol. 25, no. 12, pp. 5538–5551, oct 2016.

[5] M. Mobahi and S. H. Sadati, "An improved deep learning solution for object detection in self-driving cars," in *2020 28th Iranian Conference on Electrical Engineering, ICEE 2020*, 2020, pp. 1–5.

[6] Y. Kang, H. Yin, and C. Berger, "Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 171–185, 2019.

[7] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, mar 2013.

[8] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3-4, pp. 197–387, 2013.

[9] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, dec 2015. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0007-7

[10] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, N. Tache, Ed. O'Reilly, 2017. [Online]. Available: http://oreilly.com/safari

[11] Y. Pang, M. Sun, X. Jiang, and X. Li, "Convolution in convolution for network in network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1587–1597, may 2018.

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

[13] INEGI, "Accidentes de tránsito," 2021. [Online]. Available: https://www.inegi.org.mx/temas/accidentes/

[14] F. Lindner, U. Kressel, and S. Kaelberer, "Robust recognition of traffic signals," *IEEE Intelligent Vehicles Symposium, Proceedings*, no. July 2004, pp. 49–53, 2004.

[15] G. V. Angeles Perez, J. Castillejos Lopez, A. L. R. Cabello, E. B. Grajales, A. P. Espinosa, and J. L. Q. Fabian, "Road Traffic Accidents Analysis in Mexico City through Crowdsourcing Data and Data Mining Techniques," *International Journal of Computer and Information Engineering*, vol. 12, no. 8, 2018.

[16] M. Bertoncello and D. Wee, "Ten ways autonomous driving could redefine the automotive world," *McKinsey and Company*, vol. 6, pp. 1–6, 2015.

[17] L. Yue, M. Abdel-Aty, Y. Wu, and L. Wang, "Assessment of the safety benefits of vehicles' advanced driver assistance, connectivity and low level automation systems," *Accident Analysis & Prevention*, vol. 117, pp. 55–64, 2018.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. Cambridge, MA: MIT Press, 2016. [Online]. Available: http://files.sig2d.org/sig2d14.pdf{#}page=5

[19] D. J. Livingstone, *Artificial neural networks: methods and applications*. Springer, 2008.

[20] J. Zhang, H. Yuan, and H. Dong, *Introduction to deep learning*. Springer, 2020.

[21] I. E. Naqa and M. J. Murphy, "What is Machine Learning?" in *Machine Learning in Radiation Oncology: Theory and Applications*, I. E. Naqa, M. J. Murphy, and R. Li, Eds. Springer International Publishing, 2015, pp. 3–11.

[22] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[23] M. Gopal, *Applied Machine Learning*. McGraw-Hill Education, 2018. [Online]. Available: https://books.google.com.mx/books?id=BsKlDwAAQBAJ

[24] S. Raschka and V. Mirjalili, *Python Machine Learning*. Packt, 2019, no. December 2019.

[25] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning and Data Mining*, 2nd ed., C. Sammut and G. I. Webb, Eds. New York: Springer, 2018.

[26] K. Murphy, *Machine learning: A probabilistic Perspective*. The MIT Press, 2012, vol. 5, no. 2.

[27] C. Molnar, *Interpretable Machine Learning*, 2019.

[28] R. Lopez and J. Fernandez, *Las redes neuronales artificiales*, 1st ed. Netbiblo, 2008.

[29] J. Torres, *Deep Leaning: Introducción práctica con Keras*. Kindle direct, 2019, vol. 53, no. 9. [Online]. Available: www.journal.uta45jakarta.ac.id

[30] Z. H. Zhou, "A brief introduction to weakly supervised learning," *National Science Review*, vol. 5, no. 1, pp. 44–53, 2018.

[31] K. Hsu, S. Levine, and C. Finn, "Unsupervised learning via meta-learning," *arXiv*, 2018.

[32] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[33] T. O. Ayodele, "Types of Machine Learning Algorithms," *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[34] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electron Markets*, pp. 4910–4914, 2021.

[35] G. Ciaburro and B. Venkateswaran, *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles*. Packt Publishing Ltd, 2017.

[36] F. Chollet, *Deep learning with Python*. Manning New York, 2018, vol. 361.

[37] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[38] A. Nandy and M. Biswas, "Neural Network Basics," in *Neural Networks in Unity*. Springer, 2018, pp. 1–26.

[39] A. Bhardwaj, W. Di, and J. Wei, *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing Ltd, 2018.

[40] J. Patterson and A. Gibson, *Deep learning: A practitioner's approach*. .ºReilly Media, Inc.", 2017.

[41] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53 040–53 065, 2019.

[42] S. Skansi, *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018.

[43] Z. Zhang, S. Zohren, and S. Roberts, "Deeplob: Deep convolutional neural networks for limit order books," *IEEE Transactions on Signal Processing*, vol. 67, no. 11, pp. 3001–3012, 2019.

[44] E. Garcia Sanchez, "Introducción a las redes neuronales de convolución. Aplicación a la visión por ordenador," p. 50, 2019.

[45] V. Podlozhnyuk, "Image convolution with CUDA," *NVIDIA Corporation white paper*, vol. 2097, no. 3, p. 21, 2007.

[46] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "CNN-RNN: A Unified Framework for Multi-label Image Classification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, 2016, pp. 2285–2294.

[47] T. Guo, J. Dong, H. Li, and Y. Gao, "Simple convolutional neural network on image classification," in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. IEEE, 2017, pp. 721–724.

[48] R. Devkar and S. Shiravale, "A Survey on Multi-label Classification for Images," *Article in International Journal of Computer Applications*, vol. 162, no. 8, pp. 975–8887, 2017. [Online]. Available: https://www.researchgate.net/publication/315302952

[49] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, apr 2016.

[50] A. Shustanov and P. Yakimov, "CNN Design for Real-Time Traffic Sign Recognition," *Procedia Engineering*, vol. 201, pp. 718–725, 2017. [Online]. Available: https://doi.org/10.1016/j.proeng.2017.09.594

[51] J. G. Wang and L. B. Zhou, "Traffic Light Recognition with High Dynamic Range Imaging and Deep Learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 4, pp. 1341–1352, 2019.

[52] G. Ozturk, R. Koker, O. Eldogan, and D. Karayel, "Recognition of Vehicles, Pedestrians and Traffic Signs Using Convolutional Neural Networks," in *4th International Symposium on Multidisciplinary Studies and Innovative Technologies, ISMSIT 2020 - Proceedings*, 2020, p. 8.

[53] D. A. Alghmgham, G. Latif, J. Alghazo, and L. Alzubaidi, "Autonomous Traffic Sign (ATSR) Detection and Recognition using Deep CNN," in *Procedia Computer Science*, vol. 163. Elsevier B.V., 2019, pp. 266–274.

[54] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/results

[55] A. Salvador, X. Giro-I-Nieto, F. Marques, and S. Satoh, "Faster R-CNN Features for Instance Search," pp. 394–401, 2016.

[56] R. Gavrilescu, C. Zet, C. Fosalau, M. Skoczylas, and D. Cotovanu, "Faster R-CNN:an Approach to Real-Time Object Detection," *EPE 2018 - Proceedings of the 2018 10th International Conference and Expositions on Electrical And Power Engineering*, pp. 165–168, dec 2018.

[57] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, 2015, pp. 5353–5360.

[58] R. R. Tobias, L. Carlo De Jesus, M. E. Mital, S. Lauguico, M. Guillermo, R. R. Vicerra, A. Bandala, and E. Dadios, "Faster R-CNN Model with Momentum Optimizer for RBC and WBC Variants Classification," in *2020 IEEE 2nd Global Conference on Life Sciences and Technologies*. Institute of Electrical and Electronics Engineers Inc., mar 2020, pp. 235–239.

A. Extractor de fotogramas

```
import cv2
vidcap = cv2.VideoCapture('NOMBRE_VIDEO.avi')
success,image = vidcap.read()
cuadros = int(vidcap.get(cv2.CAP_PROP_FRAME_COUNT))
contador = 0
bandera = 0
while contador < cuadros-1:
success,image = vidcap.read()
# imagen = cv2.imread("frame"+str(contador)+".png")
# guardar fotograma como png
cv2.imwrite("frame%d.png" % contador, image)
# salir al presionar escape
if cv2.waitKey(10) == 27:
break
contador += 1
completado = round(contador/cuadros,2)
if bandera != completado:
bandera = completado
print(completado*100, "%")
```

B. Seleccionador de fotogramas

```
from random import randint
import os, pandas
df = pandas.read_csv('LN5.csv')
clip = "nightClip5--"
fotos = len(next(os.walk('./N5'))[2])
c1, c2, c3, c4, c5 = 0,0,0,0,0
ban = []
k = 75
df1 = pandas.DataFrame({'filename': [], 'width':[], 'height':[],
'class':[], 'xmin':[], 'ymin':[], 'xmax':[], 'ymax':[]})
while (c1 & c2 & c4 & c5) <k:
sid = randint(0,fotos)
digitos = len(str(sid))
fid = str(sid).zfill(5)
picture=clip+fid+'.jpg'
print(picture)
if (df.loc[(df['filename'] == picture) & (df['class'] == 'warning')]).empty
!= False and c3<k:
if picture not in ban:
ban.append(picture)
items = df.index[df['filename']==picture].tolist()
for i in items:
df.iloc[i]
df1 = df1.append(df.iloc[i], ignore_index=True)
c3 +=1
elif (df.loc[(df['filename'] == picture) & (df['class'] == 'stopLeft')]).empty !=
False and c4<k:
if picture not in ban:
ban.append(picture)
items = df.index[df['filename']==picture].tolist()
for i in items:
df.iloc[i]
df1 = df1.append(df.iloc[i], ignore_index=True)
c4 +=1
elif (df.loc[(df['filename'] == picture) & (df['class'] == 'goLeft')]).empty !=
False and c5<k:
if picture not in ban:
ban.append(picture)
items = df.index[df['filename']==picture].tolist()
for i in items:
```

```
df.iloc[i]
df1 = df1.append(df.iloc[i], ignore_index=True)
c5 +=1
elif (df.loc[(df['filename'] == picture) & (df['class'] == 'go')]).empty
!= False and c1<k:
if picture not in ban:
ban.append(picture)
items = df.index[df['filename']==picture].tolist()
for i in items:
df.iloc[i]
df1 = df1.append(df.iloc[i], ignore_index=True)
c1 +=1
elif (df.loc[(df['filename'] == picture) & (df['class'] == 'stop')]).empty
!= False and c2<k:
if picture not in ban:
ban.append(picture)
items = df.index[df['filename']==picture].tolist()
for i in items:
df.iloc[i]
df1 = df1.append(df.iloc[i], ignore_index=True)
c2 +=1

df1.drop_duplicates()

df1.to_csv('TD5.csv', index=False)
```

C. Eliminador de imágenes

```
import os, pandas
df2 = pandas.read_csv('LD13.csv')
clip = "dayClip13--"
fotos = len(next(os.walk('./D13'))[2])
for i in range(fotos):
digitos = len(str(i))
fid = str(i).zfill(5)
picture=clip+fid+'.jpg'
if picture not in df2['filename'].values and
os.path.exists('./D13/'+picture)==True:
print('No existe '+picture)
os.remove('./D13/'+picture)
print('Ha sido eliminada '+ picture)
```

D. Configuración del algoritmo Faster RCNN

```
model {
faster_rcnn {
num_classes: 6
image_resizer {
keep_aspect_ratio_resizer {
min_dimension: 600
max_dimension: 1024
}
}
feature_extractor {
type: 'faster_rcnn_inception_v2'
first_stage_features_stride: 16
}
first_stage_anchor_generator {
grid_anchor_generator {
scales: [0.25, 0.5, 1.0, 2.0]
aspect_ratios: [0.5, 1.0, 2.0]
height_stride: 16
width_stride: 16
}
}
first_stage_box_predictor_conv_hyperparams {
op: CONV
regularizer {
l2_regularizer {
weight: 0.0
}
}
initializer {
truncated_normal_initializer {
stddev: 0.01
}
}
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
```

```
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
mask_rcnn_box_predictor {
use_dropout: false
dropout_keep_probability: 1.0
fc_hyperparams {
op: FC
regularizer {
l2_regularizer {
weight: 0.0
}
}
initializer {
variance_scaling_initializer {
factor: 1.0
uniform: true
mode: FAN_AVG
}
}
}
}
}
second_stage_post_processing {
batch_non_max_suppression {
score_threshold: 0.0
iou_threshold: 0.6
max_detections_per_class: 100
max_total_detections: 300
}
score_converter: SOFTMAX
}
second_stage_localization_loss_weight: 2.0
second_stage_classification_loss_weight: 1.0
}
}

train_config: {
batch_size: 12
optimizer {
momentum_optimizer: {
learning_rate: {
manual_step_learning_rate {
initial_learning_rate: 0.0002
```

```
schedule {
step: 900000
learning_rate: .00002
}
schedule {
step: 1200000
learning_rate: .000002
}
}
}
momentum_optimizer_value: 0.9
}
use_moving_average: false
}
gradient_clipping_by_norm: 10.0
fine_tune_checkpoint: "/content/drive/MyDrive/E/models/research/pretrained_model/model.ckpt"
from_detection_checkpoint: true
# Note: The below line limits the training process to 200K steps, which we
# empirically found to be sufficient enough to train the COCO dataset. This
# effectively bypasses the learning rate schedule (the learning rate will
# never decay). Remove the below line to train indefinitely.
num_steps: 100000
data_augmentation_options {
random_horizontal_flip {
}
}
}

train_input_reader: {
tf_record_input_reader {
input_path: "/content/drive/MyDrive/E
/tensorflow-object-detection-faster-rcnn
/data/train/Foreign.tfrecord"
}
label_map_path: "/content/drive/MyDrive/E
/tensorflow-object-detection-faster-rcnn
/data/train/Foreign_label_map.pbtxt"
}

eval_config: {
num_examples: 8000
# Note: The below line limits the evaluation process to 10 evaluations.
# Remove the below line to evaluate indefinitely.
max_evals: 10
```

```
}

eval_input_reader: {
tf_record_input_reader {
input_path: "/content/drive/MyDrive/E
/tensorflow-object-detection-faster-rcnn
/data/valid/Foreign.tfrecord"
}
label_map_path: "/content/drive/MyDrive/E/
tensorflow-object-detection-faster-rcnn
/data/train/Foreign_label_map.pbtxt"
shuffle: false
num_readers: 1
}
```

E. Algoritmo ejecutado en el cuaderno Jupyter

```
!pip install tensorflow_gpu==1.15
repo_url = 'shorturl.at/iqGQ3'
num_steps = 100000
num_eval_steps = 50

MODELS_CONFIG = {
'ssd_mobilenet_v2': {
'model_name': 'ssd_mobilenet_v2_coco_2018_03_29',
'pipeline_file': 'ssd_mobilenet_v2_coco.config',
'batch_size': 12
},
'faster_rcnn_inception_v2': {
'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
'pipeline_file': 'faster_rcnn_inception_v2_coco.config',
'batch_size': 12
},
'rfcn_resnet101': {
'model_name': 'rfcn_resnet101_coco_2018_01_28',
'pipeline_file': 'rfcn_resnet101_pets.config',
'batch_size': 8
}
}

‘MODELS_CONFIG‘.
selected_model = 'faster_rcnn_inception_v2'

MODEL = MODELS_CONFIG[selected_model]['model_name']

pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']

batch_size = MODELS_CONFIG[selected_model]['batch_size']

import os

%cd /content/drive/MyDrive/L

repo_dir_path = os.path.abspath(os.path.join('.', os.path.basename(repo_url)))

!git clone {repo_url}
%cd {repo_dir_path}
```

```
!git pull
%cd /content/drive/MyDrive/L
!git clone --quiet https://github.com/tensorflow/models.git

!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -q Cython contextlib2 pillow lxml matplotlib pycocotools tf_slim

%cd /content/drive/MyDrive/L/models/research
!protoc object_detection/protos/*.proto --python_out=.

import os
os.environ['PYTHONPATH'] += ':/content/drive/MyDrive/L/models/research/
:/content/drive/MyDrive/L/models/research/slim/'

!python object_detection/builders/model_builder_test.py

%cd /content/drive/MyDrive/L/tensorflow-object-detection-faster-rcnn/data
!curl -L "https://app.roboflow.com/ds/oJ4V5dQbqX?key=G3tbZbVGS5"
> roboflow.zip; unzip roboflow.zip; rm roboflow.zip
test_record_fname = '/content/drive/MyDrive/L/
tensorflow-object-detection-faster-rcnn
/data/valid/local.tfrecord'
train_record_fname = '/content/drive/MyDrive/L/
tensorflow-object-detection-faster-rcnn
/data/train/local.tfrecord'
label_map_pbtxt_fname = '/content/drive/MyDrive/L/
tensorflow-object-detection-faster-rcnn
/data/train/local_label_map.pbtxt'

%cd /content/drive/MyDrive/L/models/research

import os
import shutil
import glob
import urllib.request
import tarfile
MODEL_FILE = MODEL + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
DEST_DIR = '/content/drive/MyDrive/E/models/research/pretrained_model'

if not (os.path.exists(MODEL_FILE)):
urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
```

```
tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

os.remove(MODEL_FILE)
if (os.path.exists(DEST_DIR)):
shutil.rmtree(DEST_DIR)
os.rename(MODEL, DEST_DIR)

!echo {DEST_DIR}
!ls -alh {DEST_DIR}

fine_tune_checkpoint = os.path.join(DEST_DIR, "model.ckpt")
fine_tune_checkpoint

import os
pipeline_fname = os.path.join('/content/drive/MyDrive/L/models/
research/object_detection/samples/configs/', pipeline_file)

assert os.path.isfile(pipeline_fname), '`{}` not exist'.format(pipeline_fname)

def get_num_classes(pbtxt_fname):
from object_detection.utils import label_map_util
label_map = label_map_util.load_labelmap(pbtxt_fname)
categories = label_map_util.convert_label_map_to_categories(
label_map, max_num_classes=90, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
return len(category_index.keys())

import re

num_classes = get_num_classes(label_map_pbtxt_fname)
with open(pipeline_fname) as f:
s = f.read()
with open(pipeline_fname, 'w') as f:

# fine_tune_checkpoint
s = re.sub('fine_tune_checkpoint: ".*?"',
'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

# tfrecord files train and test.
s = re.sub(
'(input_path: ".*?)(train.record)(.*?")',
'input_path: "{}"'.format(train_record_fname), s)
```

```
s = re.sub(
'(input_path: ".*?)(val.record)(.*?")',
'input_path: "{}"'.format(test_record_fname), s)

# label_map_path
s = re.sub(
'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

# Set training batch_size.
s = re.sub('batch_size: [0-9]+',
'batch_size: {}'.format(batch_size), s)

# Set training steps, num_steps
s = re.sub('num_steps: [0-9]+',
'num_steps: {}'.format(num_steps), s)

# Set number of classes num_classes.
s = re.sub('num_classes: [0-9]+',
'num_classes: {}'.format(num_classes), s)
f.write(s)

!cat {pipeline_fname}
model_dir = 'training/'
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip
LOG_DIR = model_dir
get_ipython().system_raw(
'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
.format(LOG_DIR)
)

get_ipython().system_raw('./ngrok http 6006 &')

! curl -s http://localhost:4040/api/tunnels | python3 -c \
"import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"

!pip install lvis

!python /content/drive/MyDrive/L/models/research/object_detection/model_main.py \
--pipeline_config_path={pipeline_fname} \
--model_dir={model_dir} \
--alsologtostderr \
--num_train_steps={num_steps} \
--num_eval_steps={num_eval_steps}
```