Clasificación y Conteo de Tráfico en Video Mediante un Paradigma de Inteligencia Computacional

Universidad Distrital Francisco José de Caldas



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

William Eduardo Sierra Lozano
Sebastián Leonardo Bejarano Medellín
Director: Ing. MSc Cesar Andrey Perdomo Charry
Modalidad: Monografía

6 de julio de 2021

A mis padres Leonardo y Claudia principalmente por haberme permitido la oportunidad de estudiar la carrera a pesar de los contratiempos al principio, a mi hermana Gabriela por apoyarme durante todo el camino y hacer de la vida algo más feliz y llevadero, a mi novia Leidy Stefania por compartir conmigo todo este proceso de ejecución del proyecto y estar conmigo en la universidad incluso mientras me quejaba, que esto permita cumplir nuestros sueños juntos. A mis amigos por distraernos cuando fue necesario durante nuestra carrera y en otros ámbitos, al profe Cesar por haber soportado toda nuestra molestadera a lo largo del proyecto y al grupo LASER por permitirnos las herramientas para llevar a cabo esto que esperamos que sea una semilla para nuevos proyectos.

Sebastián

A Dios por abrir las puertas para estudiar en una grandiosa universidad y levantarme en los momentos más duros que pude atravesar en el camino, a ms padres William Y Bibiana por creer en mi y formarme como una persona responsable, respetuosa y perseverante, a mi hermano Daniel por inspirarme a ser mejor persona, a mi novia Luisa por apoyarme y enseñarme que es posible soñar en grande y creer que se puede llegar más lejos si se propone, A mi grupo de amigos por compartir momentos de calidad dentro y fuera de la universidad y motivarme a ser mejor estudiante, al grupo LASER y el profesor Cesar por darnos la oportunidad de desarrollar este proyecto perteneciendo al grupo de investigación y por el acompañamiento que nos dieron, a mis profesores por inculcar en mi valores y conocimientos valiosos para mi vida profesional y personal.

William

Índice general

Página

Resumen **9**

Introducción **10**

Objetivos **12**

1. Pruebas de diferentes cámaras y plataformas **13**
1.1. Sistema embebido . . . 13
1.2. Revisión de plataformas . . . 13
1.2.1. Raspberry Pi . . . 13
1.2.2. NVIDIA Jetson Nano . . . 14
1.2.3. Google Coral Dev Board . . . 15
1.2.4. Caracterización de los dispositivos . . . 16
1.3. Cámara digital . . . 17
1.4. Evaluación de cámaras . . . 17

2. Generación de la base de datos de entrenamiento de los modelos **20**
2.1. Bases de datos . . . 20
2.2. Construcción de base de datos . . . 20

3. Entrenamiento y validación de los diferentes modelos **24**
3.1. El problema de inteligencia computacional . . . 24
3.2. Ingeniería de características . . . 25
3.3. Extracción de características y entrenamiento . . . 25
3.4. Extracción de características tipo SIFT . . . 25
3.5. La neurona artificial . . . 28
3.6. El perceptrón multicapa . . . 29
3.7. El aprendizaje como problema de optimización . . . 30
3.8. Descenso de gradiente estocástico . . . 31
3.9. Entropía binaria cruzada . . . 31
3.10. Entropía categórica cruzada . . . 32
3.11. Implementación de clasificadores . . . 32
3.12. Clasificador de perceptron multicapa . . . 32
3.12.1. Clasificador de No Vehículo/Vehículo . . . 32
3.12.2. Clasificador de tipo de vehículo . . . 35
3.13. Imagen digital . . . 37

3.14. Filtro digital bidimensional . . . 38
3.15. Red neuronal convolucional . . . 38
3.16. Clasificador con red neuronal convolucional . . . 39
3.16.1. Clasificador de Vehículo/No Vehículo . . . 40
3.16.2. Clasificador para tipo de vehículo . . . 41
3.17. Algoritmo genético simple . . . 43
3.17.1. Genoma . . . 43
3.17.2. Métodos de selección de padres . . . 44
3.17.3. Métodos de crossover . . . 44
3.18. Clasificador evolucionado por medio de algoritmo genético simple . . . 44
3.19. Transfer Learning . . . 47
3.19.1. MobileNet V2 . . . 49
3.19.2. MobileNetV2 Vehículo - No vehículo . . . 50
3.19.3. MobileNetV2 Tipo de vehículo . . . 51

4. Creación del algoritmo y base de datos de actualización en tiempo real para los vehículos contados 53
4.1. Procesamiento de la imagen para obtener regiones de interés . . . 53
4.2. Desarrollo de la interfaz de usuario en Python . . . 55
4.2.1. Obtención de región de interés ROI . . . 56
4.2.2. Clasificar ROI como vehículo o no vehículo . . . 57
4.2.3. Agregar Tracker . . . 59
4.2.4. Clasificar el tipo de vehículo . . . 59
4.2.5. Estimación del color del vehículo . . . 60
4.2.6. Estimación de la la velocidad . . . 60
4.2.7. Enviar a la base de datos . . . 61
4.3. Pruebas en vídeo . . . 62

5. Interfaz para el dispositivo 65
5.1. NodeJS . . . 65
5.2. jQuery . . . 66
5.3. Flask . . . 66
5.4. Docker . . . 66
5.5. Pantalla de inicio . . . 67
5.5.1. Validación de código y autorización de acceso . . . 67
5.5.2. Cookie de sesión . . . 68
5.6. Sección principal . . . 68
5.6.1. Comunicación entre la app principal en Flask y la plataforma en NodeJS . . . 69
5.7. Pantalla de gráficas . . . 71
5.7.1. Gráficos con los datos de la base de datos en tiempo real . . . 71
5.8. Resumen de la versión 1.0 . . . 73
5.9. Modificaciones para la Coral Dev Board . . . 73
5.9.1. Quantización después del entrenamiento . . . 74
5.9.2. Compilador EdgeTPU . . . 75
5.9.3. Carcasa para el dispositivo final . . . 76

6. Conclusiones **78**
6.1. Trabajos Futuros 79

7. Anexos **81**

Índice de figuras

1.1. Raspberry Pi 4B - Tomada de [RaspberryPi4, 2021] 14
1.2. Raspberry Pi Zero W - Tomada de [RaspberryPiZeroW, 2021] 14
1.3. Tarjeta de desarrollo NVIDIA Jetson Nano - Tomada [JetsonNano, 2021] 15
1.4. Coral Dev Board - Tomada de [GoogleCoral, 2021] 15
1.5. Cámara de raspberry versión 1.3, tomada de [Raspberry1.3, 2021] . . 18
1.6. Cámara de raspberry versión 2.1, tomada de [Raspberry2.1, 2021] . . 18
1.7. Cámara arducam, tomada de [Arducam, 2021] 18
1.8. Cámara Coral Dev Board, tomada de [GoogleCoral, 2021] 18
1.9. Imagen de prueba de la cámara de Raspberry Pi Rev 1.3. 19
1.10. Imagen de prueba de la cámara de Raspberry Pi Rev 2.1. 19
1.11. Imagen de prueba de la cámara Arducam. 19
1.12. Imagen de prueba de la cámara Google Coral Camera. 19

2.1. Ejemplo de imágenes de la base de datos de Brasil [Laroca et al., 2021]. 22
2.2. Ejemplo de imágenes de la base de datos de Kaggle [Huttunen, 2019]. 22
2.3. Ejemplo de imágenes de la base de datos de universidad de Madrid [Arróspide et al., 2012]. 22
2.4. Ejemplo de imágenes de la base de datos generada durante el proyecto. 22

3.1. Aprendizaje supervisado 24
3.2. Localización y dibujo de puntos clave mediante el algoritmo SIFT [Lowe, 2004] 26
3.3. Interpretación de bolsa de palabras visuales, tomada de [Davida, 2018]. 27
3.4. Algoritmo propuesto para obtener el dataset a partir del Bag of Visual Words 28
3.5. Neurona y Neurona artificial [Vrahatis et al., 2000] 28
3.6. Ejemplo de red neuronal totalmente conectada [Jorge, 2001]. 29
3.7. Métrica de error en función de dos parámetros [Bishop, 2006b] 30
3.8. Disminución en la varianza de los centroides utilizados para BoVW a medida que varían los clusters. 33
3.9. Función de coste de la arquitectura ganadora. 35
3.10. Exactitud de la arquitectura ganadora. 35
3.11. Varianza según el número de clusters para la clasificación de tipo de vehículo. 35
3.12. Función de coste de la arquitectura ganadora. 37
3.13. Exactitud de la arquitectura ganadora. 37
3.14. Proceso de digitalización [Ferrero and Mora, 2008]. 38

3.15. Red neuronal convolucional para tarea de clasificación, tomada de [Vertebrar, 2021] . . . 39
3.16. Función de coste de la arquitectura ganadora. . . . 41
3.17. Exactitud de la arquitectura ganadora. . . . 41
3.18. Función de coste de la arquitectura ganadora. . . . 43
3.19. Exactitud de la arquitectura ganadora. . . . 43
3.20. Ciclo de vida del algoritmo genético de pyGAD . . . 46
3.21. Evolución para la red convolucional de vehículo y no vehículo. . . . 47
3.22. Evolución para la red convolucional de tipo de vehículo. . . . 47
3.23. Redes neuronales convolucionales famosas vs operaciones de punto flotante necesarias tomada de [Bianco et al., 2018] . . . 48
3.24. Arquitectura MobileNetV2 tomada de [Sandler et al., 2018] . . . 50
3.25. Función de coste de la arquitectura ganadora. . . . 51
3.26. Exactitud de la arquitectura ganadora. . . . 51
3.27. Función de coste de la arquitectura ganadora. . . . 52
3.28. Exactitud de la arquitectura ganadora. . . . 52

4.1. Método de extracción de fondo [Opencv-documentation, 2021] . . . 54
4.2. Algoritmo MOG2 aplicado sobre un vídeo de prueba. . . . 55
4.3. Máscara sin filtrar y sin tresholding. . . . 55
4.4. Máscara con filtro aplicado . . . 55
4.5. Diagrama de bloques del proceso de validación . . . 56
4.6. Obtención de las regiones de interés. . . . 56
4.7. Matriz de confusión de validación para la mejor solución de clasificador binario. . . . 57
4.8. Visualización de los vehículos detectados. . . . 58
4.9. Diagrama UML de la clase trackedVehicle. . . . 58
4.10. Matriz de confusión de validación generada por el clasificador de Tipo de Vehículo con el dataset de validación. . . . 59
4.11. Implementación del clasificador en el algoritmo de conteo. . . . 60
4.12. GUI con la línea de velocidad y espacio para variables. . . . 61
4.13. Ejemplo de captura de información en MongoDB usando Robo 3T. . 62
4.14. Captura video prueba 1. . . . 64
4.15. Captura video prueba 2. . . . 64
4.16. Captura video prueba 3. . . . 64

5.1. Pantalla de entrada de la interfaz web. . . . 67
5.2. Pantalla principal del aplicativo web del algoritmo de conteo de tráfico. 69
5.3. Demostración del stream de la cámara en un computador portatil. . . 70
5.4. Pantalla de gráfica de conteo total de vehículos. . . . 72
5.5. Pantalla de gráfica de conteo total de vehículos discriminados por tipo. 72
5.6. Flujo para tener un modelo compatible con Edge TPU [Coral, 2019] . 74
5.7. Dimensiones de la Google Coral Dev Board [Google, 2019] . . . 76
5.8. Perspectiva lateral. . . . 76
5.9. Perspectiva lateral. . . . 76
5.10. Perspectiva trasera. . . . 76
5.11. Perspectiva frontal. . . . 77

5.12. Perspectiva lateral. 77
5.13. Perspectiva lateral. 77

Índice de tablas

1.1. Tabla de comparación de los dispositivos embebidos disponibles para el proyecto 16

2.1. Resumen bases de datos 21

3.1. Resultados de las múltiples corridas con las arquitecturas propuestas 34
3.2. Resultados de las múltiples corridas con las arquitecturas propuestas 37
3.3. Resultados de las múltiples corridas con las arquitecturas propuestas 41
3.4. Resultados de las múltiples corridas con las arquitecturas propuestas 42
3.5. Resultados de optimización mediante algoritmo genético para Vehículo - No Vehículo 47
3.6. Resultados de optimización mediante algoritmo genético para Tipo de Vehículo 47
3.7. Operaciones de punto flotante según dispositivo embebido 49
3.8. Resultados de Transfer Learning para Vehículo - No Vehículo 51
3.9. Resultados de Transfer Learning para Tipo de Vehículo 52

4.1. Etiqueta de color junto con su valor RGB 60
4.2. Tabla de conteo y porcentajes de error en vídeo de prueba 1 63
4.3. Tabla de conteo y porcentajes de error en vídeo de prueba 2 63
4.4. Tabla de conteo y porcentajes de error en vídeo de prueba 1 63

5.1. Afectación de la precisión después de la quantización del modelo. . . 75

Resumen

En este proyecto se realizó un algoritmo de clasificación y conteo de tráfico utilizando redes neuronales como modelo de clasificación, estas se entrenaron utilizando back propagation con descenso de gradiente estocástico y con un algoritmo genético simple, el tipo de redes que se manejó fue feedforward y convolucional, esto con la idea de no dejar de lado la ingeniería de características que puede representar un desafío en tipos de problema donde no hayan modelos de aprendizaje profundo que extraigan características por ellos mismos.

Una vez obtenidos estos modelos, se procedió a realizar un algoritmo que permita utilizarlos para lograr la clasificación de una región de interés que fue obtenida a través de algoritmos de extracción de fondo para agregarle un elemento que haga seguimiento del vehículo hasta que llegue a un límite donde será contado y agregado a la base de datos teniendo ciertas características del mismo en cuenta, tal como el tipo, el color y un estimado de la velocidad con la que pasa por la calle. El tipo fue tomado con un clasificador, el color se eligió utilizando una distancia euclidiana con unos colores base y la estimación de velocidad fue hecha tomando la fórmula de velocidad promedio. La evaluación final del algoritmo mostró un error máximo del 10 % en un vídeo de prueba de más de 2000 vehículos, finalmente una plataforma web fue implementada para colocarla en un dispositivo. como adicional, ya que esto no se encuentra dentro de los alcances del proyecto, esta plataforma será descrita en su respectivo capítulo.

Introducción

Sin duda alguna el tráfico en una ciudad es un aspecto de interés para muchas disciplinas, la ingeniería no se escapa de todo esto. Bien es sabido que con el paradigma de Smart Cities, la semaforización inteligente sería uno de los aspectos más importantes para regular el flujo vehícular y así evitar atascos, sin embargo, para lograr esto es necesario tener datos estadísticos en tiempo real de que es lo que sucede en las calles, es decir, conocer el flujo vehícular por cierta avenida en particular.

Los avances de la electrónica han dado paso a un desarrollo bastante amplio y veloz en cuanto al procesamiento digital de imágenes, permitiendo hacer análisis más complejos sobre las mismas, por ejemplo distinguir que objetos se encuentran en una porción de imagen sólo con conocer sus características frecuenciales o descomposición en otro tipo de bases. Otra temática importante en este espacio es la inteligencia computacional, la cual ha tenido unos desarrollos muy ceñidos a los de la electrónica y las ciencias de la computación, dando paso a la creación de sistemas no lineales que se encargan, por ejemplo, de la clasificación de entidades basados en características propias de la entidad.

El problema principal de este proyecto es lograr obtener un algoritmo basado en modelos de inteligencia computacional que sea capaz de contar los vehículos que pasan por una calle además de obtener ciertas características de ellos como el tipo, el color, y una estimación de la velocidad con la que pasa por esa calle. Se discutirá entonces el entrenamiento de los modelos, la validación de los mismos, y como irán finalmente implementados en el algoritmo que se propondrá más adelante en una evaluación final.

Bien es sabido que a lo largo de los últimos años, las tareas de visión por computador se han trasladado del campo del procesamiento digital de la imagen hacia el de la inteligencia computacional y el aprendizaje profundo (Deep Learning de acá en adelante), dejando un poco de lado cierto tipo de algoritmos que se utilizaban para las tareas de clasificación, detección y segmentación las cuales serán definidas a lo largo de este documento.

Varios trabajos con este mismo enfoque han sido desarrollados a través de los últimos años [Ahmed et al., 2019] [Abdelwahab, 2019] [Song et al., 2019]
[Sun et al., 2017] teniendo en cuenta diferentes tipos de algoritmos de inteligencia tales como Support Vector Machines (SVM), Redes neuronales feedforward (FFNN), Redes neuronales convolucionales (CNN), entre otros, pero el objetivo sigue siendo el mismo, contar los vehículos que pasan por una calle.

A lo largo de este documento, se tratará la discusión de la selección de cámaras para un posible dispositivo futuro además de la plataforma en la que este podría implementarse, esto teniendo en cuenta varios factores como la cantidad de operacio-

nes de punto flotante por segundo, la disponibilidad para pruebas, el costo, y otros factores varios que serán explicados más adelante para generar una ponderación que permita especificar cual sería la solución más viable y porqué. Después de esto se tratará el tema de la generación de una base de datos de tipo no relacional en la cual se guarden las imágenes en carpetas separadas que irán etiquetadas según su clasificación para después generar unos datasets que serán utilizados para el entrenamiento y validación de los modelos que se propondrán en una sección posterior. Luego de todo lo que tiene que ver con la generación de la base de datos, se procedió a experimentar con dos tipos de red neuronal, feedforward y convolucional, para ver qué tan bien se comportan en las tareas de clasificación necesarias para lograr el algoritmo, el desempeño de estas se evaluará bajo el mismo indicador dependiendo de la cantidad de clases que hayan. El entrenamiento, considerando el aprendizaje como un problema de optimización, se realizó a través del algoritmo de descenso de gradiente estocástico con back propagation y utilizando un algoritmo genético simple para mirar las diferencias entre ellos, esto se propone con el fin de lograr un mejor modelo teniendo en cuenta diferentes optimizadores. Hay que aclarar que a medida que avanzó el proyecto se tomaron ciertas decisiones en cuanto a cual arquitectura pasa para probar un optimizador diferente basado en su indicador de desempeño. Se tuvo en cuenta además que ya hay ciertas redes de tipo convolucional que ya se encuentran entrenadas y forman parte del estado del arte, es decir, tienen un desempeño muy bueno comparado con otros modelos de clasificación en la actualidad, basado en esto se eligió MobileNetV2 como extractor de características para hacer un par de clasificadores y comparar su desempeño con los generados con el entrenamiento desde cero. Finalmente, una vez obtenidos los modelos se pasó a la generación del algoritmo teniendo en cuenta el cómo obtener una región de interés para que los clasificadores puedan trabajar, ya que a pesar de que se trata de una tarea de detección de objetos, no se utilizaron detectores de objetos de aprendizaje profundo tales como YOLO v3 [Redmon and Farhadi, 2018] o SSD [Liu et al., 2016] para poder hacer un tipo de procesamiento de imagen que sería, además de interesante para revisar algoritmos de extracción de fondo, una mejor oportunidad de sólo entrenar los clasificadores utilizando etiquetado sencillo en vez de codificaciones de bounding box. Una vez se obtuvo este algoritmo se pasó a la evaluación del mismo donde se hallaron resultados de 10 % de error máximo para un vídeo de prueba con más de 2000 vehículos.

Objetivos

Objetivo general

Construir e implementar el modelo de un sistema de clasificación de tráfico vehicular presente en cualquier calle de la ciudad mediante la toma de imágenes en tiempo real y un paradigma de inteligencia computacional.

Objetivos específicos

- Realizar pruebas de captura de imágenes con diferentes cámaras para establecer con cual se generará el dataset.
- Generar una base de datos de entrenamiento y validación para cualquiera de los sistemas de clasificación inteligentes, es decir, tomar el video necesario para realizar el procesamiento.
- Emplear técnicas de inteligencia computacional que permitan clasificar de forma acertada los datos que lleguen a la base de datos para que la identificación que haga el modelo sea precisa y exacta.
- Crear una base de datos que se actualice en tiempo real con la información de los vehículos que transiten por la zona a estudiar para realizar la identificación correcta de los mismos con base en los datos allí almacenados.

Capítulo 1

Pruebas de diferentes cámaras y plataformas

1.1. Sistema embebido

Es un circuito electrónico computarizado que está diseñado para cumplir una labor específica en un producto [Galeano, 2007]. Cuando se habla de sistemas embebidos se referiere a equipos electrónicos que incluyen procesamiento de datos. A diferencia de una PC (en cualquiera de sus diversos formatos), los sistemas embebidos se diseñan para satisfacer una función específica (reloj digital, reproductor de MP3, teléfono celular, router, sistema de control de automóvil –ECU– o de satélite o de planta nuclear, etc.) [Aires., 2013].

Lo que se necesita ahora es encontrar un sistema embebido capaz de llevar a cabo la tarea de inferencia mediante redes neuronales contando con que además permita el procesamiento de imagen de una manera rápida y precisa. Teniendo esto en cuenta, hay una serie de sistemas embebidos de propósito general conocidos como microcomputadores los cuales son sistemas embebidos de propósito general que implementan sistemas operativos basados en distribuciones de linux adaptadas para el procesador que llevan.

1.2. Revisión de plataformas

1.2.1. Raspberry Pi

También conocida como RPi, es una computadora de una sola tarjeta, es exactamente como suena. Es justo como un computador de escritorio, un computador portatil, o un smartphone sólo que la diferencia está en que se encuenta impresa en una única tarjeta [Halfacree, 2018]. A pesar de ser un poco vaga la definición que da Raspberry Pi en su guía para principiantes, se trata de una poderosa herramienta sobre la que corre un sistema operativo llamado Raspbian basado en una distribución de Debian Linux. Se encuentran en diferentes versiones. Desde la Raspberry Pi Model B que fue lanzada en el 2012, estas tarjetas han sido seleccionadas por el mercado debido a su tamaño y su capacidad computacional, se basan en procesadores de arquitectura ARM de 32 y 64 bits.

En los modelos más recientes de Raspberry, se tienen la Raspberry Pi 4B que se puede observar en la figura 1.1 y la Raspberry Pi Zero W que se puede observar en la figura 1.2. La diferencia principal de un modelo al otro es el procesador en el que se basan, por ejemplo, la Raspberry 4B se basa en un procesador ARM Cortex A72 mientras que la Raspberry Pi Zero W está basada en un procesador BCM2835 con menores capacidades pero listo para dispositivos wearable, los dispositivos wearable son los que una persona puede llevar sobre su ropa o elementos de uso diario.



Figura 1.1: Raspberry Pi 4B - Tomada de [RaspberryPi4, 2021]

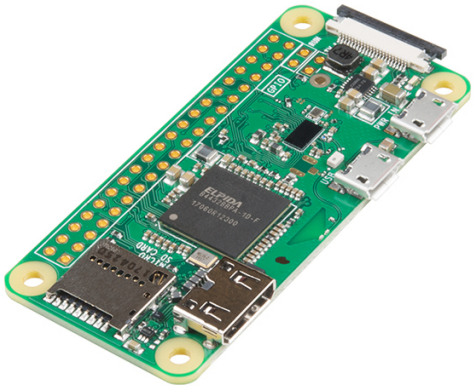


Figura 1.2: Raspberry Pi Zero W - Tomada de [RaspberryPiZeroW, 2021]

1.2.2. NVIDIA Jetson Nano

Con todo el boom de la inteligencia artificial, NVIDIA decidió no quedarse atrás y decidió fabricar un dispositivo embebido de bajo costo y tamaño pqueño llamado NVIDIA Jetson Nano, descrita como la computadora para inteligencia artificial de bajo costo, consta de un procesador ARM A57 que es de una gama un poco más baja que el de la Raspberry Pi 4B que se introdujo en la sección anterior, sin embargo, la diferencia entre estas dos tarjetas se encuentra principalmente en que la Jetson Nano viene con una GPU de arquitectura Maxwell (Registrada por NVIDIA) que permite el uso nativo de frameworks de inteligencia aftificial a una velocidad más rápida debido al paralelismo que puede trabajar este tipo de procesadores.



Figura 1.3: Tarjeta de desarrollo NVIDIA Jetson Nano - Tomada [JetsonNano, 2021]

En la figura 1.3 se puede observar la tarjeta de desarrollo ofrecida por NVIDIA para aplicaciones de inteligencia artificial, fue anunciada en Marzo de 2019, una ventaja de este kit es que compoarte la misma distribución del GPIO de la Raspberry, haciendo portables tarjetas que hayan sido desarrolladas especialmente para la otra plataforma, tiene un SDK parecido y esto permite la fácil portabilidad de código.

1.2.3. Google Coral Dev Board

Normalmente es costumbre encontrar una creación de Google para cada cosa tecnológica que se pueda venir a la mente de cualquier persona, en este caso, los computadores embebidos de tamaño pequeño no fueron la excepción. En la figura 1.4 se puede observar la creación de Google, ésta se basa en un procesador ARM A53 y lo que la hace tan especial es una unidad de procesamiento de tensores ASIC (TPU de ahora en adelante) que básicamente es una unidad de procesamiento de vectores de alta dimensionalidad para aplicaciones que requieren paralelismo en este tipo de operaciones.

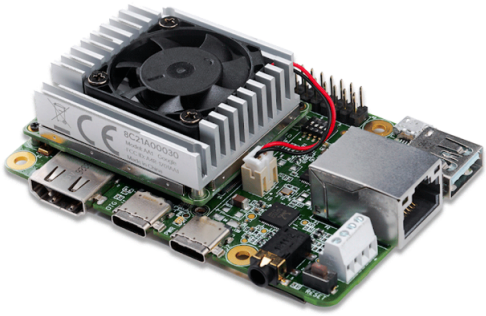


Figura 1.4: Coral Dev Board - Tomada de [GoogleCoral, 2021]

En la lámina de aluminio disipadora de calor que se ve cono el ventilador, se encuentra un chip denominado por Google como EdgeTPU el cual es capaz de correr tensores en paralelo a una gran velocidad, este chip tiene por dimensiones 10mm x 15mm lo cual lo hace perfecto para aplicaciones que requieran cálculos tensoriales de alta velocidad y además que sean dispositivos wearable o de pequeña dimensión.

1.2.4. Caracterización de los dispositivos

Una vez introducidos los diferentes dispositivos que se tienen disponibles para pruebas, se procederá a hacer una comparación de sus diferentes características para poder elegir con base en una métrica que se propondrá más adelante. Los aspectos a tener en cuenta serán:

- Cantidad de núcleos del procesador principal.
- Frecuencia del procesador principal.
- Memoria RAM.
- Posesión de GPU o TPU.
- Disponibilidad.

La cantidad de núcleos se ponderará de 1 a 5 donde 5 será el procesador con una cantidad de núcleos mayor, siguiendo esta misma lógica, la frecuencia del procesador principal se tratará de la misma manera, la RAM seguirá la misma regla mencionada anteriormente mientras que la posesión de GPU o TPU es un parámetro binario el cual tendrá valor de 0 o 5 dependiendo de si tiene o no respectivamente, la disponibilidad se evaluará de 1 a 5 donde 1 significa mucha dificultad para su uso y 5 significa uso libre durante el desarrollo del proyecto, finalmente el precio se evaluará con la misma idea de la cantidad de núcleos sólo que en vez de adicionarse al puntaje, se restará de este, por lo cual el puntaje perfecto sería de 20 puntos.

Tabla 1.1: Tabla de comparación de los dispositivos embebidos disponibles para el proyecto

Dispositivo	# de núcleos	f del procesador [GHz]	Memoria RAM	GPU/TPU	Disponibilidad	Precio [USD]	Puntaje
Raspberry Pi 4B	4	1.5	4GB	0	1	63	13
Raspberry Pi Zero W	1	1	0.5GB	0	5	27	9
Jetson Nano	4	1.43	4 GB	5	1	109	16
Coral Dev Board	4	1.5	1 GB	5	5	130	16

En la tabla 1.1 se puede observar que hubo un empate entre la Jetson Nano y la Google Coral Dev Board, esto debido a la poca ram que tiene la Google Coral Dev Board y a la poca disponibilidad que tiene la Jetson Nano para trabajar con ella, por lo tanto se elegirá la Google Dev Board como la mejor opción, teniendo en cuenta de que actualmente en el mercado existe la versión de 4GB de ram disponible por el mismo precio.

1.3. Cámara digital

Una cámara es un dispositivo capaz de captar una imagen a través de la luz capturada en un sensor a través de un lente que enfoca hacia un lugar de interés. En especial, una cámara digital se compone de un lente con cierta apertura que llevará la luz que contiene la escena hacia un sensor llamado CCD, éste es una matriz de fotodiodos con un tipo especial de filtro para captar la información RGB de la imagen a la cual se encuentra apuntando. La información es transmitida a través de una serie de protocolos entre los cuales, para sistemas embebidos se implementan las interfaces CSI, CSI-2 y en algunos más avanzados la CSI-3

1.4. Evaluación de cámaras

Debido a la situación que transcurre en los años 2020 y el 2021 se tiene la posibilidad de probar con 3 tipos de cámaras con interfaz CSI compatibles con la Raspberry Pi 4 y la única compatible con la Google Coral Dev Board que es la que se utilizará para tomar las imágenes finalmente según la sección anterior. Las pruebas se realizarán en condiciones de luz día hacia espacios varios.

Cámara Raspberry Pi v1.3

- Sensor OV5647 de 5 Megapixeles
- Distancia focal de 3.51mm
- Ángulo de apertura de 53.5 grados
- Mejor resolución del 1080p
- Interfaz CSI

Cámara Raspberry Pi v2.1

- Sensor Sony IMX219 de 8 Megapixeles
- Distancia focal de 3.04mm
- Ángulo de apertura de 62 grados
- Mejor resolución del 1080p
- Interfaz CSI

Cámara Arducam Sony IMX219

- Sensor Sony IMX219 de 8 Megapixeles
- Distancia focal de 3.04mm
- Ángulo de apertura de 62 grados

- Mejor resolución del 1080p
- Interfaz CSI

Cámara Coral Dev Board

- Sensor Omnivision OV5645 de 5 Megapixeles
- Distancia focal de 2.5mm
- Ángulo de apertura de 87 grados
- Mejor resolución del 1080p
- Interfaz CSI-2

Como se puede apreciar los sensores de la cámara v2.1 y de la cámara Arducam son los mismos lo único que cambia es que la arducam trae un lente con foco ajustable, en las figuras 1.5, 1.6, 1.7 y 1.8 se pueden ver las disposiciones y modelos de estas cámaras.



Figura 1.5: Cámara de raspberry versión 1.3, tomada de [Raspberry1.3, 2021]

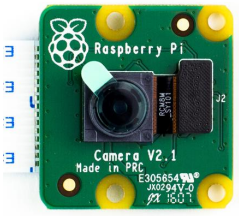


Figura 1.6: Cámara de raspberry versión 2.1, tomada de [Raspberry2.1, 2021]



Figura 1.7: Cámara arducam, tomada de [Arducam, 2021]

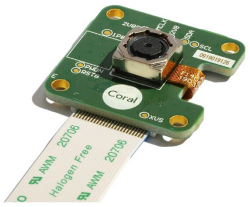


Figura 1.8: Cámara Coral Dev Board, tomada de [GoogleCoral, 2021]

Dado a que en la sección anterior se obtuvo la Google Coral Dev Board como la mejor opción para implementar el algoritmo y la interfaz que esta maneja con las cámaras es CSI-2, la única opción viable es trabajar con la cámara de la Coral Dev Board, esta también se encuentra totalmente disponible para el trabajo a lo largo del proyecto. Sin embargo, a medida de que va a avanzando la tecnología mientras se escribe este documento, se puede hacer una integración de diferentes dispositivos de aceleración de inferencia a plataformas como la Raspberry, lo cual puede dejar abierto el buffet tan grande de cámaras que tiene ésta para integrar, por lo cual se deja una pequeña revisión de varios dispositivos para trabajos futuros.

En las figuras 1.9, 1.10, 1.11, 1.12, se pueden observar respectivamente los resultados de la toma de imágenes con las cámaras Raspberry Pi Rev 1.3, Raspberry Pi Rev 2.1, Arducam y Google Coral Camera.



Figura 1.9: Imagen de prueba de la cámara de Raspberry Pi Rev 1.3.



Figura 1.10: Imagen de prueba de la cámara de Raspberry Pi Rev 2.1.

Debido a las condiciones de luz, la toma de la fotografía también puede variar, estas son cosas que deben tomarse en cuenta a la hora de tener un dispositivo que mida el flujo de tráfico, ésto debido a que en una avenida principal puede haber vehículos a cualquier hora, incluídas las altas horas de la noche. Las tres primeras fotografías fueron tomadas con la Raspberry, mientras que la última, debido a la compatibilidad, fue tomada con la plataforma Google Coral Dev Board.



Figura 1.11: Imagen de prueba de la cámara Arducam.



Figura 1.12: Imagen de prueba de la cámara Google Coral Camera.

En condiciones de luz buena, la cámara con el color más fiel, se puede apreciar que es la Raspberry Pi Rev 2.1, se puede pensar en ella para una implementación con otro tipo de sistema embebido en el futuro que tenga una GPU o una TPU para poder pensar en embebeer el algoritmo basado en modelos de inteligencia computacional.

Capítulo 2

Generación de la base de datos de entrenamiento de los modelos

2.1. Bases de datos

Para construir los modelos de clasificación vehicular implementados con inteligencia computacional es necesario tener una base de datos que a la larga no es más que un conjunto de datos que comparten información o caraterísticas que los llevan a estar en un mismo contexto, además se debe tener una organización de esa información para que exista un fácil acceso y manejo de la misma.

Actualmente dentro de las bases de datos se tienen diferentes opciones como lo son las bases de datos dinámicas y/o estáticas que permiten o no la modificación de la información que contienen, también están las bases de datos jerárquicas, de red, relacionales y no relaciones, deductivas y multidimensionales, que dependiendo de la organización que se tenga de los datos o la información que estas contengan el dataset pertenecerá a alguna de las anteriormente mencionadas.

En este proyecto la información se maneja dentro de una base de datos no relacional que se caracteriza por no trabajar con estructuras definidas, esto permite que los datos no tengan que estar almacenados en tablas, y la información tampoco se organiza en registros o campos lo cual es necesario para contruir una base de datos de imágenes y poder darle un uso correcto para el entrenamiento de los medelos de inteligencia computacional, además las imágenes se almacenan en folders o carpetas que están etiquetadas de forma que sea fácil su comprensión a nivel estético y su manejo a nivel del algoritmo utilizado.

2.2. Construcción de base de datos

El propósito inicial de este trabajo era poder generar una base de datos de entrenamiento y validación con algunos miles de imágenes extraidas de videos grabados manualmente en la ciudad de Bogotá, pero la situación de pandemia transcurrida entre los años 2020 y 2021 también causó impedimentos a la hora de querer generar esa base de datos deseada, sin embargo, se logró construir una base de datos funcional y completa que es la unión de varias bases de datos y que contiene un cúmulo importante de imágenes que tienen como fuente central las bases de datos presentes

en [Arróspide et al., 2012] y [Huttunen, 2019] de forma gratuita y que a su vez se complementaron con parches extraidos a través de un script python de los datos presentados en [Laroca et al., 2021] y también de videos tomados de forma manual desde un puente peatonal ubicado en la ciudad de Bogotá con ayuda de un celular Xiaomi Redmi Note 8, para tener un tratamiento de la información de los videos capturados primero se realizó la extracción de los frames segundo a segundo de cada uno de ellos por medio del software *vlc* y se almacenaron en carpetas para después cargarlos en la herramienta $coco - annotator$ y poder pintar las cajas o bounding boxes de cada vehículo útil para el entrenamiento y/o validación de los modelos, finalmente se descargó toda la información estos últimos en archivos con formato JSON y se procesaron mediante un script de python para obtener como resultaldo final los parches de cada vehículo anotado.

La tabla 2.1 muestra el resumen de los datos que se utilizaron para los diferentes entrenamientos realizados.

Tabla 2.1: Resumen bases de datos

Nombre referente base de datos	Numero de imágenes	Entrenamiento
Kaggle	7325	Vehículo/No Vehículo
Universidad de Madrid	17542	Tipo Vehículo
Universidad de Brasil	3485	Vehículo/No Vehículo
Colombia	997	Tipo Vehículo

Básicamente en la clasificación de vehículo y no vehículo se tienen imágenes de todo tipo de vehículos y del posible fondo que puede haber, este pueden ser postes, calle, arboles, bolardos, entre otros. Para la clasificación por tipo de vehículo se tuvieron en cuenta las clases como Ambulancia, Bicicleta, Bus, Camion, Carro, Moto, Taxi y Van; la selección de dichas clases se hizo a partir del tipo de vehículos que se pueden encontrar normalmente en vías principales de la ciudad de Bogotá y cuyo ejemplo pueden se puede tomar de la base de datos generada durante este proyecto como se explicará más adelante, además estos tipos de vehículos contienen caracteristicas muy diferentes entre si lo cual permite diferencialos de forma más clara y fácil ya sea que estén a una distancia lejana de la cámara o cerca de esta.

La diversidad de las imágenes utilizadas para este proyecto es un punto a favor que permitirá mejorar las exactitudes que pueden tener los diferentes modelos de clasificación generados a partir de la inteligencia computacional y aquí trabajados, lo anterior porque existen imágenes tomadas en diferentes ángulos, con calidades variantes en Mpx, con condiciones distintas en el ambiente que rodea a los vehículos como pueden ser por ejemplo: la posición del sol y la sombra que puede generar el vehículo, si se tiene luz natural o artificial, si está lluvioso o no, etc.

En las figuras 2.1, 2.2, 2.3 y 2.4 se presentan algunos ejemplos de las imágenes que contienen las bases de datos utilizadas, como se puede ver, hay vehículos cuya placa se alcanza a apreciar bastante bien, lo cual daría cabida en otro proyecto o en una mejore de este a hacer una detección de placa para saber si el vehículo es privado o público para incluirlo también en las estadísticas.



Figura 2.1: Ejemplo de imágenes de la base de datos de Brasil [Laroca et al., 2021].



Figura 2.2: Ejemplo de imágenes de la base de datos de Kaggle [Huttunen, 2019].



Figura 2.3: Ejemplo de imágenes de la base de datos de universidad de Madrid [Arróspide et al., 2012].



Figura 2.4: Ejemplo de imágenes de la base de datos generada durante el proyecto.

Para poder tener imágenes que fueran funcionales y permitieran tener buenos resultados en el desarrollo de este trabajo cada uno de los datasets fue escogido bajo un criterios especiales, por ejemplo, la base de datos mostrada en la figura 2.1 presenta imágenes en muy buena calidad y en ángulos donde los vehículos se ven correctamente y sus carectísticas resaltan entre unos y otros lo cual a su vez facilita el aprendizaje de los sistemas planteados; también se tiene el caso d la base de datos de la figura 2.2 que contiene dentro de todas imágenes algunas tomadas a vehículos que transitan en Colombia y aunque no son la gran mayoría esto facilita que los modelos de inteligencia computacional generados al ser probados en las condiciones de la ciudad de Bogotá sean más exactos en sus los resultados que arrojen; en tercer lugar se tiene la base de datos de la figura 2.3 la cual cuenta con imágenes de los 8 tipos de vehículos que se desean clasificar en este trabajo; por último está la base de datos de la figura 2.4 que permite complementar la información de las demás bases de datos mencionadas anteriormente con datos de vehículos colombianos y además al momento de estos ser tomados se tuvieron en cuenta los ángulos en los que podría ver un dispositvo dichos vehículos puesto en un poste de energía, un semáforo o un puente vehícular.

El tamaño de las imágenes de las bases de datos del trabajo no es una condición que se manejo como un estándar ya que por parámetros como el tipo de vehículo o la distancia a la que fue tomada la foto el tamaño puede variar entre imágenes

significativamente, pero esto no se convirtió en un impedimento ya que mediante el uso de herramientas que proporciona python como lo son OpenCV o Tensorflow se pueden reescalar todas las imágenes de las bases de datos a un valor espécifico y que además depende del modelo que se quiera implmentar, la capacidad de hardware para el procesamiento y de las necesidades que se tengan en la clasificación.

Finalmente se reunieron los dos pares de bases de datos en dos carpetas para poder utilizar cada una en su respectivo modelos de detección (Vehículo/No Vehículo y Tipo de Vehículo).

Capítulo 3

Entrenamiento y validación de los diferentes modelos

3.1. El problema de inteligencia computacional

Un problema de inteligencia computacional podría describirse como la aproximación de un fenómeno de la realidad a través de un tipo de función matemática que tenga en cuenta unas características que describan de forma clara y contundente al proceso.

Se habla de un aproximador universal cuando la función o modelo que se utiliza como aproximador cumple con el teorema de aproximación universal, en este caso, el aproximador universal que se trabajó a lo largo de esta sección es la red neuronal. En la figura 3.1 se puede ver uno de los tipos de aprendizaje que se puede configurar, se denomina aprendizaje supervisado ya que con una métrica de error, el supervisor va a corregir al aproximador universal o aprendiz, de tal manera que este minimice la métrica de error. Dependiendo del tipo de problema se elige una métrica de error que se ajuste adecuadamente.

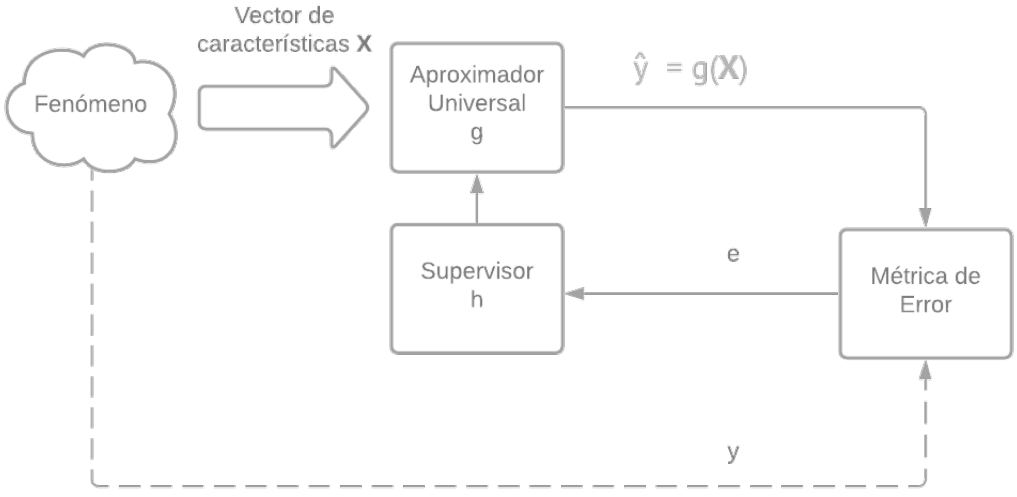


Figura 3.1: Aprendizaje supervisado

3.2. Ingeniería de características

Cuando se trata un problema de inteligencia computacional con aprendizaje supervisado, normalmente se requiere un vector de números reales o vector de características que describa el fenómeno al cual se quiere aproximar. Una característica es una representación numérica de datos en bruto, hay muchas formas de llevar datos de cualquier tipo a una representación numérica [Zheng and Casari, 2018]. Basados en esto, la ingeniería de características se describe como el conjunto de métodos que se utiliza para extraer información relevante de un conjunto de datos. Para el problema que se tratará en este documento, se revisó un método de extracción de características sobre imágenes que se explicará más adelante.

3.3. Extracción de características y entrenamiento

En esta parte del proyecto lo que se espera es obtener un buen set de características de tal manera que sea posible utilizar clasificadores que contemplan la no linealidad del espacio de características que se genera a partir de tomar ciertos puntos importantes de la imagen sin tener filtros sintonizados mediante algoritmos de optimización, lo que se nombra en los papers del estado del arte como deep learning y redes de tipo convolucional, todo esto sin dejar de lado el paradigma de deep learning donde estas características serán extraídas puramente de la información.

El principal objetivo es no dejar de lado la ingeniería de características que es una parte importante de estos proyectos donde se involucran clasificadores, para esto se comparará cual tiene mejor desempeño en validación utilizando el mismo número de épocas y la misma métrica de precisíon debido a que se tiene un set de datos balanceado, esto hace que ésta sea una buena medida del desempeño del clasificador.

3.4. Extracción de características tipo SIFT

Para la extracción de características de las imágenes se hace uso del detector-descriptor SIFT (Scale Invariant Feature Transform) por medio de la librería OpenCV ya que este permite tomar puntos de interés dentro de las imágenes para que posteriormente puedan ser usados como herramienta para representar y clasificar los objetos de las imágenes tratadas para este proyecto.

El algoritmo SIFT que puede ser encontrado en [Lowe, 2004], tiene una implementación open source que puede ser encontrada en OpenCV, básicamente este artículo menciona que el algoritmo tiene 4 pasos importantes que se encuentran a continuación:

- Detección en escala de espacio: El primer nivel de cómptuo que busca sobre todas las escalas y localizaciones en la imagen. Se implementa eficientemente utilizando la función de diferencia de Gaussianas para identificar los potenciales puntos clave que son invariantes a la escala y a la orientación.

- Localización de puntos clave: En cada ubicación candidata, un modelo detallado se ajusta para determinar ubicación y escala. Los puntos clave se seleccionan basados en sus medidas de estabilidad.
- Asignación de orientación: Una o más orientaciones son asignadas a cada punto clave basado en un gradiente de direcciones de este punto. Todas las operaciones futuras que son hechas en los datos de la imagen han sido transformadas a la orientación y localización relativa de tal manera que proveen invarianza a esta información.
- Descriptor de punto clave: Los gradientes de imagenes locales son medidos en la escala seleccionada para ese punto clave. Estos son transformados en una representación que admite diferentes niveles de distorsión y cambios de iluminación.

En resumen este algoritmo recive una imagen de entrada y puede extraer puntos clave en un espacio de $\mathbb{R}^{128}$, por lo cual por cada punto clave que se extraiga se obtendrá un vector de 128 posiciones.

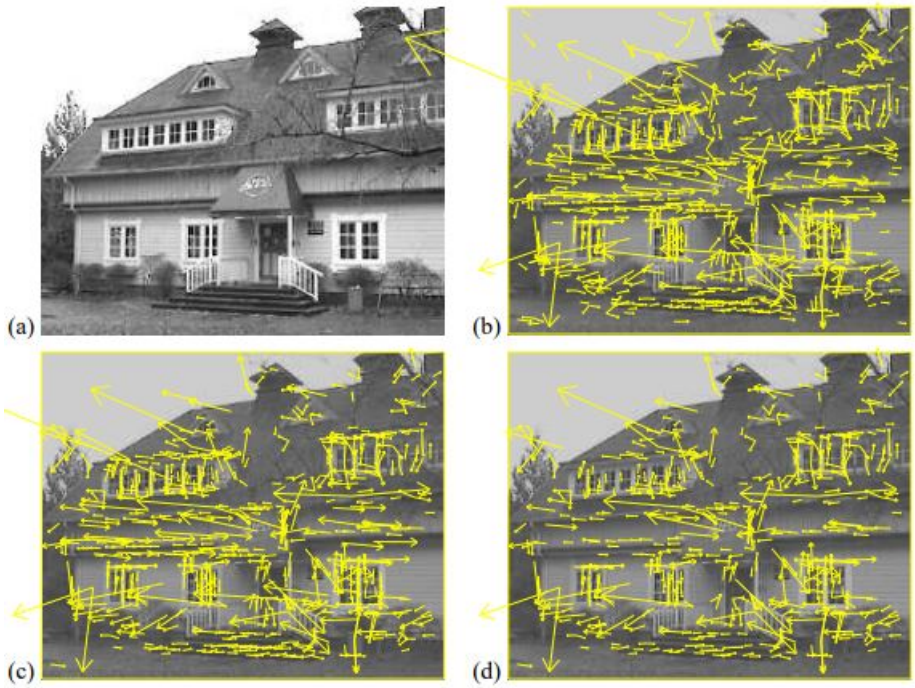


Figura 3.2: Localización y dibujo de puntos clave mediante el algoritmo SIFT [Lowe, 2004]

En la figura 3.2 se puede apreciar la localización de los puntos para dierentes escalas, de acuerdo al artículo a medida que se aplican ciertos cambios como ajuste de contraste por umbral, ajuste de curvaturas principales por umbral, varios puntos se van perdiendo, sin embargo, se conserva una gran mayoría.

Cuando se trata de clasificar imágenes, una de las técnicas más utilizadas es la de "Bolsa de palabras visuales"(Bag of Visual Words) que puede ser encontrada en [Desealers et al., 2008], básicamente consiste en tomar los puntos claves de

una cantidad de imágenes (en este caso el set de entrenamiento dispuesto para el clasificador) y utilizar algún tipo de algoritmo de clustering, en el caso del trabajo presente se utilizará el K-Means, para generar un tipo de "histograma" y basado en esto se obtendrá un vector para alimentar al perceptron multicapa.

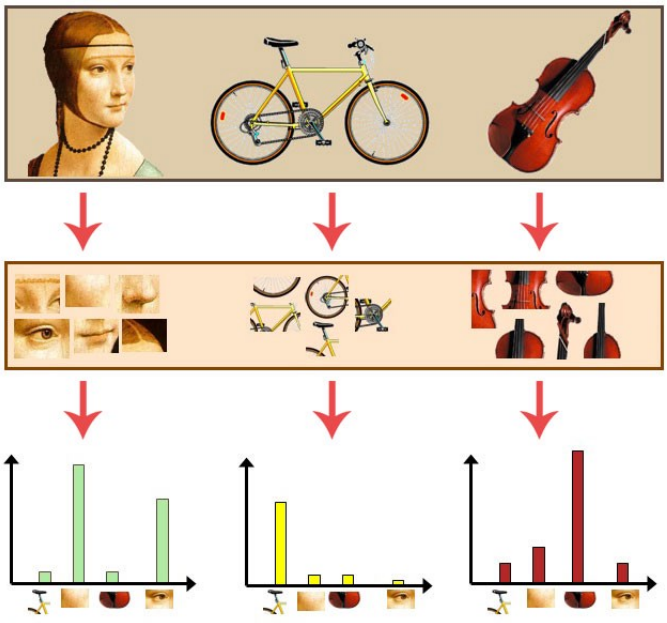


Figura 3.3: Interpretación de bolsa de palabras visuales, tomada de [Davida, 2018].

En la figura 3.3 se puede encontrar una interpretación del algoritmo de bolsa de palabras visuales, esta imagen fue tomada del sitio TowardsDataScience pero es aplicable al proyecto que se desarrolla en este documento, básicamente se toman ciertas partes de la imagen y se escanea para ver si coincide con un histograma que se genera a partir de un entrenamiento, con esto se ven las características de mayor impacto en una imagen. Además de esto en la figura 3.4 se puede encontrar el algoritmo que se propone para lograr llegar al dataset de entrenamiento y validación que se utilizará en cada uno de los clasificadores.

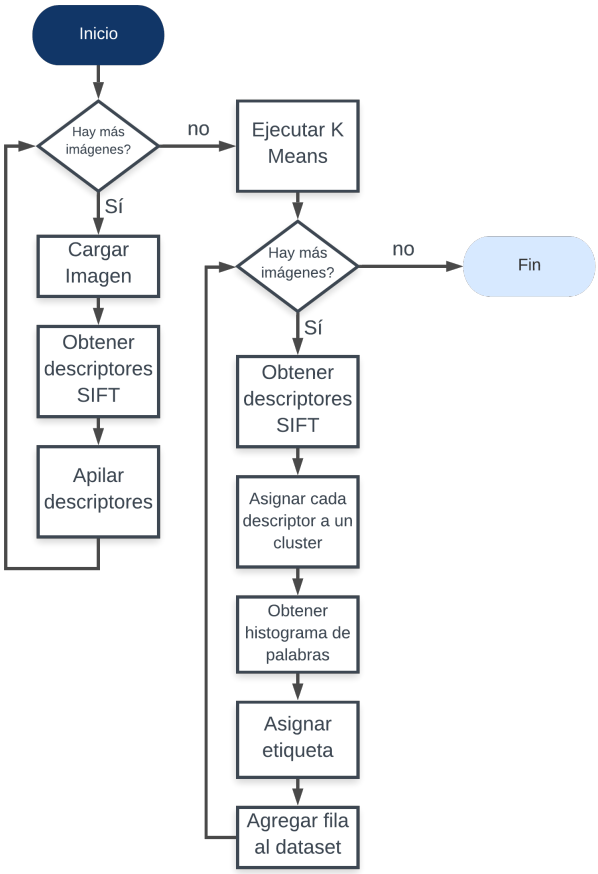


Figura 3.4: Algoritmo propuesto para obtener el dataset a partir del Bag of Visual Words

3.5. La neurona artificial

Una neurona biológica es la mínima unidad de procesamiento del cerebro humano, basados en esto, los científicos de datos e ingenieros, desarrollaron un modelo para emular este comportamiento que se veía de transmisión de información.

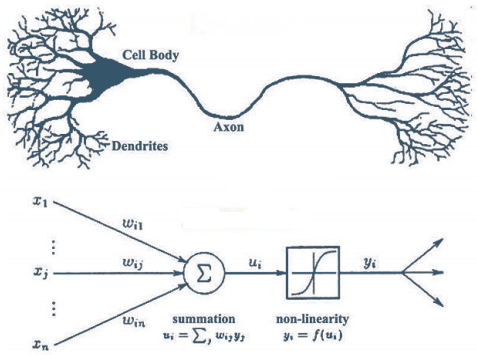


Figura 3.5: Neurona y Neurona artificial [Vrahatis et al., 2000]

En la figura 3.5 se puede observar una ilustración de una neurona real junto con su representación matemática, la representación matemática de una neurona se ve en (3.1) y está basado en el funcionamiento biológico de la misma. La neurona biológica tiene unos estímulos que se recogen a través de sus dentritas, de alguna manera se procesan enre el cuerpo de la célula y finalmente son conectados por el axón hacia otras neuronas. De la misma manera, los estímulos en este caso serían un vector de entradas X, que tiene una dimensionalidad dada, luego se pondera mediante unos pesos $\omega_{i,j}$ y finalmente esta ponderación se pasa a través de una no linealidad para que la neurona artificial sea capaz de aproximar cierto tipo de funciones.

$$y = \sigma(\sum_{i=1}^{n} \omega_i x_i) \tag{3.1}$$

Adicional a esto, al diagrama de la figura 3.5 se le puede adicionar una desviación o bias que modificará a (3.1) agregándole un término que bajará o subirá su offset.

3.6. El perceptrón multicapa

Modelo eficiente para el reconocimiento estadístico de patrones basado en una representación matemática de un sistema biológico de procesamiento de información [Bishop, 2006a]. Las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es "un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona" [Jorge, 2001].

Un ejemplo de red neural se presenta en la Figura 3.6, donde se observa la organización que puede tener una red neural en la que sus neuronas están interconectadas y organizadas por capas (Entrada, ocultas y salida).

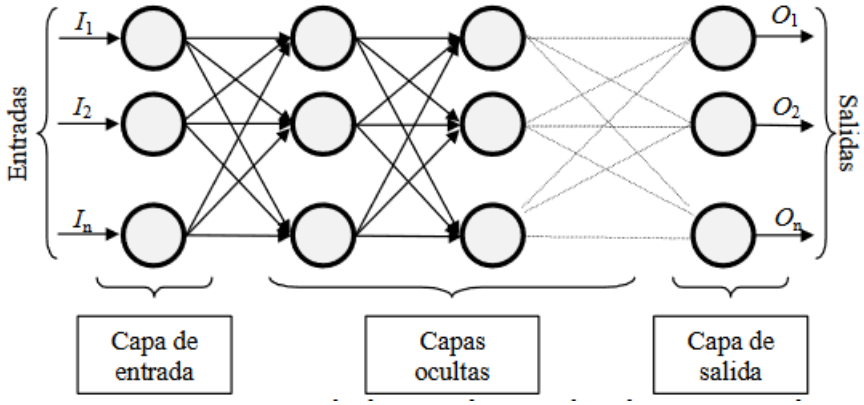


Figura 3.6: Ejemplo de red neuronal totalmente conectada [Jorge, 2001].

3.7. El aprendizaje como problema de optimización

Cuando se trata el aprendizaje mediante modelos que requiere que se varíen ciertos parámetros con el fin de lograr un máximo o un mínimo, se está hablando de un problema de optimización. Centrándose en la perspectiva del cálculo diferencial, uno halla un máximo o un mínimo cuando la primera derivada de una función matemática, definida continua, se vuelve cero, dependiendo de su segunda derivada se puede catalogar como mínimo o máximo. Normalmente en problemas de optimización un poco más elaborados, no es trivial la solución de ecuaciones que surgen de las derivadas o llegar a un valor fijo donde se encuentre un mínimo o un máximo.

El entrenamiento de un modelo de inteligencia computacional, en este caso una red neuronal, se logra a través de distintos algoritmos de optimización o heurísticas cuyo objetivo es minimizar el error como se ve en la figura 3.1. Los parámetros de este aproximador universal se ajustarán de acuerdo a ciertos algoritmos que hacen que el error tienda a minimizarse.

Considérese la figura 3.7 donde se muestra una superficie del error de un modelo que tiene como parámetros ω_1 y ω_2, a medida de que se varían estos parámetros de manera correcta, se va a llegar a un mínimo global o local. Como es un problema de 3 dimensiones, es sencillo imaginar o dibujar una superficie, sin embargo, normalmente los problemas de inteligencia computacional en su mayoría gozan de una dimensionalidad más alta, por lo cual, no es posible ver siempre este paisaje de optimización.

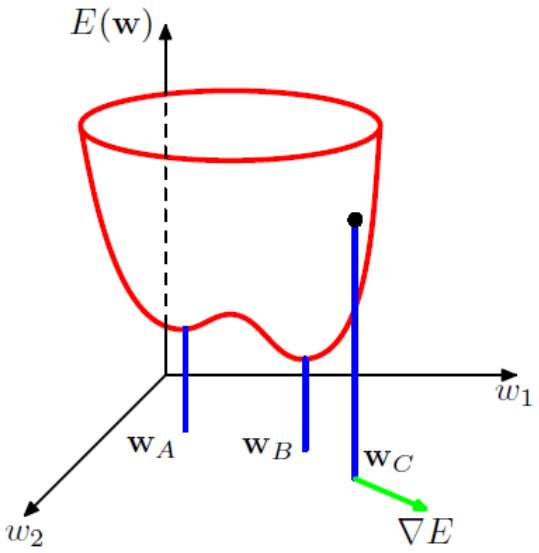


Figura 3.7: Métrica de error en función de dos parámetros
[Bishop, 2006b]

Al final de cuentas, tampoco es posible siempre obtener una ecuación que tenga una solución analítica, por lo cual se recurre a otros métodos numéricos como el método de Newton-Rhapson, el método de la bisección, entre otros que están fuera

del enfoque del proyecto, sin embargo, sí que se revisará el algoritmo de optimización con el cual se desarrolló el proyecto.

Dependiendo de la tarea que se vaya a realizar sea regresión, clasificación ó predicción se pueden tener diferentes funciones de pérdida que se acomoden mejor al problema, en los casos de clasificación binaria la entropía binaria cruzada es una buena opción para datasets balanceados y para clasificación con múltiples clases la entropía categórica cruzada puede ser una buena opción también.

3.8. Descenso de gradiente estocástico

Para hablar del descenso de gradiente estocástico, primero es necesario hablar del algoritmo de descenso de gradiente. Normalmente en funciones multivariadas, el vector gradiente da la dirección de más rápido crecimiento de la función, por lo tanto, si se toma el negativo de este gradiente, se podría hallar la dirección de más rápido decrecimiento de la función. Teniendo esto en mente se propone una actualización de parámetros de tal manera que se siga la dirección del gradiente.

$$\omega[k+1] = \omega[k] - \eta \nabla F(\omega[k]) \tag{3.2}$$

En (3.2) se puede ver que la actualización de parámetros se dará teniendo en cuenta la dirección de más rápido decrecimiento, el factor η se le conoce en el mundo de inteligencia computacional como la tasa de aprendizaje, si se considera pequeña, lo más probable es que $F(\omega[k+1]) \leq F(\omega[k])$.

Cuando se dé este algoritmo para aprendizaje, se toma cada muestra de la base de datos y se actualizan los parámetros de acuerdo a (3.2) y cada iteración por la base de datos se le conoce como época. La modificación que propone el descenso de gradiente estocástico, es que no se sigue el orden que tiene la base de datos sino que se escogen muestras al azar de esta para ejecutar la actualización de parámetros, una característica de este algoritmo es el ruido en las curvas de entrenamiento, sin embargo, el azar puede hacer que se llegue en menor número de épocas a la misma exactitud o función de pérdida que el descenso de gradiente ordinario.

3.9. Entropía binaria cruzada

Gracias a las funciones de activación que pueden tener las neuronas en la capa de salida, su valor se puede asociar al valor de confidencia o confiabilidad con el que se clasifica una muestra de la base de datos como perteneciente a una clase. Con ayuda de la función de activación sigmoidal, se pueden tener valores entre 0 y 1 que se asociarán a las características anteriormente mencionadas, esto da paso a la entropía binaria cruzada que se describe como función de pérdida en (3.3).

$$J_{bce} = -y_m log(g(x_m)) - (1 - y_m) log(1 - g(x_m)) \tag{3.3}$$

El primer término incentiva la no aparición de falsos positivos durante el entrenamiento, de tal manera que si el elemento pertenece a la clase y el modelo lo clasifica como que no pertenece ese término se hará grande y positivo por el negativo

que tiene al principio. Por otro lado, el segundo término de la función incentiva la aparición de verdaderos negativos, si el elemento realmente no pertenece a la clase y el modelo la clasifica como que sí pertenece, la función aumenta y el algoritmo de optimización debería penalizar estos aumentos [Ho and Wookey, 2020].

Esta función de pérdida se asocia a la probabilidad de que el elemento pertenezca o no a una clase de manera binaria ya que sólo debería aportar dos posibles valores de verdad.

3.10. Entropía categórica cruzada

Como una extensión de la entropía binaria cruzada existe la entropía categórica cruzada, esta función se centra principalmente en el primer término de (3.3) y lo repite tantas veces como clases hayan, esto para incentivar la no aparición de falsos positivos en cada clase, en (3.4) se puede apreciar la función de pérdida de la entropía categórica cruzada [Ho and Wookey, 2020].

$$J_{cce} = -\sum_{k=1}^{K} y_m^k log(g^k(x)) \tag{3.4}$$

Se utiliza con la función de activación softmax a la salida de tal manera que esté asociada a la probabilidad de pertenencia a la clase real.

3.11. Implementación de clasificadores

Para la clasificación de las imágenes se usaron cuatro métodos los cuales son el perceptrón multicapa, red neuronal convolucioanl, clasificador evolucionado por medio de un algoritmo genético simple, red convolucional entrenada por transfer learning.

El desarrollo e implementación de estos modelos de inteligencia computacional fue hecho mediante la librería TensorFlow que ofrece Python para la elaboración de matemática con tensores utilizando unidades de procesamiento tales como la GPU o TPU que permiten un alto paralelismo en los cálculos debido a los núcleos de sus procesadores.

TensorFlow ofrece un paquete el cual integró a su funcionamiento cuyo nombre es Keras, está más orientado hacia el desarrollo de algoritmos de inteligencia computacional tales como las redes neuronales profundas, por lo cual, para la implementación de las redes neuronales se utilizó principalmente Keras y para el resto el desarrollo se hizo en TensorFlow.

3.12. Clasificador de perceptron multicapa

3.12.1. Clasificador de No Vehículo/Vehículo

La principal tarea es distinguir si la región de interés que se está pasando al clasificador es un vehículo o es parte del camino (ya que puede haber ruido en la extracción del fondo) para esto se tomó la metodología de Bag of Visual Words, en la figura

3.8 se puede ver una gráfica de la cantidad de clusters utilizados para la técnica junto con la varianza que produce en el K Means al hacerse, por lo cual utilizando el criterio del codo se puede decir que con 1500 clusters es suficiente para hacer una separación con la carga computacional suficiente sin malgastar recursos, esto se debe a que si trazamos una recta secante entre número de clusters = 0 y número de clusters = 1000 su pendiente es mucho mayor en magnitud a la de la recta secante con número de clusters = 1000 y número de clusters = 2000, a pesar de que la varianza puede seguir disminuyendo, el número de características visuales para distinguir entre si la imagen es o no un vehículo no debería tener un diccionario de palabras tan grande para menor procesamiento.

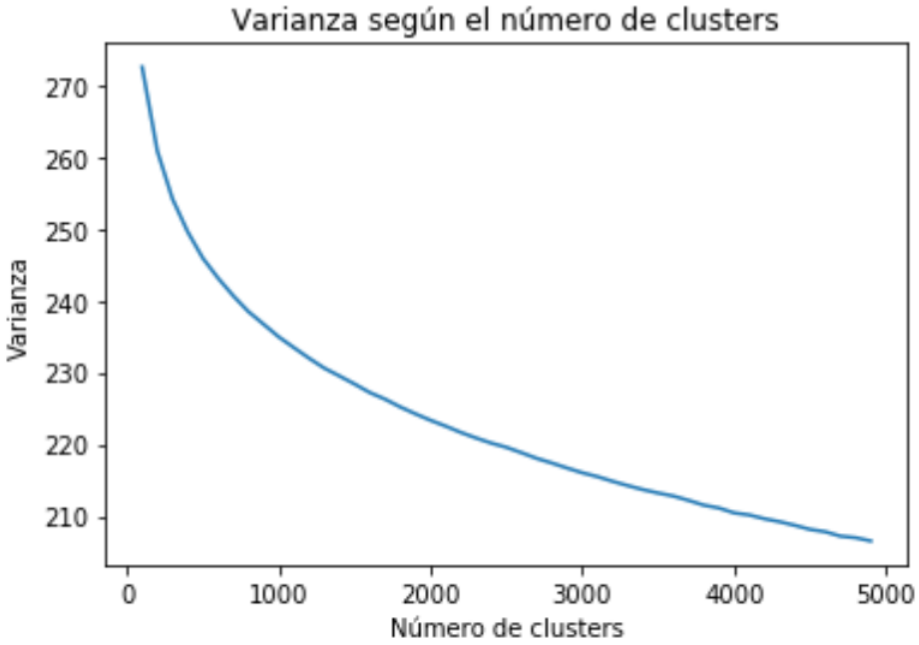


Figura 3.8: Disminución en la varianza de los centroides utilizados para BoVW a medida que varían los clusters.

Independiente del número de clusters del criterio del codo, las arquitecturas con las que se evaluará el desempeño de la técnica utilizada se nombrarán en la lista a continuación, hay que tener en cuenta que todas las redes tendrán en este caso posibilidad de salida No vehículo o Vehículo.

- Red neuronal con una única capa oculta con el doble de neuronas que la entrada y tasas aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una única capa oculta con la misma cantidad de neuronas que la entrada y tasas aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una única capa oculta con la mitad de neuronas que la entrada y tasas aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con dos capas ocultas, las dos capas con el mismo número de neuronas que la capa de entrada y tasas aprendizaje de 0.01, 0.001 y 0.0001.

- Red neuronal con dos capas ocultas, la primera con el mismo número de neuronas que la capa de entrada, la segunda con el doble de neuronas de la capa de entrada y tasas aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con dos capas ocultas, la primera con el mismo número de neuronas que la capa de entrada, la segunda con la mitad de neuronas de la capa de entrada y tasas aprendizaje de 0.01, 0.001 y 0.0001.
- Red Neuronal con dos capas ocultas, la primera con la mitad de neuronas de la capa de entrada, la segunda con la mitad de neuronas de la primera capa y tasas aprendizaje de 0.01, 0.001 y 0.0001.

Para los siete casos mencionados anteriormente se mostrará la tasa de aprendizaje, el valor de la función de pérdida final en validación para esa tasa y finalmente el valor de la exactitud en validación. Además para evitar saturar con gráficas sólo se pondrán las gráficas de la mejor arquitectura a la mejor tasa de aprendizaje.

En la tabla 3.1 se encuentran los resultados de los casos mencionados en la lista anterior de experimentos sobre diferentes redes neuronales, estos experimentos se realizaron con 100 épocas y batches de 20 elementos, además de esto cada arquitectura se corrió 100 veces con pesos inicializados aleatoriamente.

Además de esto como se presentaron los promedios de las métricas de las arquitecturas en su mejor época, se utilizarán para hacer un próximo análisis para escoger la mejor arquitectura.

Tabla 3.1: Resultados de las múltiples corridas con las arquitecturas propuestas

Arquitectura	Número de neuronas en capa oculta 1	Número de neuronas en capa oculta 2	Tasa de aprendizaje	μ coste	σ coste	μ exactitud	σ exactitud
1	3000	0	0.01	0.3410	0.026	88.91 %	2.28 %
1	3000	0	0.001	0.3514	0.020	88.34 %	1.96 %
1	3000	0	0.0001	0.3874	0.021	86.39 %	1.92 %
2	1500	0	0.01	0.3076	0.034	88.91 %	3.12 %
2	1500	0	0.001	0.3162	0.024	88.63 %	2.15 %
2	1500	0	0.0001	0.3598	0.0361	85.90 %	3.32 %
3	750	0	0.01	0.2945	0.025	88.79 %	2.13 %
3	750	0	0.001	0.2997	0.023	88.74 %	2.21 %
3	750	0	0.0001	0.3621	0.030	85.12 %	2.68 %
4	1500	1500	0.01	0.3600	0.031	88.20 %	2.84 %
4	1500	1500	0.001	0.3861	0.025	86.57 %	2.07 %
4	1500	1500	0.0001	0.3965	0.029	85.53 %	2.30 %
5	1500	3000	0.01	0.4544	0.031	88.87 %	2.58 %
5	1500	3000	0.001	0.4489	0.021	86.08 %	2.06 %
5	1500	3000	0.0001	0.4620	0.025	85.37 %	2.01 %
6	1500	750	0.01	0.3328	0.020	87.98 %	1.90 %
6	1500	750	0.001	0.3512	0.021	86.98 %	1.95 %
6	1500	750	0.0001	0.3714	0.010	85.40 %	0.56 %
7	750	375	0.01	0.3137	0.022	87.84 %	1.79 %
7	750	375	0.001	0.3223	0.018	87.60 %	1.62 %
7	750	375	0.0001	0.3815	0.015	83.97 %	0.63 %

En la tabla 3.1 se puede observar que la arquitectura con una función de coste más baja es la número 3 con una tasa de aprendizaje de 0.01, con ella se logró una función de pérdida promedio de 0.2945 y una exactitud promedio de 88.79 %, a continuación en las imágenes 3.9 y 3.10 se puede observar el resultado de los experimentos.

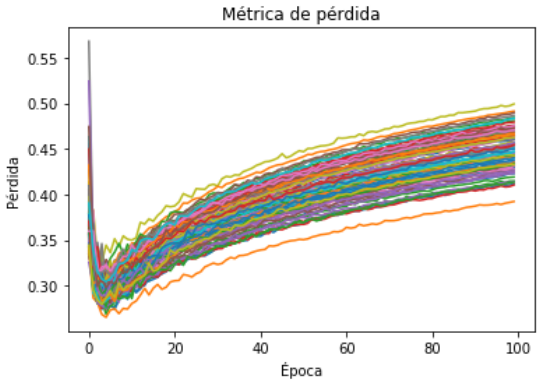


Figura 3.9: Función de coste de la arquitectura ganadora.

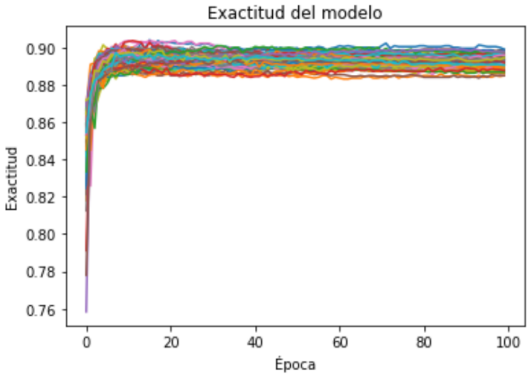


Figura 3.10: Exactitud de la arquitectura ganadora.

Como se puede ver, hubo un overfitting desde aproximadamente la época 5 y lo que se hizo fue con ayuda de la API de Keras, tomar el mejor modelo en validación de cada curva que se puede apreciar ahí, así se sacaron los valores en la tabla 3.1.

3.12.2. Clasificador de tipo de vehículo

Al igual que en la clasificación anterior se debe hacer K Means con las características extraídas para agruparlas de tal manera que sea posible hacer el Bag of Visual Words, por lo cual también debe analizarse el codo de la figura 3.11, como esta región se encuentra al rededor de los 600 o 700 clusters, para no dejar ciertas características por fuera, dejamos el número en 1500 para no exagerar tampoco la situación.

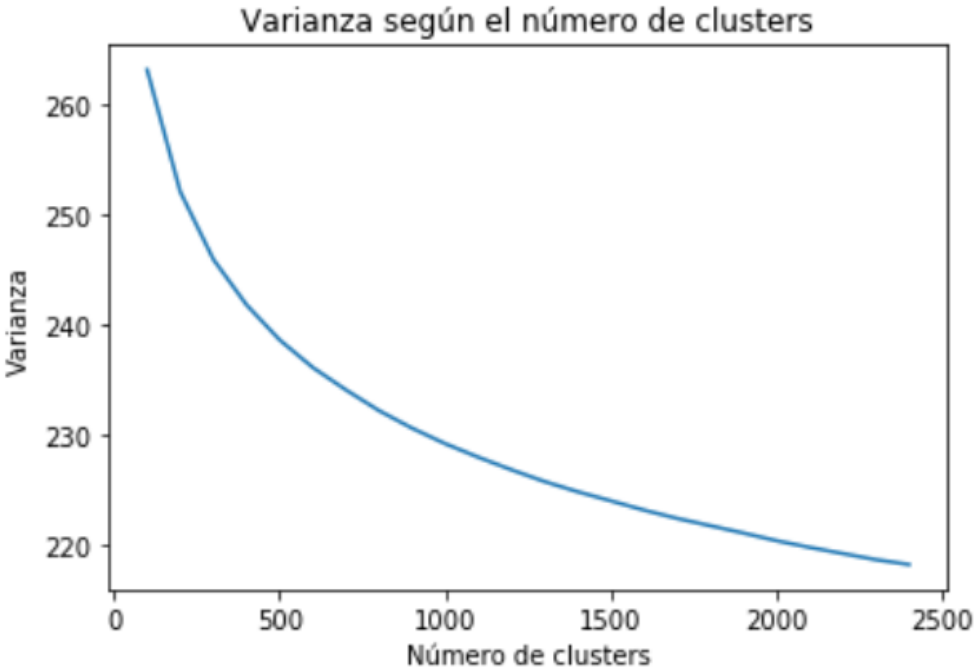


Figura 3.11: Varianza según el número de clusters para la clasificación de tipo de vehículo.

En esta sección lo que se hará es implementar un clasificador capaz de discernir si la imagen con la que se está trabajando (después de ser clasificada como vehículo) entra en las categorías de bicicleta, bus, camion, camioneta, carro, moto, vehículo no motorizado como remolques, Tractomulas o finalmente van.

Considerando que la mejor arquitectura en el clasificador implementado anteriormente fue la de K-K/2-K/4 siendo K el número de clusters del diccionario de palabras visuales, se seguirá utilizando esta arquitectura de $K/2^n$ pero esta vez los experimentos serán agregando más capas que cumplan con esta regla. Como el dataset está separado de manera diferente y se agregaron más vehículos para hacer la distinción de manera correcta.

Para este clasificador tendremos en cuenta los siguientes experimentos teniendo en cuenta que la elección de tasas de aprendizaje correctas parece estar entre el 0.01 y el 0.001:

- Red neuronal con una capa oculta con la mitad de neuronas que en la entrada con tasas de aprendizaje de 0.01, 0.005 y 0.001.
- Red neuronal con dos capas ocultas, la primera con la mitad de neuronas que en la entrada, la segunda con la mitad de neuronas que la primera y con tasas de aprendizaje de 0.01, 0.005 y 0.001.
- Red neuronal con tres capas ocultas, la primera con la mitad de neuronas que en la entrada, la segunda con la mitad de neuronas que la primera, la tercera con la mitad de neuronas que la segunda y con tasas de aprendizaje de 0.01, 0.005 y 0.001.
- Red neuronal con cuatro capas ocultas, la primera con la mitad de neuronas que en la entrada, la segunda con la mitad de neuronas que la primera, la tercera con la mitad de neuronas que la segunda, la cuarta con la mitad de neuronas que la tercera y con tasas de aprendizaje de 0.01, 0.005 y 0.001.

Estos 4 experimentos de nuevo se harán con 100 épocas y con batch de 20 elementos, sin embargo como se tiene una clasificación que ya no es binaria se procederá a cambiar la función de coste por una entropía cruzada categórica y la función de activación de la capa de salida con 8 neuronas.

En la tabla 3.2 se pueden encontrar los resultados de los experimentos que se corrieron, hay que recordar que con respecto a la anterior cambió la función de pérdida y la cantidad de neuronas en la salida, además como son clases excluyentes, también se cambió la función de activación en la salida a softmax.

El hecho de cambiar la función de pérdida obedece a que la anterior era una clasificación binaria y la que se describe en esta sección es una descripción de más de un tipo de clase, por lo tanto, se utilizó la función de pérdida entropía cruzada categórica.

Tabla 3.2: Resultados de las múltiples corridas con las arquitecturas propuestas

Arquitectura	Capa 1	Capa 2	Capa 3	Capa 4	lr	μ Coste	σ Coste	μ Exactitud	σ Coste
1	750	0	0	0	0.01	1.7714	0.21	52.05 %	1.25 %
1	750	0	0	0	0.005	1.7454	0.16	52.72 %	0.68 %
1	750	0	0	0	0.001	1.7458	0.12	51.74 %	0.50 %
2	750	375	0	0	0.01	1.6463	0.23	61.44 %	1.16 %
2	750	375	0	0	0.005	1.5181	0.18	59.13 %	0.86 %
2	750	375	0	0	0.001	1.6714	0.16	59.30 %	0.63 %
3	750	375	185	0	0.01	1.6131	0.20	62.35 %	1.12 %
3	750	375	185	0	0.005	1.6235	0.18	63.21 %	0.90 %
3	750	375	185	0	0.001	1.6348	0.16	62.90 %	0.66 %
4	750	375	185	93	0.01	1.5984	0.22	62.29 %	1.26 %
4	750	375	185	93	0.005	1.6096	0.14	62.64 %	0.56 %
4	750	375	185	93	0.001	1.6222	0.16	62.37 %	0.67 %

Como se puede apreciar, si se basa el mejor resultado en la función de coste, la mejor arquitectura en este caso es la arquitectura 4 con una tasa de aprendizaje de 0.01, las imágenes de este entrenamiento se pueden ver en las figuras 3.12 y 3.13.

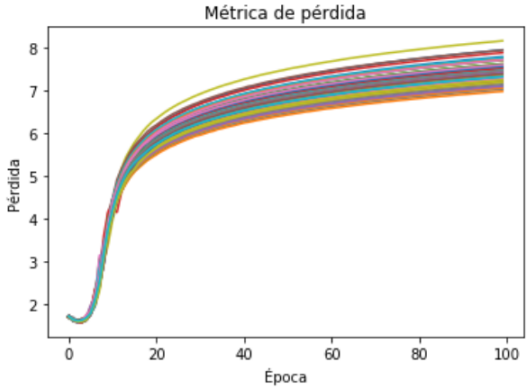


Figura 3.12: Función de coste de la arquitectura ganadora.

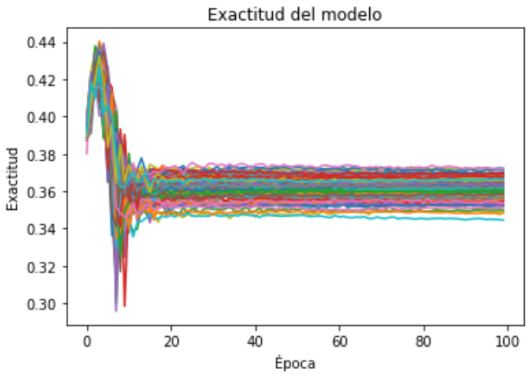


Figura 3.13: Exactitud de la arquitectura ganadora.

Sin embargo, también se puede observar que se presenta demasiado overfit en una época demasiado temprana, esto se puede deber a una gran cantidad de factores en el dataset generado tales como, columnas redundantes, outliers, entre otras cosas que se espera sean corregidas al momento de utilizar las redes convolucionales que son especialmente para el tratamiento de imágenes. Al igual que en el punto anterior, con la ayuda de Keras se guardó el mejor modelo de cada curva para sacar los valores.

Este sería el que se utilizará para discernir el tipo de vehículo que se detectó como vehículo con el clasificador anterior.

3.13. Imagen digital

Una imagen puede ser definida como una función de dos dimensiones $f(x, y)$, donde x y y son coordenadas espaciales y $f(x, y)$ la intensidad de gris en cualquier par

de coordenadas (x, y). Cuando x, y, f son cantidades discretas, la imagen es digital [Gonzalez and Woods, 2002].

Las imágenes digitales, ya sean generadas por ordenador o creadas por medio de un aparato de captura (cámara, escáner, etc.), suponen la traducción de valores de luminosidad y color a un lenguaje que pueda entender el ordenador y los periféricos relacionados con el. Todo esto supone una ventaja principal, estabilidad: los ceros y unos que forman una imagen digital permanecerán estables, y así la imagen no variará durante el transcurso del tiempo. Así todo, la tecnología todavía no puede obtener los resultados que nos ofrecen las imágenes analógicas de las cámaras 'tradicionales' [Ferrero and Mora, 2008].

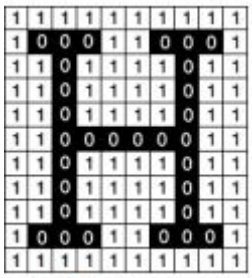


Figura 3.14: Proceso de digitalización [Ferrero and Mora, 2008].

Como se muestra en la Figura 3.14 para realizar la digitalización de una imagen es necesario tomar una muestra de la misma y se almacena en un mapa que parte de una matriz de puntos(pixeles), cada pixel tiene un valor tonal (negro o blanco).

3.14. Filtro digital bidimensional

Al igual que en las señales de una sola dimensión, en imágenes existen filtros los cuales son otra señal bidimensional pero de más baja dimensión usualmente. Para utilizar estos filtros se hace necesaria una operación de convolución entre la imagen cuya característica quiere ser ampliada mediante el filtro y el filtro en sí. En imágenes comunmente se tiene el uso de filtros de media aritmética, filtros gaussianos, filtros de suavidad, filtros de mediana, entre otros. Este conocepto es importante debido a que la red neuronal convolucional hace uso de filtros o kernels como los llama para poder hacer una extracción de características por si misma.

3.15. Red neuronal convolucional

Una red neuronal convolucional donde la señal de entrada se alimenta a través de una serie de capas de filtros digitales y de agrupación[Li and Venkatesan, 2017]. En la figura 3.15 se puede ver una de las tareas convencionales de este tipo de redes convolucionales.

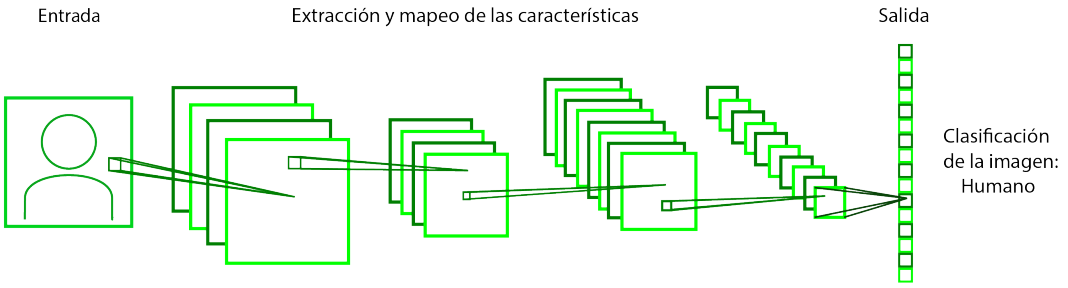


Figura 3.15: Red neuronal convolucional para tarea de clasificación, tomada de [Vertebrar, 2021]

Básicamente lo que se tiene es una serie de capas de convolución con unos filtros cuyos parámetros se aprenden mediente aprendizaje supervisado, se define la cantidad de filtros en una capa y se producirá la misma cantidad de mapas de características, los cuales son la imagen después de ser filtrada. Para una tarea de clasificación, se agregan capas totalmente conectadas para lograr obtener una salida que esté asociada a la probabilidad de que la imagen pertenezca a una clase.

3.16. Clasificador con red neuronal convolucional

Ya que las redes neuronales convolucionales se utilizan más para la clasificación de imágenes debido a la capacidad que tienen de extraer características sin perder la noción de que una imagen es más no es sólo un vector, sino de que es una matriz de puntos con color e intensidad, se implementan también y principalmente los clasificadores de imágenes haciendo uso de este tipo de redes.

Para los experimentos que se van a trabajar en esta parte, se tienen bastantes hiperparámetros, lo cual dificulta un poco la elección de algunos de ellos, sin embargo para facilitar un poco el cálculo en el procesado, se harán las siguientes conseciones:

- Las capas ocultas de la red utilizarán función de activación ReLU.
- Se utilizará tamaño de filtro impar de hasta 7x7.
- La cantidad de filtros utilizados será potencia de 2.

Los experimentos que se proponen en este caso van a depender de los resultados que se pongan en el anterior, no hay que perder de vista que cuando se habla de redes convolucionales hay que tener en cuenta varias cosas como el padding, el striding y si se habla de same o valid convolution, en este momento se hará la selección de estos parámetros para los experimentos:

- Stride: El stride es qué tantos espacios se mueve el filtro al ser colocado sobre la imagen, en los casos que se trabajarán, el stride será de 1.
- Padding: El padding refiere a que cuando uno tiene una imagen de 100x100 y le aplica un filtro de 2x2 le va a resultar una imagen de 98x98, esto provoca

una pérdida de información pero también una reducción en las dimensiones de la imagen la cual puede ser provechosa para el cálculo en el algoritmo, por lo cual tendremos 0 padding.

- Convolution: Al momento de seleccionar el padding como cero, se cae también en el hecho de utilizar convoluciones válidas, tal cual como se trabaja en el procesamiento de imagen.

Para los tipos de clasificadores con red convolucional que se trabajan en esta parte, el tamaño de la imagen de entrada será de 200x200 en formato RGB para no perder la información del color y así tener una mejor clasificación, por lo tanto el tensor de entrada tendrá dimensiones 200x200x3.

3.16.1. Clasificador de Vehículo/No Vehículo

Para este clasificador se realizarán los siguientes experimentos con el fin de ver cual es la arquitectura que resulta con los mejores resultados en validación:

- Red neuronal con una capa de convolución con 16 filtros de 3x3 y una capa de max pooling con tasas de aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una capa de convolución con 16 filtros de 5x5 y una capa de max pooling con tasas de aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una capa de convolución con 16 filtros de 7x7 y una capa de max pooling con tasas de aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una capa de convolución con 32 filtros de 3x3 y una capa de max pooling con tasas de aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una capa de convolución con 32 filtros de 5x5 y una capa de max pooling con tasas de aprendizaje de 0.01, 0.001 y 0.0001.
- Red neuronal con una capa de convolución con 32 filtros de 7x7 y una capa de max pooling con tasas de aprendizaje de 0.01, 0.001 y 0.0001.

Todas las redes mencionadas anteriormente llevan una capa Flatten que se utiliza para volver los datos que se tienen un único vector para poderlas clasificar con una red neuronal feedforward que irá al final de esta, no se agregarán capas ocultas a esta red sino que se trabajará con la parte convolucional.

Tabla 3.3: Resultados de las múltiples corridas con las arquitecturas propuestas

Arquitectura	# Filtros C1	Tamaño Filtro	lr	μ Coste	σ Coste	μ Exactitud	σ Exactitud
1	16	3x3	0.01	0.1357	0.017	95.50 %	0.73 %
1	16	3x3	0.001	0.1787	0.011	93.31 %	0.611 %
1	16	3x3	0.0001	0.2819	0.012	88.59 %	0.640 %
2	16	5x5	0.01	0.1633	0.033	94.59 %	1.27 %
2	16	5x5	0.001	0.1662	0.011	93.73 %	0.54 %
2	16	5x5	0.0001	0.2532	0.014	89.74 %	0.76 %
3	16	7x7	0.01	0.1931	0.035	93.50 %	1.42 %
3	16	7x7	0.001	0.1654	0.014	94.03 %	0.60 %
3	16	7x7	0.0001	0.2272	0.013	90.90 %	0.74 %
4	32	3x3	0.01	0.1255	0.012	95.97 %	0.466 %
4	32	3x3	0.001	0.1674	0.008	93.71 %	0.43 %
4	32	3x3	0.0001	0.2610	0.011	89.56 %	0.61 %
5	32	5x5	0.01	0.1324	0.016	95.60 %	0.69 %
5	32	5x5	0.001	0.1563	0.023	94.14 %	0.82 %
5	32	5x5	0.0001	0.2309	0.11	90.97 %	0.60 %
6	32	7x7	0.01	0.1715	0.025	94.18 %	0.86 %
6	32	7x7	0.001	0.1550	0.013	94.48 %	0.64 %
6	32	7x7	0.0001	0.2090	0.012	91.67 %	0.638 %

Teniendo en cuenta los resultados de la tabla 3.3, se podría afirmar que la arquitectura con un mejor promedio de coste, por lo tanto la ganadora, es la arquitectura numero 4 con una tasa de aprendizaje de 0.01, los resultados de estos entrenamientos se pueden ver en las figuras 3.16 y 3.17, como se puede observar acá sí se presenta una mejor optimización de la función de pérdida a medida que van pasando épocas del entrenamiento.

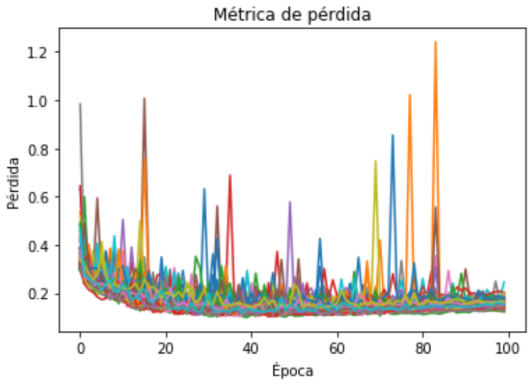


Figura 3.16: Función de coste de la arquitectura ganadora.

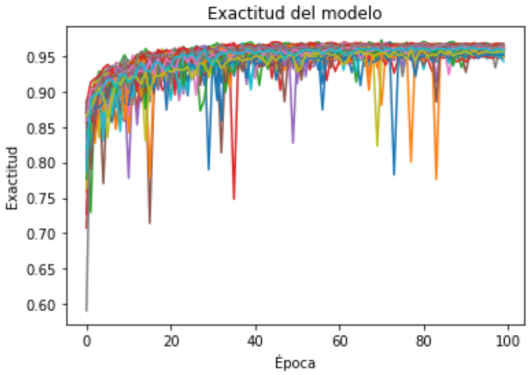


Figura 3.17: Exactitud de la arquitectura ganadora.

3.16.2. Clasificador para tipo de vehículo

Con base en los resultados de la tabla 3.3, se puede ver que la arquitectura que presentó el mejor desempeño de generalización fue la arquitectura número 4 que consta de 32 filtros de 3x3 en la capa número uno, cero filtros en la segunda capa y con tasa de aprendizaje de 0.01. Se tendrá esto en cuenta para plantear los siguientes

experimentos, ya que además se ve en la misma tabla que los mejores desempeños se lograron con filtros de 3x3.

Además de esto, al ver que se tienen 8 clases diferentes, se preferirá tener más mapas de características (que son las imágenes después de ser filtrdas), se propondrán arquitecturas que vayan aumentando su número de filtros a medida que crece su número de capas para poder tener un mejor discernimiento en la feedforward del final.

Como se mencionó en el inicio de la sección de clasificadores basados en red convolucional, los parámetros de entrada, padding y striding serán los mismos, teniendo esto en cuenta los experimentos que se tendrán en cuenta son los siguientes:

- Red neuronal convolucional con una capa de convolución con 8 filtros de 3x3 y una max pooling y tasas de aprendizaje de 0.01, 0.001, 0.0001.
- Red neuronal convolucional con dos capas de convolución con 8 filtros de 3x3, 16 filtros de 3x3, dos max pooling, las capas convolucional y max pooling serán intercaladas y tasas de aprendizaje de 0.01, 0.001, 0.0001.
- Red neuronal convolucional con dos capas de convolución con 16 filtros de 3x3, 8 filtros de 3x3, dos max pooling, las capas convolucional y max pooling serán intercaladas y tasas de aprendizaje de 0.01, 0.001, 0.0001.
- Red neuronal convolucional con tres capas de convolución con 8 filtros de 3x3, 16 filtros de 3x3, 32 filtros de 3x3, tres max pooling, las capas convolucional y max pooling serán intercaladas y tasas de aprendizaje de 0.01, 0.001, 0.0001.
- Red neuronal convolucional con tres capas de convolución con 32 filtros de 3x3, 16 filtros de 3x3, 8 filtros de 3x3, tres max pooling, las capas convolucional y max pooling serán intercaladas y tasas de aprendizaje de 0.01, 0.001, 0.0001.

Tabla 3.4: Resultados de las múltiples corridas con las arquitecturas propuestas

Arquitectura	# Filtros C1	Tamaño Filtro	# Filtros C2	Tamaño Filtro	#Filtros C3	Tamaño Filtro	lr	μ Coste	σ Coste	μ Exactitud	σ Exactitud
1	8	3x3	0	0x0	0	0x0	0.01	1.3176	0.16	54.15 %	3.21 %
1	8	3x3	0	0x0	0	0x0	0.001	1.2211	0.13	57.63 %	2.68 %
1	8	3x3	0	0x0	0	0x0	0.0001	1.4088	0.15	49.30 %	2.88 %
2	8	3x3	16	3x3	0	0x0	0.01	1.1220	0.09	61.08 %	1.69 %
2	8	3x3	16	3x3	0	0x0	0.001	1.0147	0.13	61.90 %	2.42 %
2	8	3x3	16	3x3	0	0x0	0.0001	1.4240	0.10	48.28 %	2.13 %
3	16	3x3	8	3x3	0	0x0	0.01	1.1040	0.16	62.00 %	2.87 %
3	16	3x3	8	3x3	0	0x0	0.001	1.1021	0.13	61.81 %	2.33 %
3	16	3x3	8	3x3	0	0x0	0.0001	1.4123	0.11	49.14 %	1.98 %
4	8	3x3	16	3x3	32	3x3	0.01	1.0143	0.16	64.39 %	1.75 %
4	8	3x3	16	3x3	32	3x3	0.001	1.0212	0.13	64.98 %	2.54 %
4	8	3x3	16	3x3	32	3x3	0.0001	1.4043	0.14	50.24 %	2.26 %
5	32	3x3	16	3x3	8	3x3	0.01	0.9957	0.20	65.82 %	3.64 %
5	32	3x3	16	3x3	8	3x3	0.001	1.0318	0.17	64.75 %	3.16 %
5	32	3x3	16	3x3	8	3x3	0.0001	1.4808	0.10	45.98 %	1.19 %

Con base en los resultados de la tabla 3.4, se puede decir que si se toma como referencia el valor del promedio de la función de coste en las cien corridas, la arquitectura ganadora es la número 5 con una tasa de aprendizaje de 0.01, además de esto, en las figuras 3.18 y 3.19 se puede ver el resultado de los experimentos.

A pesar de que se plantearon 100 épocas por cada corrida, con ayuda del Early Stopping de Keras fue posible detener las corridas antes de que empezara a tener demasiado overfitting y así evitar el gasto computacional de entrenamiento.

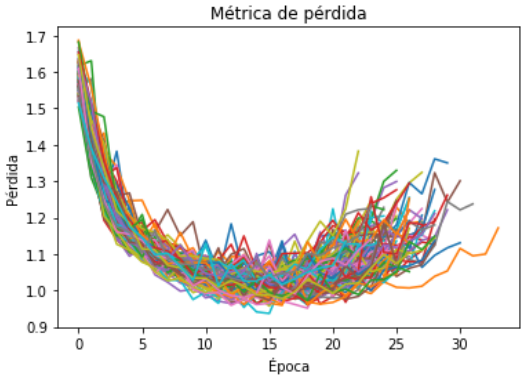


Figura 3.18: Función de coste de la arquitectura ganadora.

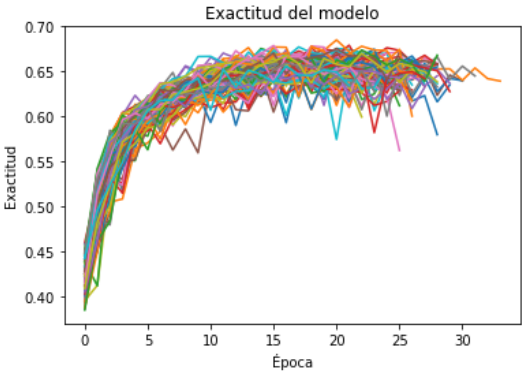


Figura 3.19: Exactitud de la arquitectura ganadora.

3.17. Algoritmo genético simple

En una sección anterior se trató el aprendizaje como un problema de optimización de tal manera que esto da paso a más heurísticas de minimización. Una de las más famosas en problemas de optimización son los algoritmos genéticos,estos están basados en la evolución bajo el paradigma de Charles Darwin [Darwin, 1936] donde sobrevive el individuo que mejor se adapte a un ambiente según sus características morfológicas. Son algoritmos de búsqueda global debido a que se evalúan varias soluciones al azar en la ejecución del algoritmo, esto quiere decir que un individuo puede empezar a buscar su evolución desde distintos puntos de inicio. Un algoritmo genético consta de varias partes, un genoma principalmente que será la descripción principal del problema de optimización, un método de selección de padres que se entrecruzarán para dar origen a un nuevo individuo, un método de cruce para este nuevo individuo y finalmente un método de mutación para que la diversidad poblacional se mantenta a lo largo de la ejecución del algoritmo.

3.17.1. Genoma

Para que un algoritmo genético pueda funcionar, se requiere de un genoma, el cual es una descripción de los parámetros que tiene en cuenta la función objetivo que quiere ser minimizada, en el caso de las redes neuronales serían sus pesos y desviaciones si se quiere optimizar su función de coste o una aplicación un poco más reciente de los algoritmos genéticos es la optimización de sus hiperparámetros (capas, cantidad de neuronas, entre otros) para lograr un mejor valor de función de pérdida en validación.

3.17.2. Métodos de selección de padres

En la literatura existen varios métodos de selección de padres para hacer crossover o entrecruce de los genomas de cada uno, entre las más populares existen la selección por ruleta, la selección por torneo, la selección por índice aleatorio hasta que se llene el número de hijos deseado, selección desde el fondo hasta el inicio [Haupt and Haupt, 1998]. Cada uno de estos métodos son más o menos efectivos dependiendo del tipo de problema y de la superficie de optimización que se esté trabajando.

3.17.3. Métodos de crossover

Así como existen distintos métodos para hacer la selección de padres, también existen varios métodos para hacer el crossover que consiste en mezclar el genoma de un individuo con el de otro individuo para generar dos individuos hijos que podrían ser o no parte de la siguiente generación dependiento de ciertos criterios [Haupt and Haupt, 1998]. Uno de los métodos de crossover más utilizados es el de fijar un punto aleatoriamente en todo el genoma y mezclar una parte con la otra de cada uno de los individuos.

Otro método bastante conocido de crossover es el de seleccionar posiciones aleatorias del genoma e intercambiarlas entre individuos, de tal manera que se hace un cruce de varias características de los mismos.

3.18. Clasificador evolucionado por medio de algoritmo genético simple

Con ayuda de la librería de Python pyGAD, la cual implementa algoritmos genéticos, se optimizará la función de pérdida de las redes convolucionales que se trabajaron anteriormente (ya que estas a diferencia de las feedforward, presentaron mejores resultados), los parámetros ahora serán los siguientes:

- Número de individuos en la población.
- Número de generaciones.
- Número de iteraciones del algoritmo.
- Método de crossover.
- Método de mutación.

Básicamente la estructura del algoritmo genético a partir de esta librería de python, siempre es la misma, por lo cual en la figura 3.20 se puede ver el flujograma de este ciclo, lo único diferente que se hará es utilizar los callbacks mencionados en el mismo para poder obtener las curvas de validación.

Esta librería tiene unos módulos especiales para trabajar con redes neuronales generadas a partir de Tensorflow Keras por lo que se tendrá compatibilidad entre

las versiones que se trabajan, de tal manera no habrá cambios en la implementación final.

En las tablas 3.3 y 3.4 se puede apreciar que para la misma arquitectura se entrenó con tres diferentes tasas de aprendizaje, lo cual arrojó tres resultados diferentes, sin embargo como en este tipo de optimización no se tiene en cuenta una tasa de aprendizaje, se tomarán los tres resultados diferentes como un único set de experimentos, es decir como se muestra en (3.5), el número total de soluciones exploradas por arquitectura es de:

$$N_{SE} = 3N_{EP}N_{IT} \tag{3.5}$$

Los términos de (3.5) son N_{SE} número de soluciones exploradas, N_{EP} número de épocas y N_{IT} número de iteraciones, para los casos de las redes convolucionales el número de soluciones exploradas por arquitectura es de $3x100x100$, es decir, 30000.

Para que el desempeño en optimización de los algoritmos genéticos sea comparable, se propone (3.6), así se hallará el número de generaciones o la variable que se quiera calcular de los parámetros del algoritmo.

$$Gen = \frac{N_{SE}}{N_{COR}N_{P_{IN}}} \tag{3.6}$$

Para tener en cuenta los términos de (3.6) son Gen número de generaciones, N_{COR} número de corridas del algoritmo genético, $N_{P_{IN}}$ número de individuos en la población inicial y N_{SE} número de soluciones exploradas. Finalmente, teniendo el número de soluciones exploradas calculado anteriormente y asumiendo una población inicial de 30 individuos con un número de corridas de 10, la cantidad de generaciones sería de 100, por lo cual cada arquitectura tendrá una exploración de la función de pérdida con estas características.

Para poder lograr una buena exploración, la experiencia indica que los siguientes parámetros deben ser fijados:

- Probabilidad de cruce: 30 %.
- Probabilidad de mutacion: 2 %.
- Método de selección de padres: ruleta.
- Padres conservados en cada generación: 10.
- Tipo de crossover: punto único.
- Tipo de mutación: reemplazo de valor aleatorio.

Con estos parámetros para el algoritmo genético, se hará la optimización para las mejores arquitecturas de las tablas 3.3 y 3.4, por lo tanto se explorará un espacio solución con esta búsqueda global.

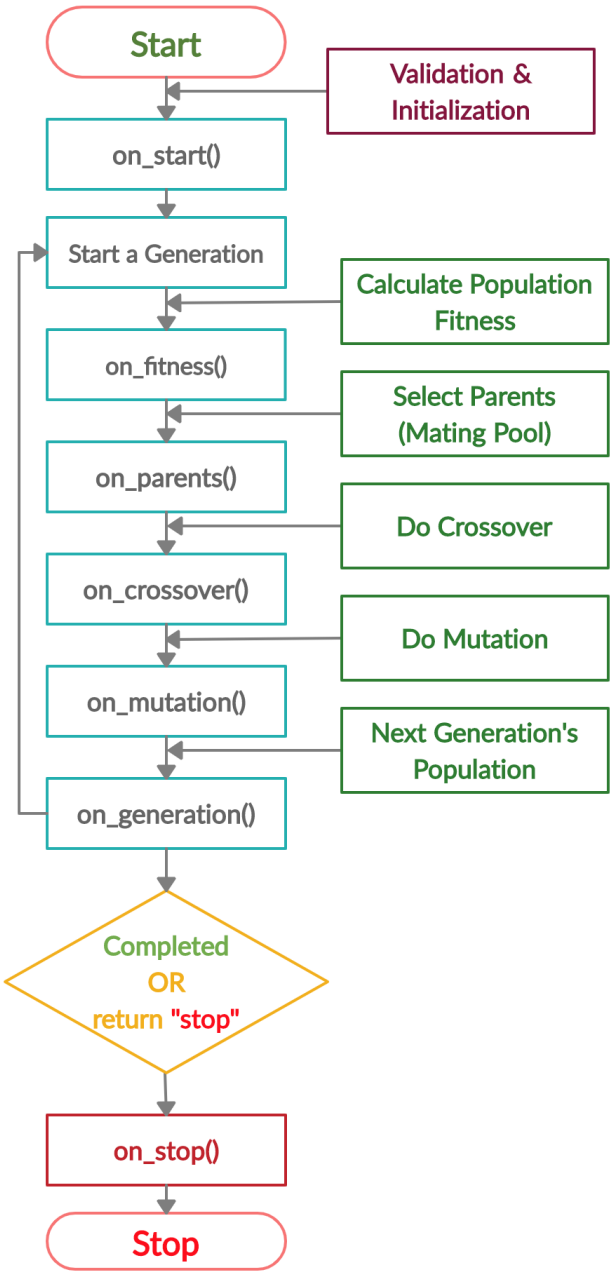


Figura 3.20: Ciclo de vida del algoritmo genético de pyGAD

En la figura 3.20 se puede ver el ciclo de vida de un algoritmo genético en pyGAD, se puede ver que tiene callbacks para cada una de las funciones que lleva el algoritmo genético, calcular la función de pérdida para cada individuo, obtener los padres para los nuevos individuos, la mutación y la siguiente generación.

La única diferencia es que el planteamiento de esta heurística no tiene enfoque de minimización sino de maximización, lo cual obligará a que la función de pérdida sea el recíproco de la función de pérdida original que para el caso de la clasificación binaria sería entropía binaria cruzada y para la clasificación de tipo sería entropía categórica cruzada como se mencionó anteriormente en la sección de los entrena-

mientos convolucionales.

Tabla 3.5: Resultados de optimización mediante algoritmo genético para Vehículo - No Vehículo

Función de pérdida ganadora	Exactitud ganadora	μ Coste	σ Coste	μ exactitud	σ exactitud
0.4651	73.25 %	0.7199	0.4564	58.92 %	12.3 %

En la tabla 3.5 se pueden observar los resultados de la optimización de la red mediante un algoritmo genético simple con las características mencionadas anteriormente, se puede observar que el desempeño mostrado por este optimizador es bueno, sin embargo, se obtuvieron mejores resultados utilizando backpropagation con descenso de gradiente para la misma arquitectura de red convolucional.

Tabla 3.6: Resultados de optimización mediante algoritmo genético para Tipo de Vehículo

Función de pérdida ganadora	Exactitud ganadora	μ Coste	σ Coste	μ Exactitud	σ Exactitud
1.91	42.25 %	1.96	0.0453	47.44 %	2.21 %

Corriendo el mismo algoritmo pero cambiando la base de datos y la aquitectura de la red, se obtienen los resultados de la tabla 3.6, en este caso la optimización de la mejor solución fue comparable con la de la misma arquitectura pero utilizando backpropagation y descenso de gradiente. En las figuras 3.21 y 3.22 se puede apreciar el resultado de la evolución de la red neuronal a medida que van pasando las generaciones.

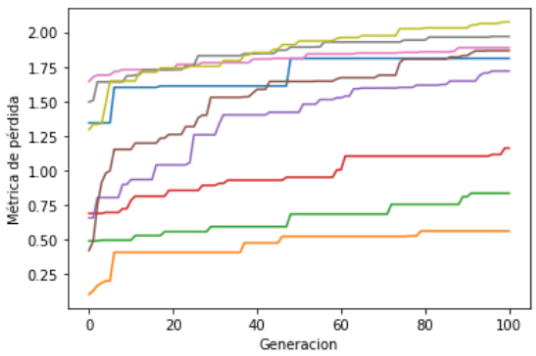


Figura 3.21: Evolución para la red convolucional de vehículo y no vehículo.

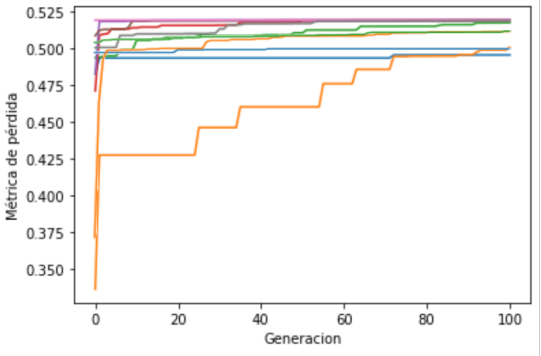


Figura 3.22: Evolución para la red convolucional de tipo de vehículo.

3.19. Transfer Learning

Varios problemas de reconocimiento, detección y segmentación de imágenes ya han sido tratados en anteriores ocasiones, por lo que ya hay un desarrollo bastante extenso en todo lo que tiene que ver con estos problemas, por ejemplo se tienen redes

convolucionales para clasificación de imágenes como LeNet (desarrollada originalmente por YanLee Cun) o redes convolucionales para la detección de imágenes como YOLO (desarrollada originalmente por Joseph Redmon y su equipo) y para terminar de ejemplificar está MaskRCNN (desarrollada por Kaimming He y su equipo).

Todas estas redes para clasificación, detección y segmentación de imágenes basan su funcionamiento respectivamente en redes convolucionales con capas feedforward al final, redes convolucionales puras y redes convolucionales como autoencoder.

Como la idea es dejar una semilla para desarrollar un sistema embebido potente capaz de realizar el conteo de tráfico en vídeo de manera correcta y confiable, se muestra una gráfica a continuación para hacer un análisis.

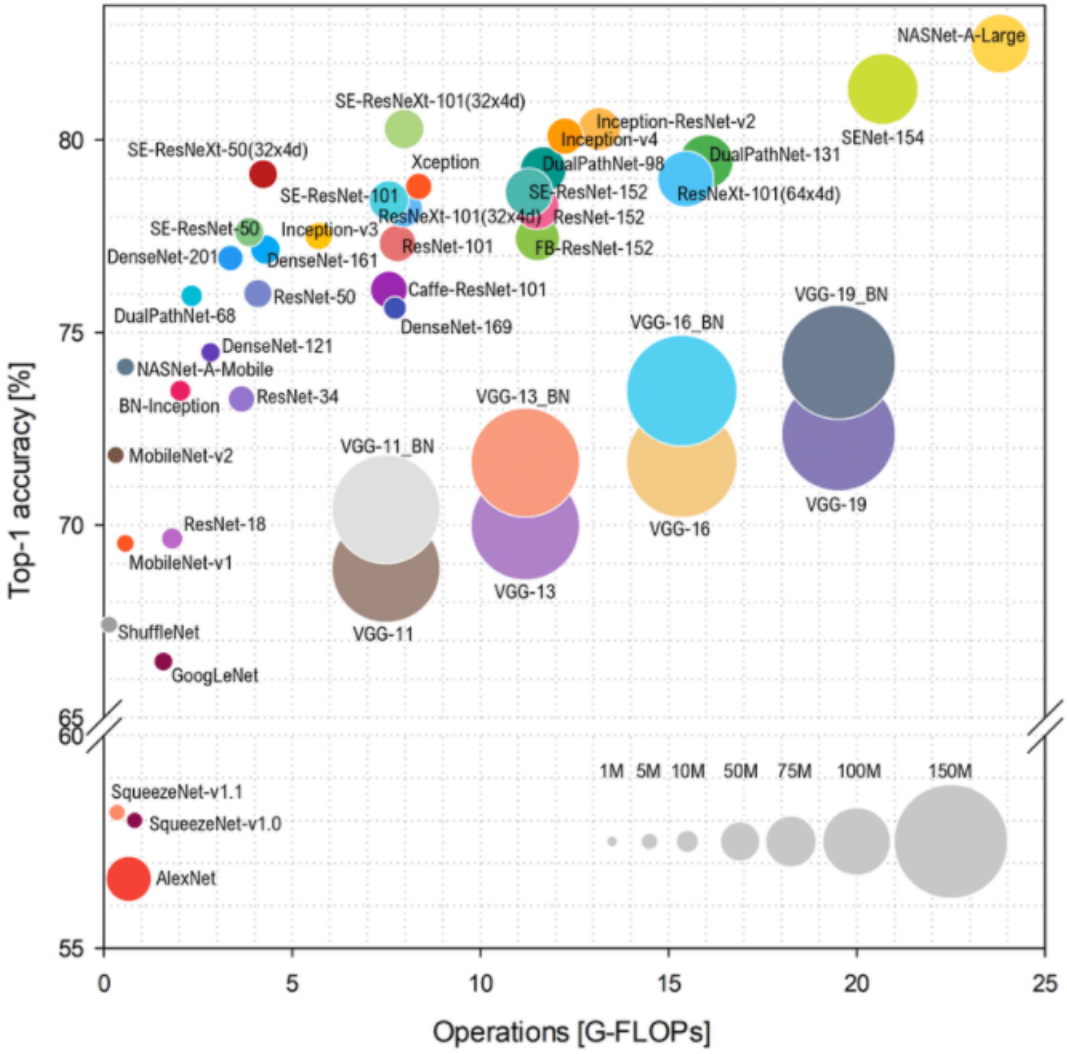


Figura 3.23: Redes neuronales convolucionales famosas vs operaciones de punto flotante necesarias tomada de [Bianco et al., 2018]

En la figura 3.23 se puede apreciar unos ejemplos de las redes neuronales convolucionales profundas más representativas del estado del arte en este momento, se tienen por ejemplo, la mencionada LeNet, VGG (en sus varias versiones), ResNET, entre otras.

Para hacer este análisis debemos tener en cuenta la tabla 3.7 que muestra unas opciones de sistema embebido junto con su capacidad de operaciones de punto flotante.

Tabla 3.7: Operaciones de punto flotante según dispositivo embebido

Dispositivo	GFLOPS
Raspberry PI 4B	9.69
nVidia Jetson Nano	472
Google Coral Dev Board	32

Teniendo en cuenta el hecho de que estas redes son lo suficientemente profundas para contar con cientos de millones de parámetros, la cantidad de operaciones de punto flotante necesarias para hacer un fordward sin optimizar es impresionante, si se mira por ejemplo el caso de la Raspberry PI, puede hacer $9{,}69x10^9$ operaciones de punto flotante **por segundo**, lo cual implica que para hacer un fordward en una red VGG-19 se demoraría al rededor de 2 segundos, esto no es de utilidad para el proyecto en cuestión debido a que se quiere hacer un conteo en tiempo real en un dispositivo embebido económico.

Haciendo el mismo ejercicio con una nVidia Jetson nano, se puede verificar que ahora se necesitaría menos de un décimo de segundo para poder hacer forward de una imagen en la misma red mencionada anteriormente por lo cual se podrían tener hasta 100 objetos en una imagen y tardaría un segundo en hacerlo, suena impresionante pero así mismo se eleva el costo de producción, ya que una Raspberry PI cuesta al rededor de 55 dolares americanos y una Jetson Nano cuesta al rededor de 100 dolares.

Por esta razón se hará uso de las redes que se encuentran a la izquierda del número 5 en el eje X de la figura 3.23, y como se tienen muchas implementaciones de código abierto de cada una, se elige la que está más a la izquierda con una exactitud elevada, en este caso la MobileNetV2.

3.19.1. MobileNet V2

Desarrollada por ingenieros de Google, MobileNetV2[Sandler et al., 2018] es una arquitectura del estado del arte para redes convolucionales que funcionan en dispositivos móviles, es una red totalmente convolucional que sirve como extractor de características o codificador de imágenes en un vector final de $1x1x1280$, éste podrá ser utilizado para diferentes aplicaciones como clasificación, detección o segmentación de la imagen.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figura 3.24: Arquitectura MobileNetV2 tomada de [Sandler et al., 2018]

En la figura 3.24 se puede encontrar las capas y el papel que juegan en la red para terminar con el vector de características final, en el artículo se puede encontrar a más detalle para qué son cada uno de los operadores que tiene en las otras columnas. La implementación opensource que uno puede encontrar en los ejemplos de TensorFlow fue entrenada con el dataset ImageNet de Google que contiene 1000 clases, entre ellas, vehículos y los que se necesitan.

3.19.2. MobileNetV2 Vehículo - No vehículo

Lo que se hará en este caso es tomar la implementación de OpenSource de Google y agregar una nueva capa totalmente conectada a la salida con una única neurona de salida, de tal manera que se podrá hacer el discernimiento entre si la región de interés es un vehículo o no.

Acá lo que se necesita básicamente es congelar las capas anteriores las cuales ya sirven como extractor de características para las 1000 clases de ImageNet y los únicos parámetros que se modificarán serán los pesos y desviaciones de las neuronas agregadas al modelo.

Al utilizar la base de datos que se ha manejado durante todos los experimentos y entrenamientos anteriores, con una tasa de 0.0001, se obtuvieron los resultados de las figuras 3.25 y 3.26, se puede apreciar que el modelo arranca de una con unos valores de pérdida bastante bajos y de exactitud bastante altos.

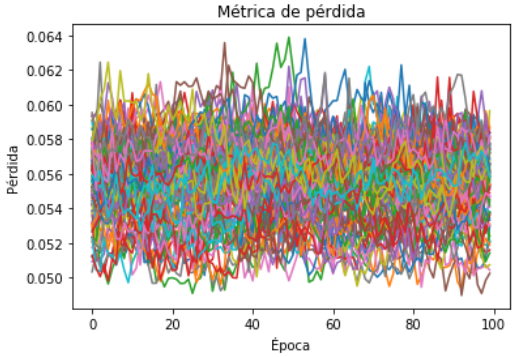


Figura 3.25: Función de coste de la arquitectura ganadora.

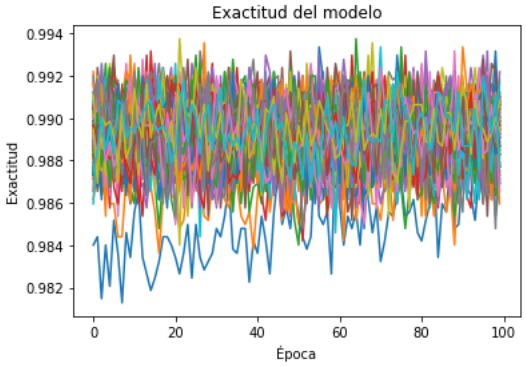


Figura 3.26: Exactitud de la arquitectura ganadora.

Las figuras que se acaban de mostrar corresponden a 100 corridas de 100 épocas, donde los pesos de las neuronas que se entrenaron fueron inicializados aleatoriamente, en la tabla 3.8 se puede encontrar el valor de la función de pérdida menor y el porcentaje de exactitud para la misma.

Para obtener del mejor modelo de cada curva se usó la función CheckPoint de Keras.

Tabla 3.8: Resultados de Transfer Learning para Vehículo - No Vehículo

Función de pérdida ganadora	Exactitud ganadora	μ Coste	σ Coste	μ Exactitud	σ Exactitud
0.049	97.95 %	0.052	0.00168	97.94 %	0.0042 %

3.19.3. MobileNetV2 Tipo de vehículo

Al igual que en el procedimiento anterior, lo que se hará es utilizar la misma implementación de Google para poder agregar una capa adicional totalmente conectada a la salida de la red con 8 neuronas ya que esa es la cantidad de tipos de vehículo que se tienen en el dataset. La capa de salida tendrá una función de activación softmax lo cual dará un valor de confiabilidad asociado a la clase a la cual pertenece.

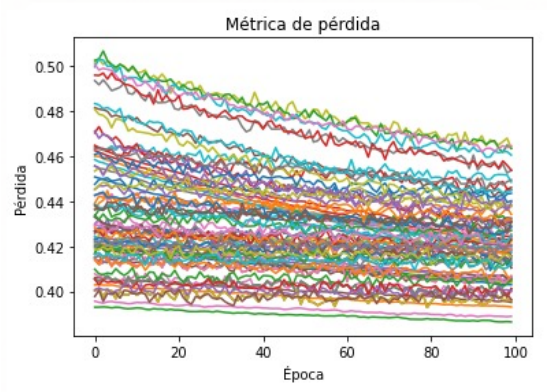


Figura 3.27: Función de coste de la arquitectura ganadora.

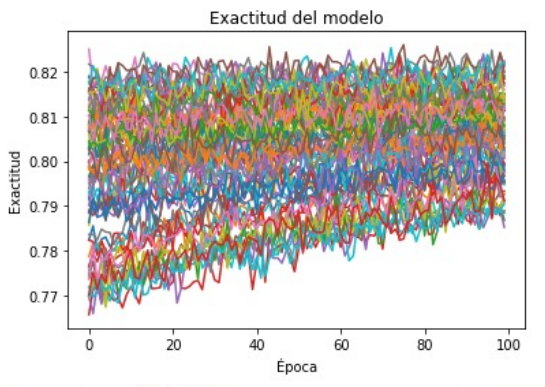


Figura 3.28: Exactitud de la arquitectura ganadora.

En las figuras 3.27 y 3.28 se puede ver los resultados de los diferentes entrenamientos que tuvo esta configuración con transfer learning, además de esto en la tabla 3.9 se puede ver los valores de pérdida y exactitud del mejor experimento junto con los promedios de los experimentos.

Tabla 3.9: Resultados de Transfer Learning para Tipo de Vehículo

Función de pérdida ganadora	Exactitud ganadora	μ Coste	σ Coste	μ Exactitud	σ Exactitud
0.3937	86.21 %	0.4295	0.0354	85.13 %	1.15 %

Capítulo 4

Creación del algoritmo y base de datos de actualización en tiempo real para los vehículos contados

Antes de hablar del algoritmo, se debe pensar en uno de los problemas más importantes que se tienen acá, el cual es extraer el elemento para pasarlo por los clasificadores.

4.1. Procesamiento de la imagen para obtener regiones de interés

En [Arinaldi et al., 2018], [S. Aqel and Aarab, 2017], [S. Kul and Sayar, 2017] y [Leelasantitham and Wongseree, 2008] se sugiere un método bastante interesante para hacer la obtención de las regiones de interés que van a ser clasificadas mediante el algoritmo de inteligencia computacional. Lo que se pretende en este caso es, con ayuda de la librería OpenCV de intel, hacer una serie de operaciones sobre la imagen de tal manera que se pueda obtener una máscara que se pueda aplicar para obtener contornos de objetos que no pertenezcan al fondo, para esto se hará uso de los algoritmos de extracción de fondo que vienen en la librería. OpenCV ofrece los siguientes algoritmos de extracción de fondo:

- KNN Background Substractor (Basado en K Nearest Neighbour).
- MOG2 Background Substractor (Basado en la mezcla de Gaussianas).
- CNT Background Substractor (Es una implementación rápida enfocada a embebidos de MOG2).
- GMG Background Substractor (Basado en series de tiempo y extracción de máscaras cuadro a cuadro).

Todos estos están debidamente referenciados en la documentación de OpenCV junto con los papers de los cuales salió cada uno de los algoritmos. En la librería se

encuentran varios métodos más de extracción de fondo pero los mencionados son los más utilizados en el estado del arte del trabajo actual.

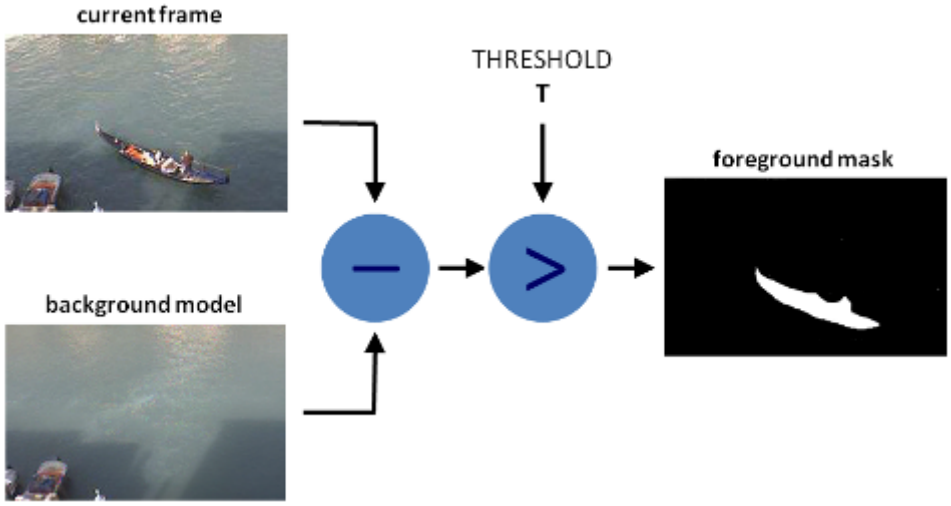


Figura 4.1: Método de extracción de fondo
[Opencv-documentation, 2021]

En la figura 4.1 se muestra uno de los métodos más simples para hacer la extracción del fondo la cual consiste en tener una imagen estática del fondo y restarla del cuadro actual, de esta manera se debería obtener los pixeles que no se encuentran en la imagen del fondo sólo con la resta y así se podría tomar esa porción de imagen para considerarla una región de interés que será clasificada, sin embargo, no siempre es posible obtener una imagen estática del fondo y más cuando se está pensando en un modelo que pueda implementarse en diferentes calles.

Cuando se presentan estos escenarios, una secuencia de imágenes puede resolver este problema. Los algoritmos mencionados anteriormente utilizan una secuencia de imágenes para hacer un modelo de cada pixel del fondo y poder decidir si es o no parte de este teniendo en cuenta métricas de tiempo. En el artículo [Marcomini and Cunha, 2018] se hace una comparación entre GMG, MOG y MOG2 donde el mejor resultado se obtiene con MOG2. Por lo tanto para extraer las regiones de interés del trabajo en cuestión se utilizará el método MOG2.

Después de aplicar el algoritmo de extracción de fondo se obtiene una máscara que se aplicará al foreground (así se llamará de ahora en adelante todo lo que no sea fondo) y de esta manera se pueden obtener los objetos que no son parte del fondo y de alguna manera se están moviendo sobre la escena. Básicamente después de esto sólo se tendrá que hacer un par de transformaciones morfológicas las cuales son erosión y dilatación para poder tener un parche que de un contorno parecido al que se necesita.

En la figura 4.2 se puede observar el algoritmo de extracción MOG2 aplicado a uno de los vídeos de prueba, como se puede observar la máscara tiene bastante ruido, por lo cual se puede hacer uso de un filtro pasabajas bidimensional.



Figura 4.2: Algoritmo MOG2 aplicado sobre un vídeo de prueba.

Luego de pasar este filtro, tal como se muestra en la figura 4.4, se erosionará la imagen (para evitar tener espacios en negro, es decir, que no hagan parte de la máscara total de cada vehículo) y se dilatará con el fin de que la región de interés no crezca en comparación con la verdadera, para esto se hará uso de las funciones implementadas en OpenCV.

Finalmente, luego de realizar este proceso se hallará cada uno de los rectángulos más pequeños que encierren cada contorno y se pasarán a través de los modelos de clasificación que se propondrán más adelante.

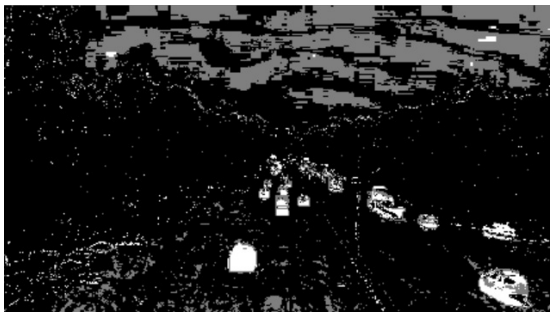


Figura 4.3: Máscara sin filtrar y sin tresholding.



Figura 4.4: Máscara con filtro aplicado

Para la validación de los modelos del capítulo anterior, lo que se realizará ahora es el conteo de tráfico en unos vídeos de prueba junto con su clasificación por tipo y color, para así poder obtener unas estadísticas que nos ayudarán a llevar cuenta de la cantidad de vehículos con cierta característica en cierta unidad de tiempo.

4.2. Desarrollo de la interfaz de usuario en Python

En la figura 4.5 se puede observar los pasos que se seguirán para poder enviar un vehículo clasificado a la base de datos con la que se trabajará.

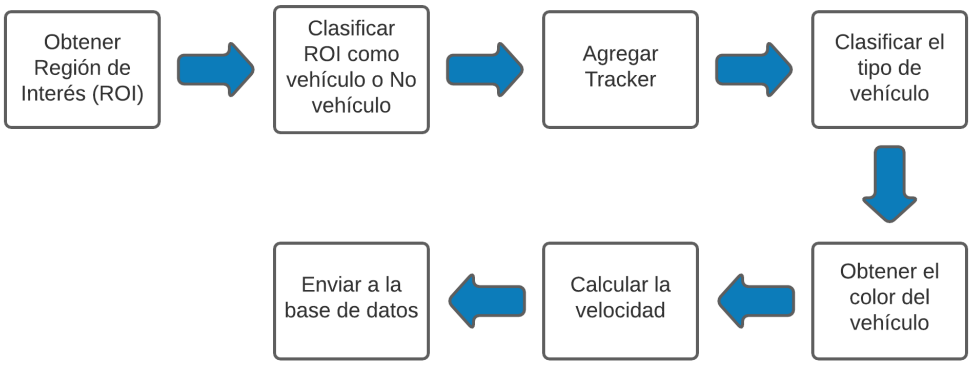


Figura 4.5: Diagrama de bloques del proceso de validación

A continuación, se explicará el proceso que se sigue en cada uno de los cuadros o procesos grandes de la validación del modelo para poder realizar un conteo del tráfico:

4.2.1. Obtención de región de interés ROI

Básicamente lo que se hace en esta etapa es hacer la extracción de fondo, filtrado, erosión y dilatación, para obtener las manchas móviles que posiblemente sean vehículos, esto con el fin de tener una imagen para alimentar las redes.

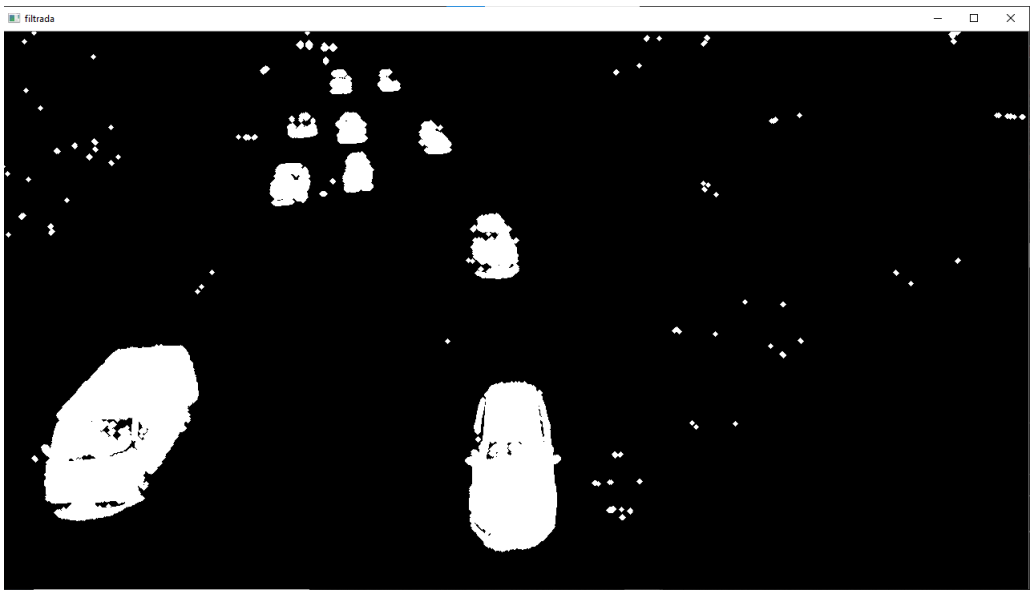


Figura 4.6: Obtención de las regiones de interés.

En la figura 4.6 se pueden apreciar los parches que serán pasados a la red, estos fueron generados con la extracción de fondo de la que se habló al inicio de la cuarta parte del desarrollo. Además de esto, se ve que hay un poco de ruido y contornos que pueden ser no deseados, el clasificador ayudará con esto para poder detectar los vehículos.

4.2.2. Clasificar ROI como vehículo o no vehículo

Una vez se encuentran las posibles regiones de interés en la imagen se pasarán por un clasificador que dirá si esta región de interés es o no un vehículo, basados en los resultados de las tablas 3.1, 3.3, 3.5 y 3.8, se puede ver que la que tiene una menor función de coste es la opción de transfer learning con 97 % de exactitud y una función de coste de 0.049 aproximadamente, sin embargo, con una arquitectura más simple, una función de coste de 0.1255 y una exactitud de aproximadamente 95.97 %, una solución más viable es la mejor de la tabla 3.3, con los datos anteriormente nombrados es la arquitectura número 4 con tasa de aprendizaje 0.01.

En la figura 4.7 se puede observar la matriz de confusión para la mejor arquitectura que se tuvo de las 100 corridas de la solución que se tomó para abordar esta parte del problema.

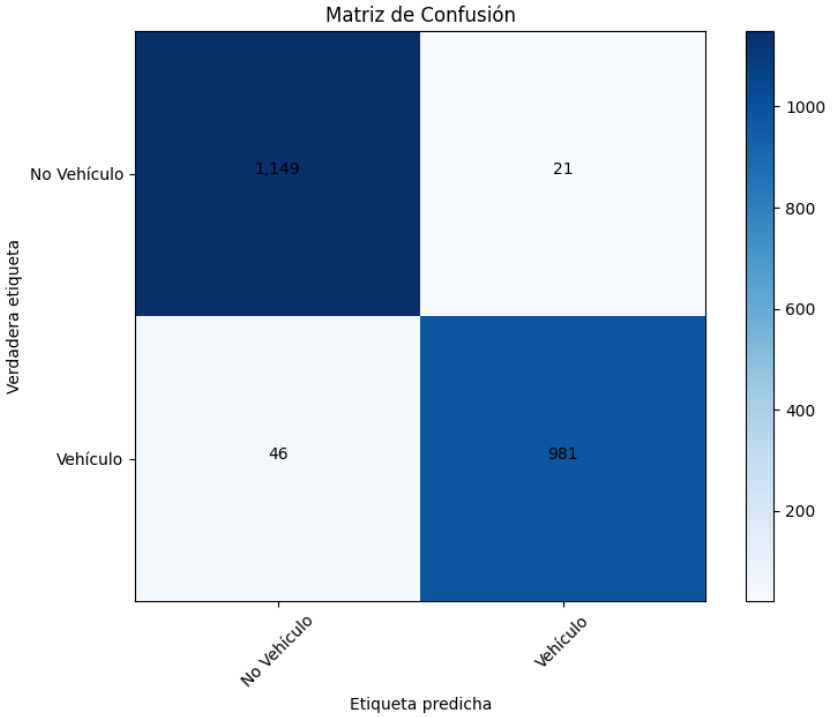


Figura 4.7: Matriz de confusión de validación para la mejor solución de clasificador binario.

Una vez implementado el clasificador en el algoritmo de conteo de vehículos, se obtiene la figura 4.8 en donde se puede ver que algunos vehículos ya tienen su correspondiente bounding box indicando que fueron reconocidos como vehículo por el clasificador.

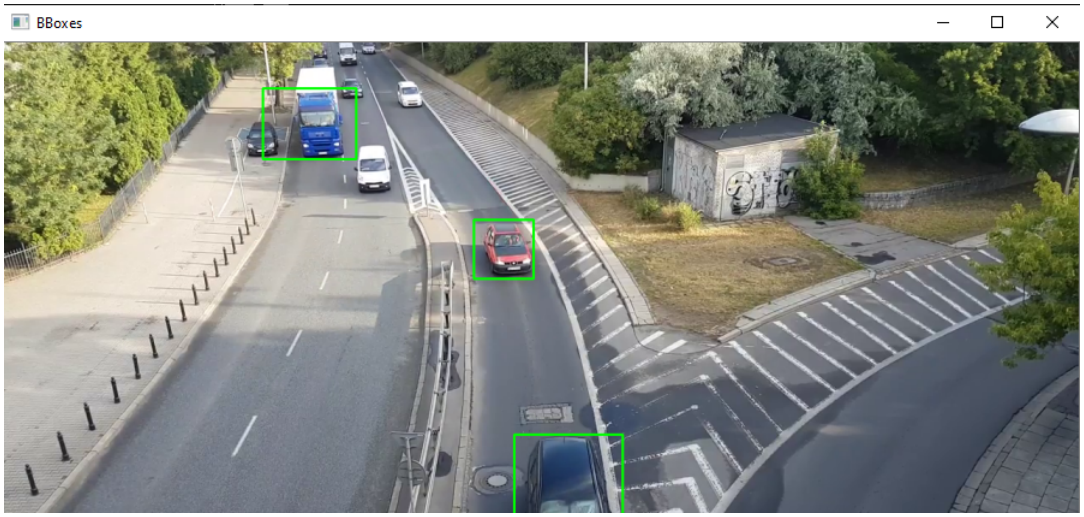


Figura 4.8: Visualización de los vehículos detectados.

Una vez se haya detectado el vehículo, se creará una clase en Python que será la encargada de guardar la información del mismo, en la figura 4.9 se puede encontrar el diagrama de UML que la describirá.

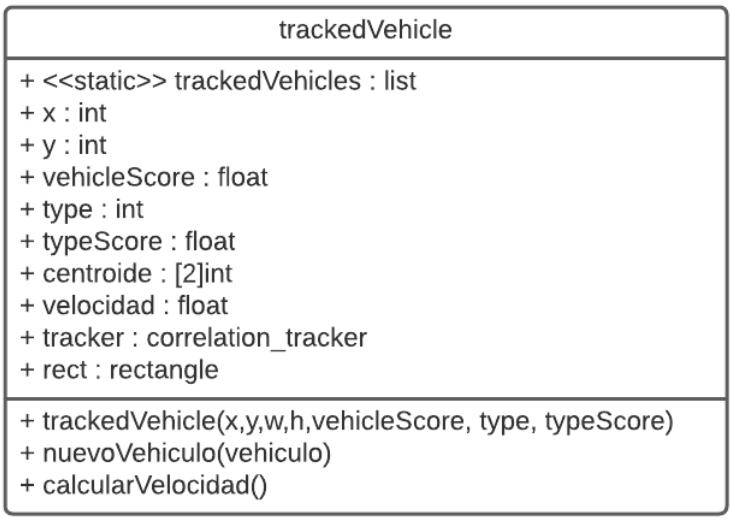


Figura 4.9: Diagrama UML de la clase trackedVehicle.

Los atributos que se pueden ver en este, son las coordenadas (x, y) de la esquina superior izquierda de la caja, el valor de confiabilidad con el que fue clasificado como vehículo, el tipo de vehículo, el valor de confiabilidad con el que fue clasificado bajo ese tipo, un arreglo de enteros de dos posiciones que guardará el centroide de la caja del vehículo, la velocidad cuando esta se calcule, el tracker de correlación asignado al vehículo y finalmente un rectángulo de la clase rectángulo de dlib. Además de esto en sus métodos se puede apreciar el constructor, un método para agregar un nuevo vehículo a la lista de vehículos con tracker y finalmente el método para calcular la velocidad del vehículo, este se explicará más adelante.

4.2.3. Agregar Tracker

Con ayuda de la librearía **dlib** que trae implementaciones para C++ y Python, se logra implementar un tracker de objetos basado en filtros de correlación discriminativos autoaprendidos, en [Danelljan et al., 2014] se puede hallar una explicación más a fondo de toda la matemática y el enfoque detrás de este tipo de tracker. Se eligió este sobre el tracker de centroide debido a que el tracker de centroide necesita que el algoritmo de detección se ejecute más seguido, lo cual requiere más recursos computacionales y es más posible que se pierda el vehículo una vez ya fue detectado.

4.2.4. Clasificar el tipo de vehículo

De la misma manera en la que se eligió un clasificador para identificar si la región de interés era un vehículo o era un no vehículo, en las tablas 3.2, 3.4, 3.6, 3.9, se puede apreciar que la mejor solución en este caso por una gran diferencia es la que se logró mediante los entrenamientos de transfer learning. Además de esto, en la figura 4.10 se puede ver la matriz de confusión generada por esta arquitectura.

Sin embargo se puede observar que se presenta un sesgo importante hacia la clase Carro, esto se puede deber a que la diferenciación de las características que se tiene que hacer de imagen a imagen para poder lograr una clasificación, no es lo suficientemente buena.

Considerando que debe ser algo portable, acá se sacrifica un poco la exactitud con tal de tener una solución que se pueda poner en un sistema embebido sin GPU.

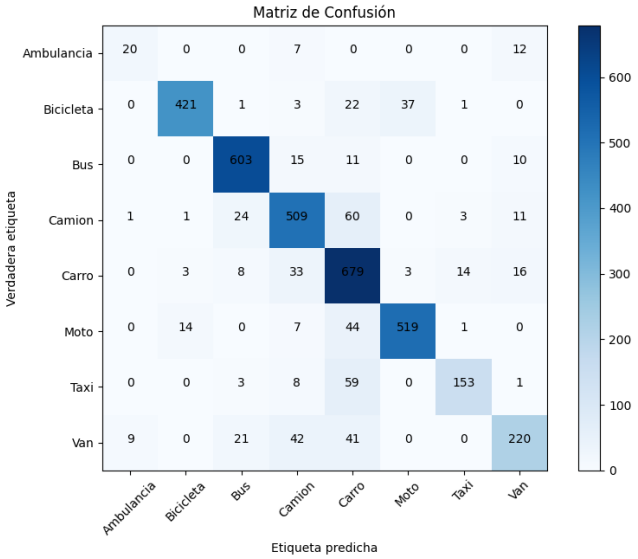


Figura 4.10: Matriz de confusión de validación generada por el clasificador de Tipo de Vehículo con el dataset de validación.

En la figura 4.10, se puede observar que hay un poco de sesgo hacia la clase de carro, esto se puede explicar si se tiene en cuenta que las características de los filtros que se extraen para diferenciar de una clase en otra clase, son bastante similares, sin embargo, en la práctica sobre el vídeo, en la figura 4.11 se logra ver que no es como que una van salga clasificada como una bicicleta ni nada por el estilo.

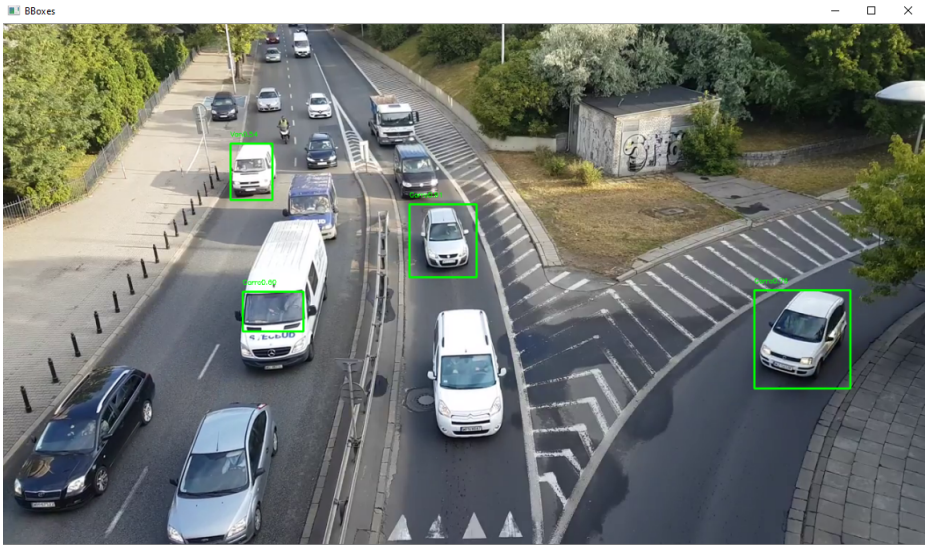


Figura 4.11: Implementación del clasificador en el algoritmo de conteo.

Con esto la etapa de clasificación de la región de interés en el tipo de vehículo queda concluída.

4.2.5. Estimación del color del vehículo

Para la obtención del color del vehículo la idea es bastante sencilla, se parte de 6 colores base junto con sus valores de RGB, como se muestra en la tabla 4.1.

Tabla 4.1: Etiqueta de color junto con su valor RGB

Color	R	G	B
Rojo	255	0	0
Verde	0	255	0
Azul	0	0	255
Negro	0	0	0
Blanco	255	255	255
Amarillo	236	208	0

Con ayuda de OpenCV, se obtiene el promedio de color de la imagen bajo consideración (Región de interés ROI) y con esto se hace una distancia euclidiana entre las coordenadas del color mostradas en la tabla anterior y el promedio de color actual de la ROI, así se asignará una etiqueta de color al vehículo.

4.2.6. Estimación de la la velocidad

Teniendo en cuenta que no se tiene una referencia fija con qué comparar la velocidad, esta será únicamente una medida adicional que podrá ser descartada o usada dependiendo de las necesidades del modelo de tráfico que se menciona al inicio de este proyecto, la manera de calcular la distancia será utilizando (4.1), de tal manera que se tomará un comportamiento de velocidad uniforme.

$$V = \frac{Distancia}{\Delta t} \tag{4.1}$$

El tiempo será tomado entre los dos puntos de interés con ayuda de la librería Time de Python, los puntos de interés pueden ser configurados a través de la interfaz gráfica que se diseñe en Python, sin embargo, como este dato puede o no ser tomado, sólo se tendrá visualización de él en la salida final del algoritmo mas no en la interfaz gráfica.

En la figura 4.12 se puede observar dos nuevas herramientas que tendrán tanto para conteo como para medición de la velocidad.

Básicamente apenas pase por la línea de velocidad, se activará una bandera que indicó que el vehículo ya pasó por ahí y se guardará el tiempo actual apenas cruce la línea azul. Después, cuando llegue a la región coloreada de rojo, se capturará de nuevo el tiempo y con una distancia predefinida (que puede variar debido a la colocación de la cámara) se realizará el cálculo mencionado anteriormente, la interfaz pedirá la distancia mediante su script en metros y el tiempo estará calculado en segundos, por lo cual para pasarla a kilómetros por hora se utilizará (4.2) y esa será la medida a guardar.

$$\frac{V}{\frac{km}{h}} = \frac{\frac{3600}{s}}{\frac{1}{h}} \frac{\frac{1}{km}}{\frac{1000}{m}} \frac{\frac{Distancia}{m}}{\frac{\Delta t}{s}} = \frac{36}{10} \frac{Distancia}{\Delta t} \tag{4.2}$$

Después de hacer la conversión de unidades se obtiene la velocidad final en kilómetros por hora, la cual será almacenada junto con los demás datos.

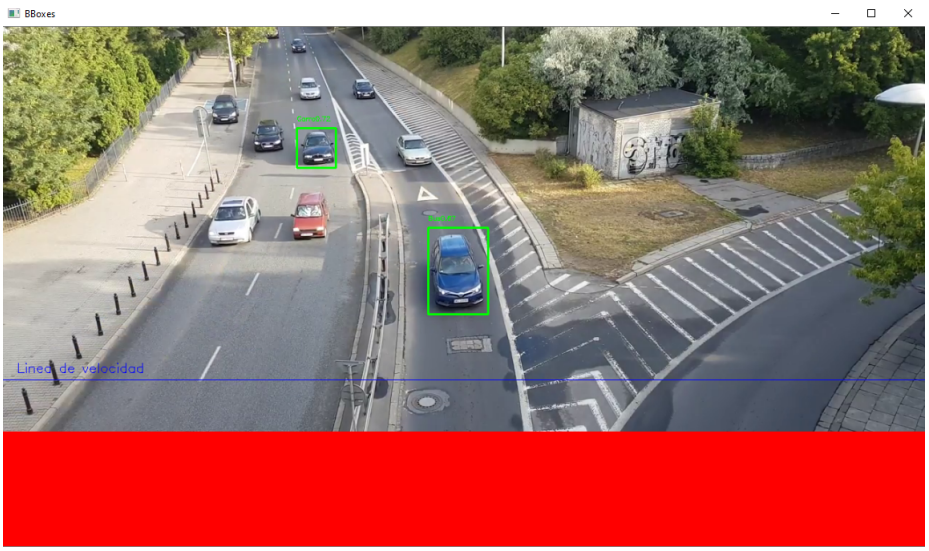


Figura 4.12: GUI con la línea de velocidad y espacio para variables.

4.2.7. Enviar a la base de datos

Una vez se hayan extraído todas las características que se nombraron anteriormente del vehículo, se procederá a generar un diccionario en Python con las siguientes características:

```
{
    "score" : Score con el cual fue clasificado,
    "tipo" : Etiqueta de tipo que se dio al vehículo,
    "color" : Etiqueta de color que se dio al vehículo,
    "velocidad" : Valor calculado de la velocidad,
    "hora" : Hora en lacual fue captado el vehículo,
    "fecha" : Fecha en la cual se captó el vehículo
}
```

Este diccionario se convertirá en un archivo de formato JSON (JavaScript Object Notation), estos archivos son de bajo peso y pueden ser almacenados por bases de datos tipo NoSQL (No relacionales), esto hace que su modificación sea relativamente sencilla en caso de que se quiera agregar una nueva característica al proyecto en versiones posteriores.

La base de datos que se eligió es cross-platform, lo cual indica que puede correr también en plataformas Linux sobre las cuales se encuentran los dispositivos tales como Raspberry, Jetson-Nano, Google Coral Dev Board, de tal manera que el script pueda correr tanto en un PC como en estos dispositivos.

La idea es que el funcionamiento no requiera tanta capacidad computacional en caso de que se quieran utilizar computadores cuyo costo sea bastante bajo (pensando en un lugar tipo oficina para colocar el dispositivo) o en un sistema embebido pensando en un semáforo.

Bajo todas estas especificaciones, MongoDB fue la mejor alternativa, ya que tiene su implementación en todas las plataformas más populares de sistema operativo, en la figura 4.13 se puede ver la conexión a través de Robo 3T que es una interfaz gráfica para Mongo.

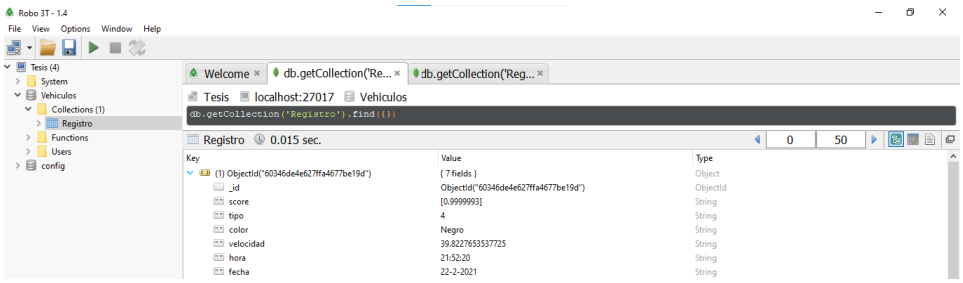


Figura 4.13: Ejemplo de captura de información en MongoDB usando Robo 3T.

Con esa base de datos resultante del algoritmo, se obtendrían las características para el modelo que se quiere realizar.

4.3. Pruebas en vídeo

Se utilizarán 3 vídeos para hacer pruebas del algoritmo que se diseñó anteriormente, para lo cual se contaron manualmente los vehículos y se clasificaron en su tipo, los

porcentajes de error se calcularán apartir de estos dos valores.

Tabla 4.2: Tabla de conteo y porcentajes de error en vídeo de prueba 1

Característica	Real	Algoritmo	Error
Conteo de vehículos	381	354	7.08 %
Conteo de Ambulancias	0	0	0 0 %
Conteo de Bicicletas	0	1	100 %
Conteo de Buses	8	50	525 %
Conteo de Camiones	20	10	50 %
Conteo de Carros	317	286	14.84 %
Conteo de Motos	5	0	100 %
Conteo de Taxis	0	0	0
Conteo de Vans	16	8	50 %

Tabla 4.3: Tabla de conteo y porcentajes de error en vídeo de prueba 2

Característica	Real	Algoritmo	Error
Conteo de vehículos	2551	2282	10.54 %
Conteo de Ambulancias	0	0	0 %
Conteo de Bicicletas	0	0	0 %
Conteo de Buses	22	62	181 %
Conteo de Camiones	96	70	27.08 %
Conteo de Carros	2123	1980	6.73 %
Conteo de Motos	6	2	66 %
Conteo de Taxis	0	0	0 %
Conteo de Vans	304	170	44.07 %

Tabla 4.4: Tabla de conteo y porcentajes de error en vídeo de prueba 1

Característica	Real	Algoritmo	Error
Conteo de vehículos	9	7	22 %
Conteo de Ambulancias	0	0	0 %
Conteo de Bicicletas	0	0	0 %
Conteo de Buses	0	0	0 %
Conteo de Camiones	0	0	0 %
Conteo de Carros	9	7	22 %
Conteo de Motos	0	0	0 %
Conteo de Taxis	0	0	0 %
Conteo de Vans	0	0	0 %

Los porcentajes de error fueron calculados respecto a los valores reales utilizando (4.3) para obtener ese valor, esta ecuación se utilizará siempre y cuando la cuenta real sea mayor a cero, de lo contrario, si no hay vehículos de ese tipo y el algoritmo cuenta, el error será del 100 %, y si no hay vehículos y el algoritmo no los cuenta, el error será del 0 %.

$$\%Error = |100\,\% - \frac{Conteo_{algoritmo}}{Conteo_{real}}\,\%| \tag{4.3}$$

En la figuras 4.14, 4.15 y 4.16 se presentan capturas de los tres videos utilizados para las pruebas mencionadas anteriormetne, como se puede observar cada cámara tiene un ángulo de visión diferente respecto a los vehículos de interés y además las condiciones de luz en el ambiente son diversas, esto permite que se tenga una simulación de posibles entornos en los que se podría poner un dispositivo completo (hardware +software) en una vía cualquiera.

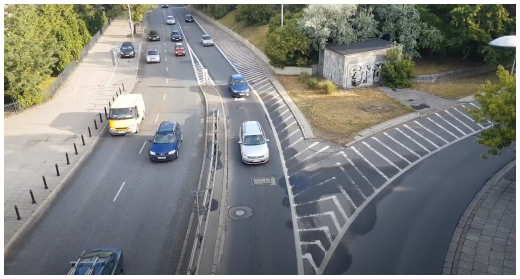


Figura 4.14: Captura video prueba 1.



Figura 4.15: Captura video prueba 2.



Figura 4.16: Captura video prueba 3.

Capítulo 5

Interfaz para el dispositivo

La mayoría de dispositivos embebidos que son capaces de correr modelos de inteligencia computacional en tiempos pequeños, tienen distribuciones de linux y tarjetas de red asociadas al hardware con el que vienen, en este caso, la Google Coral Dev Board viene con una distribución de Linux llamada Mendel y con un adaptador de WiFi que admite los protocolos 802.11n y 802.11ac, es decir, trabaja en las bandas de 2.4GHz y 5.1GHz. Gracias a la interfaz de red que tiene, se pueden montar servidores web en puertos TCP para que reciban peticiones HTTP o HTTPs, en este caso, se trabajó la creación de un servidor web mediante el lenguaje de programación JavaScript y el entorno de ejecución NodeJS (back-end), la parte del cliente (front-end) se trabajará mediante jQuery que es una API para diseño de animaciones para web a través de JavaScript. Para embebeer la aplicación de manera web en el dispositivo, se utilizará Flask debido a que es más liviano que Django y sólo se necesitó dar respuesta a un par de peticiones GET para activar o desactivar las predicciones y cambiar la imagen del stream.

Finalmente, para hacer el despliegue de la aplicación de manera sencilla, se trabajó con una red de contenedores de Docker a partir de Docker Compose para que en la misma máquina se pueda tener los servicios corriendo en puertos diferentes y tener que obviar la configuración de las librerías pues Docker se encargó de estas a partir de su archivo Dockerfile. De la comunicación entre el cliente real y el entorno virtual se encarga el archivo YAML que creará los bindings necesarios para poder acceder a la cámara.

Como recomendación final, el software embebido debe correr bajo una plataforma de Linux si es que no se quiere correr en un dispositivo que sea un microcomputador, el software quedará listo para que se conecte una cámara y se ejecute la instrucción que sube el sistema.

5.1. NodeJS

NodeJs es una plataforma desarrollada para la creación y ejecución de tareas de tipo asincrónico [OpenJS, 2010], perfectas para servidores tipo web HTTP o HTTPS. A medida de que ha avanzado la tecnología de capa de aplicación del modelo OSI o desarrollo web, han surgido distintos frameworks y plataformas para hacer uso de la arquitectura REST [Fielding and Taylor, 2000] la cual estipula unas palabras

definidas para hacer cierta petición al servidor, por ejemplo, si se quiere obtener información del servidor se hace la petición GET, si se hace una petición de modificación se utiliza la petición PUT, si se quiere eliminar un registro o información se usa la petición DELETE y si se quiere agregar nueva información o un nuevo registro se utiliza la petición POST. Después de realizar cualquiera de las peticiones anteriormente mencionadas, el servidor las gestiona mediante sesiones de usuario o mediante tokens que son generados entre back-end y front-end para poder darle camino a la información solicitada o generar un código HTTP de acuerdo a la autorización de acceso que se tenga.

5.2. jQuery

jQuery [Foundation, 2006] se define como una application programming interface (API de ahora en adelante) que simplifica la programación de los controles del lado del cliente haciendo que la interacción del programador con el document object model (DOM de ahora en adelante) sea más sencilla y sin necesidad de identificadores engorrosos o búsquedas difíciles en objetos mismos del DOM. Además de todo esto, simplifica el manejo de eventos de parte del usuario y el envío de información mediante el paradigma Asynchronous JavaScript and XML (AJAX de ahora en adelante) y el manejo de API para comunicación con otros servidores o clientes web.

5.3. Flask

Al igual que NodeJS, Flask [Ronacher, 2006] es un framework de programación de servidores web, una de las mayores diferencias es que Flask corre sobre Python haciéndolo compatible con la aplicación que se desarrolló para las redes neuronales que se trabajaron en secciones anteriores.

5.4. Docker

A medida que avanza el camino del desarrollo del software, una aplicación, se llega a la etapa de implementación y despliegue donde aparecen las operaciones conocidas como DevOps que se encargan de entregar las aplicaciones de una manera más organizada y sofisticada. Entre estas nuevas herramientas se encuentra Docker [Johnston and team, 2013], es un motor de contenedores multiplataforma desarrollado por Solomon Hymes. Un contenedor es un espacio virtual en el computador donde no se requiere un sistema operativo adicional para correr (como se hacía antiguamente cuando se quería tener servicios diferentes de Linux que no corrían sobre Windows por ejemplo) sino que se hace uso de las nuevas características de virtualización del hardware de los procesadores, haciendo así las aplicaciones menos dependientes del entorno, en otras palabras, escriba una vez, corra donde quiera.

5.5. Pantalla de inicio

En la figura 5.1 se puede observar la primera pantalla que se ve apenas uno accede a la interfaz de la aplicación web ubicada en el puerto 3000, funciona bajo el protocolo HTTP. Para hacer el ingreso al aplicativo, se quemó el código 12345 pero esto puede ser modificado en versiones posteriores para presentar una mayor seguridad en caso de querer ahondar en la interfaz web de este dispositivo.

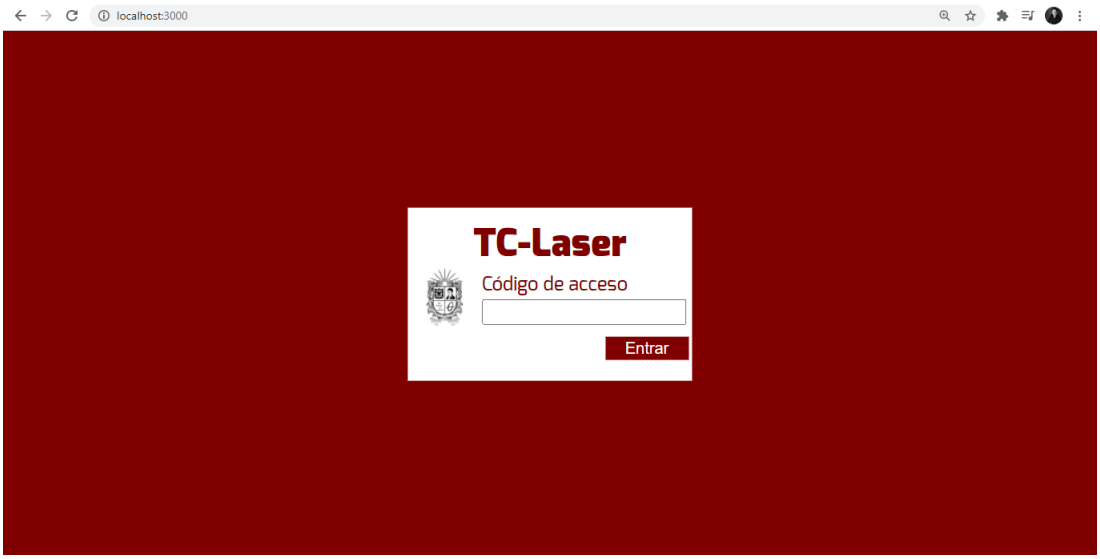


Figura 5.1: Pantalla de entrada de la interfaz web.

Esta pantalla se creó mediante HTML y CSS3, además de utilizar jQuery para la interacción con el cliente y los eventos como hacer click o utilizar la tecla de enter para iniciar sesión. Si se cumple con las reglas del aplicativo, la sección siguiente toma sentido, de lo contrario, se muestra un aviso de código incorrecto.

5.5.1. Validación de código y autorización de acceso

Para el inicio de sesión se hace una petición GET al endpoint 'entrar' de tal manera que se devuelva en un JSON el código de acceso a la aplicación para en el front-end, es decir en el lado del cliente, se haga la validación de si el código corresponde con el que se introdujo en el campo de texto. De coincidir el código se hace otra petición GET al endpoint 'iniciarSesion' donde se crea una variable cookie de sesión que permitirá contorlar que para ingresar al a plataforma haya tenido que ingresarse el código, de lo contrario devolverá un código de status HTTP 401 o Unauthorized. Finalmente lo que se hace si el sistema autoriza el acceso es hacer una redirección hacia el endpoint Index donde se hará una petición GET para obtener la página principal del software.

JQuery se presta para hacer peticiones AJAX de manera bastante sencilla, ya que tiene métodos que permiten realizar las peticiones del lado del cliente necearias para obtener una respuesta del lado del servidor, en este caso, el aplicativo desarrollado en NodeJS que se encarga de toda la lógica de back-end.

```
$.getJSON("http://localhost:3000/entrar", function(data){
    if(text === data.codigo){
        $.getJSON("http://localhost:3000/iniciarSesion", function(data)
        {
            if(data.allowAccess){
            location.href="http://localhost:3000/Index";
            }
        });
    }
    else{
        alert('Código incorrecto');
    }
});
```

Después de realizar los controles necesarios para evitar que el campo esté vacío y poder entrar con la tecla enter, se ejecuta el código anterior para hacer las peticiones y ver que se cumpla con las reglas de autorización del aplicativo.

5.5.2. Cookie de sesión

Para manejar la sesión se utiliza un módulo de NodeJS llamado express-session que permite manejar cookies de sesión en una aplicación que se desarrolle en el módulo express. Al ser una variable que puede accesarse desde cualquier parte del aplicativo sin importar el endpoint que se llame o la parte que se esté ejecutando, se declara como una variable de scope global que será gestionada por el módulo que se mencionó al principio, lo primero que se hará al cargar el aplicativo es poner esta variable de autorización de sesión en falso para que no hayan intrusos en partes posteriores de la misma, en el código a continuacón se puede ver como se hace esto.

```
    router.get('/', (req,res) =>{
    archivo = "/src/public/pages/index.html";
    res.sendFile(archivo, {root: path.join(__dirname,"..","..") });
    sess = req.session;
    sess.active=false;
});
```

5.6. Sección principal

Una vez se hayan hecho las comprobaciones de usuario y se haya creado la cookie de sesión, el usuario estará autorizado para ingresar a la página principal de la plataforma donde se encontrará con un menú con varios botones a la izquierda y en el centro la vista principal de streaming de la cámara.

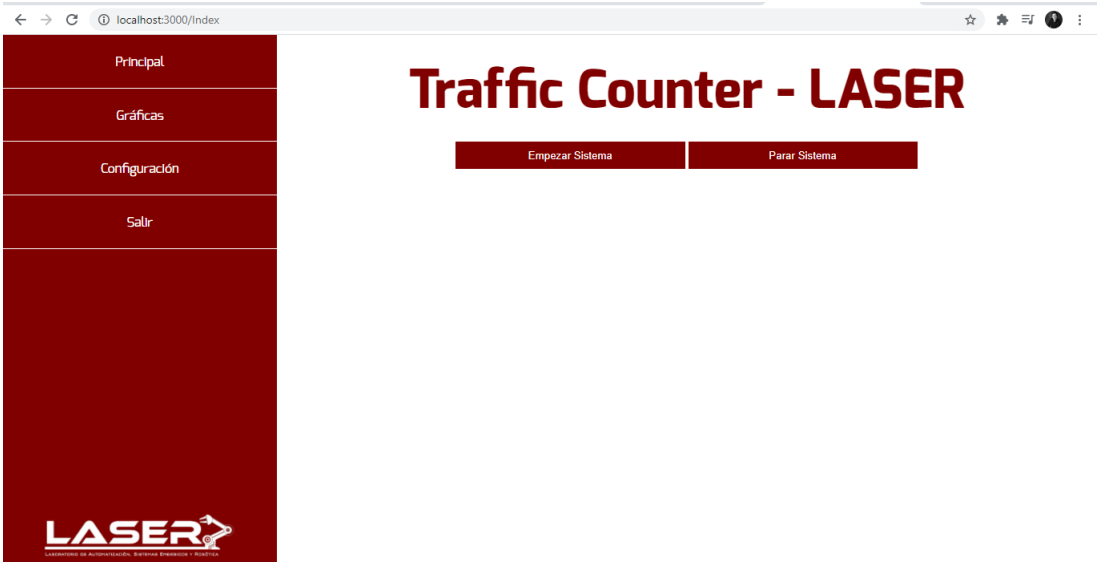


Figura 5.2: Pantalla principal del aplicativo web del algoritmo de conteo de tráfico.

En la figura 5.2, del lado izquierdo se tienen los botones que permiten navegar por el aplicativo, dirigirse a la página principal, a la sección de gráficos en donde se pueden ver gráficos del conteo de vehículos bajo ciertas características y ciertas fechas que serán elegidas por el usuario de las opciones que se le presenten, además de esto, para versiones posteriores se deja la pestaña de configuración cuya idea es que se envíe la información de límites de algoritmo, porcentaje de confiabilidad de clasificación, distancia de cálculo de velocidad, entre otras cosas, mediante la interfaz web al algoritmo que corre nativo en Python.

5.6.1. Comunicación entre la app principal en Flask y la plataforma en NodeJS

El aplicativo web se está ejecutando en el puerto 3000 mientras que el aplicativo de flask se está ejecutando en el puerto 3500, gracias a esto no tendremos problemas de CORS (Cross origin resource sharing o intercambio de recusos de origen cruzado) ya que parten de dos dominios diferentes, en este caso dos puertos diferentes. De parte del cliente, es decir jQuery, se van a lanzar dos peticiones GET de acuerdo a los botones que se ven en la sección principal de la figura 5.2, una se va a utilizar para iniciar la predicción sobre el stream de la cámara y la otra se va a utilizar para detener este mismo stream y reiniciar la cuenta sin alterar los datos de la base de datos de MongoDB. Esto se hace con el código que se muestra a continuación, la aplicación de Python no manda ninguna respuesta en este caso sino que únicamente se recarga la página para que se empiece a ver la predicción.

En la figura 5.3 se puede ver que toma el stream de la cámara en una plataforma de Linux, más adelante se explicará porqué, en este caso ubuntu, como implemento principal para realizar las predicciones. La idea principal es que esto corra sobre una cámara externa conectada a un dispositivo capaz de ejecutar las predicciones.

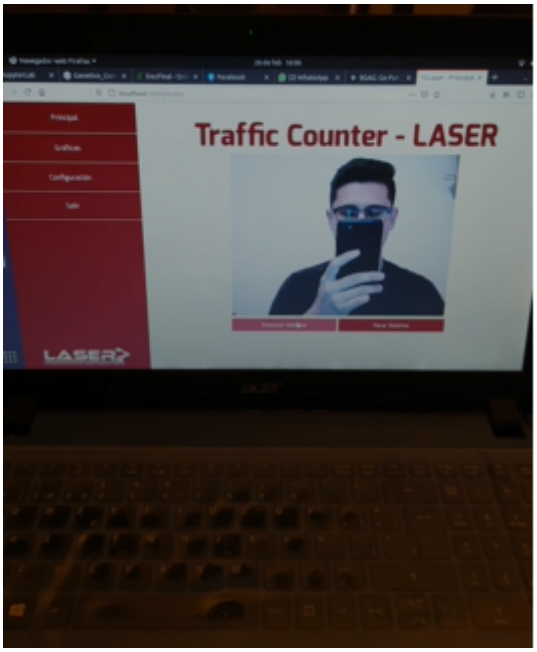


Figura 5.3: Demostración del stream de la cámara en un computador portatil.

De jQuery, el código a continuación es el que se ejecuta para hacer las peticiones hacia el aplicativo que corre las predicciones y la detección de vehículos.

```
$("#btnEmpezar").click(function(){
    $.get("http://localhost:3500/start",function(){
        console.log("Boton start clickeado");
        location.reload();
    })
});
$("#btnParar").click(function(){
    $.get("http://localhost:3500/stop",function(){
        console.log("Boton parar clickeado");
        location.reload();
    })
});
```

Básicamente son dos listeners de click que hacen uso de la función AJAX GET de jQuery para realizar una petición a un endpoint llamado 'start' para iniciar y a otro endpoint llamado 'stop' para dejar de ejecutar las predicciones, de lado del servidor de predicciones se tiene el código siguiente donde se escuchan las peticiones.

Al no tener que renderizar nada desde el servidor de predicciones, simplemente lo que se hizo fue tener en cuenta una variable global start_stream de tal manera que esta controlará en el script principal si se hace o no la predicción sobre la imagen, esto con el fin de poder utilizar este stream de imagen quizás con otro algoritmo de detección que se quiera utilizar desde otra plataforma ofreciendo el servicio tanto de predicción como de streaming de imagen desde el dispositivo o interfaz web sólo con acceder al endpoint principal en el puerto 3500.

```
@app.route('/start')
@cross_origin(supports_credentials=True)
def start():
    global start_stream
    print("peticion start",     file=sys.stderr)
    start_stream = True
    print(start_stream, file=sys.stderr)
    return Response(status=200)

@app.route('/stop')
@cross_origin(supports_credentials=True)
def stop():
    global start_stream
    print("peticion stop",     file=sys.stderr)
    start_stream = False
    print(start_stream, file=sys.stderr)
    return Response(status=200)
```

Los print que se ven en este tramo del script son únicamente con fines de verificación de funcionamiento y no tienen nada qué ver con el funcionamiento final de la aplicación, lo único es que se devuelve al final un código de status 200 que significa OK en HTTP.

5.7. Pantalla de gráficas

Lo que se pretendió en esta sección fue realizar unos gráficos por característica de tal manera que en uno de ellos sea posible ver el conteo por tipo de vehículo, el conteo por color de vehículo y finalmente la cantidad de vehículos totales, esto implica la existencia de más de una serie de tiempo por gráfica, estas gráficas fueron obtenidas con ayuda de chartJs.

5.7.1. Gráficos con los datos de la base de datos en tiempo real

Para lograr este objetivo, se realizaron 3 tipos de gráfica diferentes, una que mostrará el conteo total por día de vehículos entre un rango de fechas establecido, la siguiente mostrará un conteo total de vehículos clasificados por tipo en el mismo intervalo trabajando con los tipos que se clasificaron en secciones anteriores y finalmente la última mostrará el conteo de vehículos por color.

Ya que el código para realizar estas gráficas es muy largo y se encuentra en el repositorio que se entrega junto con el proyecto, no se pondrá en esta sección sino que sólo se procederá a explicar a grandes rasgos como se logró dicha funcionalidad.

En la figura 5.4 se puede observar la primera pantalla que es con la que se ve la gráfica del conteo total de carros en un intervalo de fechas, esta gráfica tiene una resolución en el eje X de días, para posteriores versiones la idea es que se pueda mejorar esa resolución.

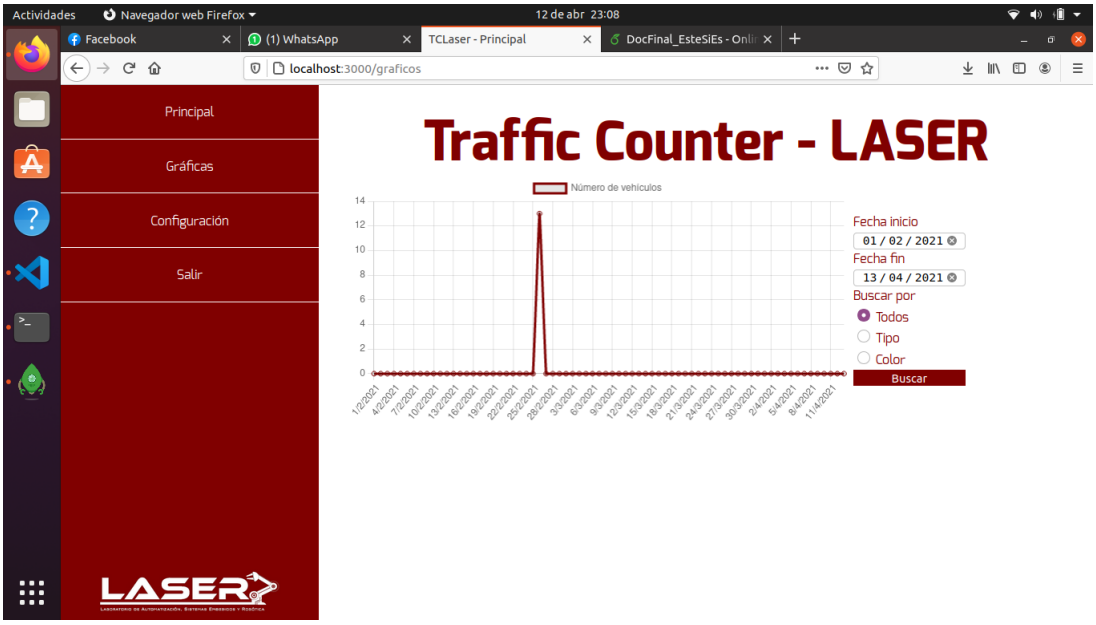


Figura 5.4: Pantalla de gráfica de conteo total de vehículos.

A partir de peticiones asíncronas al back-end que es el que se comunica con la base de datos, se logró traer la información de la base de datos que se actualiza en tiempo real acorde a la cámara, dependiendo de la selección de los controles que se pueden ver a la derecha de la figura 5.4 se puede obtener el conteo total, el conteo total discriminado por tipo o el conteo total discriminado por color, el de tipo se puede apreciar en la figura 5.5.

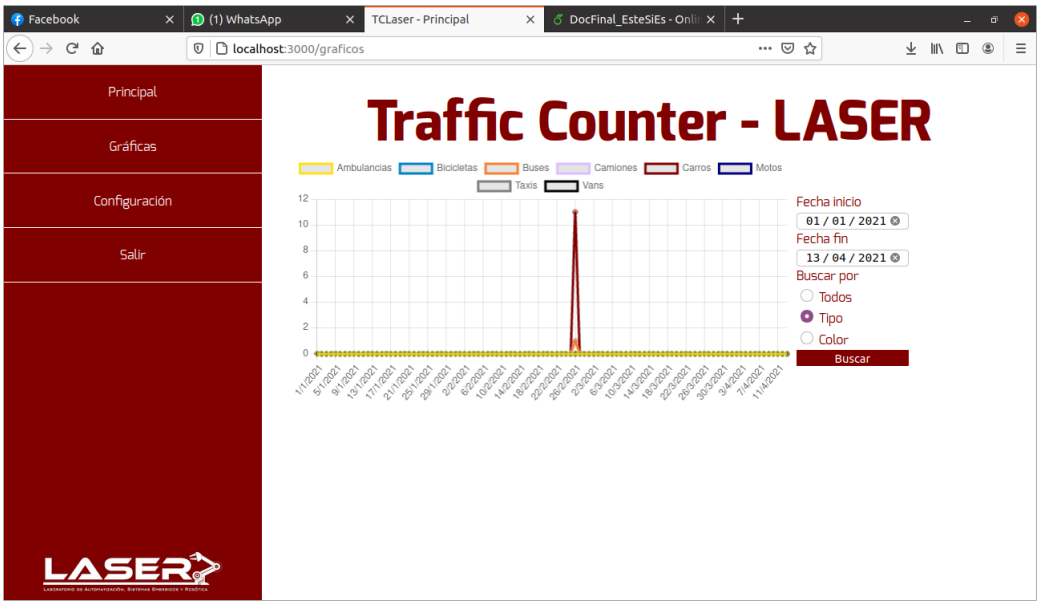


Figura 5.5: Pantalla de gráfica de conteo total de vehículos discriminados por tipo.

Para obtener los datos se hace la comunicación entre el back-end y el front-end a partir de promesas, una promesa es una interfaz que se implementa para

la comunicación asincrónica de páginas web, de tal manera que la gráfica sólo se actualizará apenas la promesa tenga un estado de fulfilled o cumplido, es decir, que la información ya se encuentre almacenada en el front-end para su posterior uso.

5.8. Resumen de la versión 1.0

En la versión 1.0 se puede encontrar la implementación de una plataforma con acceso a la cámara web sobre la cual se harán predicciones para hacer un conteo de vehículos y la clasificación posterior de los mismos, el acceso a la cámara se puede utilizar como feed para otro tipo de algoritmos de clasificación o pruebas ya que quedaró disponible para su uso en el puerto HTTP 3500. Además de esto, el script de Python almacena los datos que se toman en una base de datos de MongoDB (NoSQL) para poder hacer uso de ellos en etapas posteriores, por ejemplo, cuando se quieran mostrar las gráficas.

La interfaz web fue desarrollada utilizando HTML5, CSS, jQuery para front-end y nodejs para back-end, de tal manera que el uso de las peticiones fuera sencillo para hacer la lectura de los datos desde el front end a partir de promesas que fueron utilizadas para alimentar los gráficos. Esta versión cuenta con 3 tipos de gráfico de resolución por día, estos son conteo total de vehículos, conteo total de vehículos discriminado por tipo y conteo total de vehículos discriminado por color. Al ser una interfaz simple el acceso se controló mediante una clave quemada en el código que es 12345, sin embargo, la clave no está escrita en ningún lado del software. Para manejar la sesión se utilizó una cookie que se define como verdadera cuando la contraseña es correcta y como falsa cuando la contraseña es incorrecta, carga el sistema o se da click al botón de salir.

Finalmente para una versión posterior se deja el cambio de resolución en el eje X de las gráficas para que sea posible especificar por horas y minutos la cantidad de carros que se quiere visualizar, sin embargo, esto no es impedimento para obtener los datos ya que quedan almacenados en la base de datos anteriormente mencionada como archivos JSON y se puede utilizar cualquier aplicativo externo para accesarlos, descargarlos y hacer lo que se requiera con ellos.

Una de las salvedades que hay que hacer es que, para que corra utilizando docker, la interfaz debe ser a través de linux de tal manera que la conexión de las cámaras sea a partir de las carpetas dev, ya que windows tiene un problema con sus contenedores y no permite que se conecte virtualmente la cámara a ellos, por lo tanto la plataforma funcionará correctamente en sistemas operativos MacOS y distribuciones de Linux.

5.9. Modificaciones para la Coral Dev Board

Finalmente después de tener un algoritmo funcional, una plataforma portable que funciona en diferentes sistemas operativos, se procede a la encapsulación de la misma en un sistema embebido capaz de ejecutar nodejs y Python. Uno de los problemas más grandes acá es que las redes neuronales que se entrenaron de manera normal y la base de datos de mongoDB ya no servirán debido a que hay que optimizarlas de mejor manera para que corran en el acelerador de tensores EdgeTPU de la Google

Coral Dev Board, el problema con la base de datos de mongoDB es que no tiene una distribución y dejaron de dar soporte a las versiones para procesadores ARMv8 que son con los que cuentan la RaspberryPi en varias de sus versiones y la Google Coral Dev Board en este caso.

La EdgeTPU de Google necesita redes que funcionen con enteros de 8 bits para poder hacer uso de sus unidades aceleradoras de tensores, hay que tener en cuenta que al ser de Google permite un proceso de portabilidad que ya está documentado en las páginas, esto se realiza mediante Tensorflow Keras y Tensorflow Lite de tal manera que se consigue una gráfica congelada optimizada en enteros de 8 bits, para eso se necesita tener en cuenta la cuantización de los pesos, desviaciones y entradas después del entrenamiento, proceso que se explicará en la siguiente sección, además de esto, se deben compilar las gráficas congeladas de Tensorflow Lite para que corran en la EdgeTPU, normalmente la Google Coral Dev Board puede ejecutar el MobileNetV2 SSD a unos 70 FPS lo cual significa un gran rendimiento.

5.9.1. Quantización después del entrenamiento

La cuantificación posterior al entrenamiento es una técnica de conversión que puede reducir el tamaño del modelo al mismo tiempo que mejora la latencia del acelerador de CPU y hardware, con poca degradación en la precisión del modelo, la figura 5.6 muestra gráficamente el proceso básico para crear un modelo que sea compatible con Edge TPU.

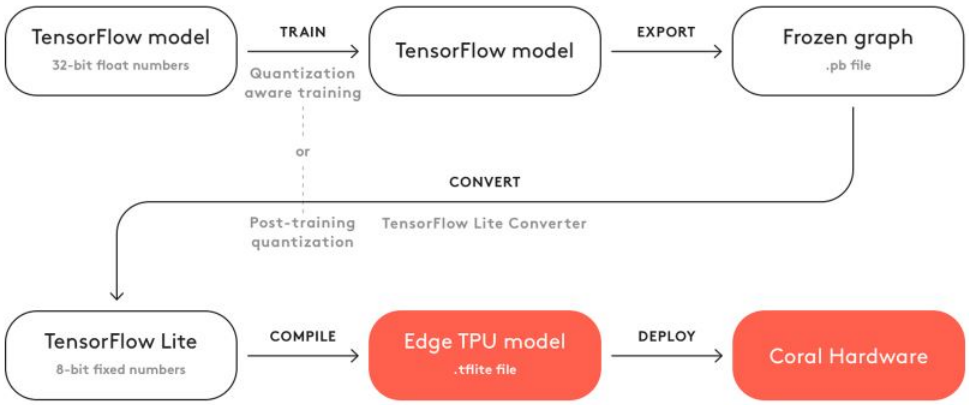


Figura 5.6: Flujo para tener un modelo compatible con Edge TPU [Coral, 2019]

Cuantizar un modelo significa convertir todos los números de coma flotante de 32 bits (como pesos y salidas de activación) a números de coma fija de 8 bits más cercanos. Esto hace que el modelo sea más pequeño y más rápido. Y aunque estas representaciones de 8 bits pueden ser menos precisas, la precisión de la inferencia de la red neuronal no se ve afectada significativamente.

La cuantificación completa después del entrenamiento de enteros no requiere ninguna modificación en la red, por lo que puede usar esta técnica para convertir una red previamente entrenada en un modelo cuantificado. Sin embargo, este proceso de conversión requiere que se proporcione un conjunto de datos representativo. Este

conjunto de datos representativo permite que el proceso de cuantificación mida el rango dinámico de activaciones y entradas, que es fundamental para encontrar una representación precisa de 8 bits de cada peso y valor de activación [Coral, 2019].

En la tabla 5.1 se muestran los resultados de la precisión antes y después de la optimización, es decir, en punto flotante de 32 bits y en enteros de 8 bits.

```
representative_dataset = representative_dataset_gen()
modelo = load_model("arq4_1.hdf5")
converter = tf.lite.TFLiteConverter.from_keras_model(modelo)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset_gen
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8  # or tf.uint8
converter.inference_output_type = tf.int8  # or tf.uint8
tflite_quant_model = converter.convert()

with open('modelo_VnV_Edge.tflite','wb') as f:
    f.write(tflite_quant_model)
```

Con el código anterior que se provee en la documentación para quantizar después del entrenamiento se puede obtener una gráfica congelada de Tensorflow que se pasará a través del compilador de EdgeTPU, esto únicamente será para que haga uso del dispositivo y no afectará en nada los resultados de la tabla 5.1.

Modelo	Precisión Float 64	Precisión Int 8
Modelo de Vehículo / No vehículo	96.95 %	95.21 %
Modelo de Tipo de Vehículo	86.21 %	80.28 %

Tabla 5.1: Afectación de la precisión después de la quantización del modelo.

5.9.2. Compilador EdgeTPU

El compilador edge TPU es una herramienta de línea de comandos que compila un modelo (.tflite) de TensorFlow Lite en un archivo que es compatible con Edge TPU, esta herramienta se puede ejecutar en cualquier sistema Linux moderno basado en Debian aunque se deben cumplir requerimientos como tener la versión de 64 bits de Debian 6.0 o superior, o cualquier derivado de la misma (como Ubuntu 10.0+) y también que la arquitectura del sistema sea de x86-64.

Edge TPU tiene aproximadamente 8 MB de SRAM que pueden almacenar en caché los datos de los parámetros del modelo. Sin embargo, una pequeña cantidad de RAM se reserva primero para el ejecutable de inferencia del modelo, por lo que los datos de los parámetros usan el espacio que queda después de eso. Naturalmente, guardar los datos de los parámetros en la RAM Edge TPU permite una velocidad de inferencia más rápida en comparación con la obtención de los datos de los parámetros de la memoria externa [GoogleResearch, 2019]. Finalmente cabe aclarar que después de quantizado el modelo en la sección anterior, este no sufrirá cambios en la precisión o función de pérdida que tuvo al ser implementado.

5.9.3. Carcasa para el dispositivo final

La idea de tener un dispositivo es para dejar un avance al grupo para que pueda retomar este proyecto tal vez haciendo uso de otros modelos o para hacer redes de dispositivos igual a este que estén comunicados entre ellos para formar una red IoT, entre otras cosas, o tal vez para diseñarle un sistema de energía autónomo que se mantenga con energía solar, las ideas son varias, sin embargo, una de las partes importantes será la implementación de una carcasa para el dispositivo final.

Hay que tener en cuenta las dimensiones de la Google Coral Dev Board que se pueden ver en la figura 5.7, pues estas son las que debe asegurar la carcasa, para esto también se tiene en cuenta el alto del disipador que trae un ventilador incrustado debido a que la generación de calor de la EdgeTPU es bastante.

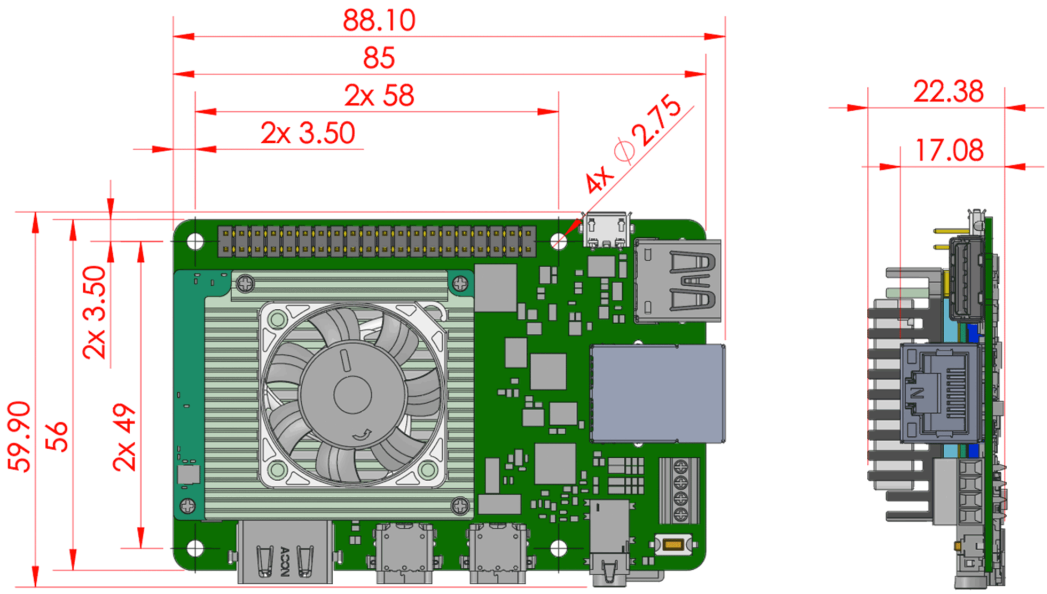


Figura 5.7: Dimensiones de la Google Coral Dev Board [Google, 2019]



Figura 5.8: Perspectiva lateral.



Figura 5.9: Perspectiva lateral.



Figura 5.10: Perspectiva trasera.

Luego de tener todas estas consideraciones en cuenta en las figuras 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, se pueden ver todas las perspectivas de la carcasa del dispositivo, debido a que la cámara iba a ser una parte importante de él, se dejó libre el flex para cambiarla en caso

de que sea necesario, además de poder realizar la conexión y desconexión de la misma en caso de presentarse alguna falla o tener que ser reemplazada.



Figura 5.11: Perspectiva frontal.



Figura 5.12: Perspectiva lateral.

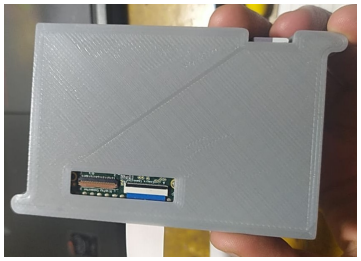


Figura 5.13: Perspectiva lateral.

Este fue el producto final obtenido del desarrollo del proyecto en cuestión, al visitar el puerto 3000 HTTP de la dirección propia cuando se encuentra en red, se puede observar la misma interfaz web y en el puerto 3500 el stream de la cámara que tiene conectado.

Capítulo 6

Conclusiones

- Al realizar las pruebas con las diferentes cámaras, se decidió optar por la cámara de la Google Coral Dev Board, esta tiene unas especificaciones muy similares a las de la Raspberry Camera v1.3, a pesar de ser un poco más bajas, al lograr la implementación del algoritmo acá diseñado, se puede hacer uso de la unidad de tensores de la Google Coral Dev Board, la cual da una mayor velocidad de inferencia la cual es necesaria en este caso para calcular características como la velocidad.

  Ya que las regiones de interés de las redes se redimensionaron a 200x200, no es necesario que la cámara tenga una resolución tan grande, sin embargo, la de la cámara anteriormente mencionada ya entra en el rango de High Deffinition.

- Debido a la contingencia presentada en los años 2020 y 2021 donde se llevó a cabo la realización del proyecto, no fue posible generar la base de datos completamente por los autores del proyecto, sin embargo, al reunir unas imágenes de internet con vehículos y partes de camino, se logró un total de 29349 imágenes como se puede apreciar en la tabla 2.1, de éstas se utilizaron 10810 para hacer la clasificación en vehículo - no vehículo y 18539 para realizar la clasificación de tipo de vehículo, las imágenes se encuentran en una buena resolución basados en las entradas de otras arquitecturas del estado del arte en clasificación de imágenes, por ejemplo, ResNET50 y MobileNetV2 utilizan una entrada de 224x224.

- En las tablas 3.1 y 3.2 se puede encontrar la experimentación con redes neuronales de tipo Feedforward, estas mostraron un desempeño aceptable en la clasificación de imágenes cuando se trataba de decidir si era un vehículo o no era un vehículo, sin embargo al llegar a la clasificación de tipo con las arquitecturas escogidas, los resultados no fueron buenos por lo cual la red de clasificación de tipo Feedforward fue descartada para una implementación final. Esta tarea fue una opción para no dejar de lado la importancia de la ingeniería de características en las tareas de clasificación.

  Por otro lado en las tablas 3.3 y 3.4 se pueden apreciar los resultados obtenidos para la clasificación de imágenes mediante redes convolucionales, al tener filtros extractores de características sintonizados mediante el aprendizaje, mostraron un mejor desempeño en las tareas de clasificación binaria (para ver si era vehículo o no) y un comportamiento mucho mejor en la tarea de clasificación de tipo de vehículo, logró subir de porcentajes del orden del 55 % a porcentajes de 65 %, sin embargo, aún no son lo suficientemente buenos para elegir en el tipo de vehículo.

El algoritmo genético ejecutado tiene resultados que se pueden ver en las tablas 3.5 y 3.6, se puede ver que su desempeño aunque comparable con el backpropagation, fue más pobre, esto puede deberse a que no se guarda la relación entre el genoma y la ubicación espacial de los filtros convolucionales como sí lo tiene en cuenta el back propagation con descenso de gradiente, la heurística de optimización no fue la preferida por esto. Finalmente en las tareas de transfer learning, las tablas 3.8 y 3.9 tienen los resultados, se puede ver que para las tareas de clasificación tanto binaria como múltiple, es decir, vehículo - no vehículo y tipo de vehículo respectivamente, tienen un comportamiento bastante sobresaliente ya que comparados con las redes anteriores, pasaron a porcentajes de más del 80 % en la tarea de clasificación de tipo de vehículo, sin embargo, en la matriz de confusión mostrada en la figura 4.10, se puede observar que hay un sesgo hacia la clase de carro, además de una mala precisión en las etiquetas que tienen mayores items, lo que lleva a pensar que habría que cambiar el tamaño de la imagen de entrada de las redes neuronales para poder hacer un mejor clasificador, si hay más información de entrada, los filtros podrían aprender otro tipo de características que se hayan perdido cuando la información era más pequeña, esto se deja a consideración para futuros trabajos.

- Como se puede observar en las tablas 4.2, 4.3 y 4.4 la implementación del software tiene un error de conteo de vehículos cercano al 20 % para todos los vídeos, sin embargo, cuando se trató de la clasificación del tipo de vehículo, se plantea en la conclusión anterior aumentar el tamaño entrada de la imagen de tal manera que sea posible que los filtros extraigan más información o información que no se haya tenido en cuenta ya que se hizo un redimensionamiento de las imágenes para el entrenamiento y la validación. La interfaz que se desarrolló en Python permitió observar la cantidad de vehículos contados además de tener almacenadas diferentes características como el color y la velocidad de cada objeto que pasó a lo largo del campo de visión del algoritmo.

- Además de obtener una predicción acertada con un porcentaje de exactitud bastante bueno, se puede utilizar el stream que genera la cámara del dispositivo para poder correr otro tipo de predicciones, lo cual puede ser útil para probar modelos diferentes una vez que el dispositivo se encuentre montado en algún lado.

- Una de las metas más importantes logradas es la obtención de los datos en una plataforma de tal manera que puedan ser utilizados para hacer modelos de predicción, entre los términos de construcción de la plataforma no se contempla en ningún momento que el web scraping no sea permitido para tomar los datos, demás está que para versiones posteriores debe haber una manera más fácil de descargar la base de datos o de generar un archivo CSV con los datos relevantes para cualquier trabajo. En las figuras 5.4 y 5.5 se puede ver la interfaz que quedará como definitiva para la plataforma.

6.1. Trabajos Futuros

- Se propone como trabajo futuro, hacer un sistema de gestión de energía autónomo a través de páneles solares que permitan que el sistema esté encendido con una batería y ésta se cargue a través de los páneles de tal manera que el sistema pueda estar corriendo 24/7.

- Por otro lado, la gestión del dispositivo embebido está fuera del alcance del proyecto dejando nada más un inicio de lo que podría ser un gran proyecto de IoT (Internet of Things) debido a que además de hacer la red capaz de gestionar los dispositivos, se debe hacer una gestión de memoria debido a que los JSONs almacenados ocuparán espacio que no es infinito ni bastante grande debido a la corta memoria de los dispositivos tales como Raspberry, Google Coral Dev Board o Jetson Nano, a pesar de que se encuentran memorias SD's en el mercado de gran capacidad, eso haría que el costo de un único dispositivo se incremente lo cual desde la parte de costos del proyecto, no estaría bien. Por lo tanto, se propone la ampliación al paradigma de Internet of Things.

Capítulo 7

Anexos

- Carpeta de Google Drive de vídeos de prueba tomados de internet y elaboración propia: `https://drive.google.com/drive/folders/1Yxfnp1nrzea5RO4a-hSjv6Sk2imsSZOC`
- Repositorio del proyecto en GitHub: `https://github.com/LASER-UD/ConteoEIdentificacionDeTrafico`
- Repositorio final del proyecto para desplegar en un PC normal `https://github.com/LASER-UD/FinalConteoEIdentificacionDeTrafico/`
- Repositorio final del proyecto para desplegar en una Google Coral `https://github.com/LASER-UD/CoralTCLaser`

Bibliografía

[Abdelwahab, 2019] Abdelwahab, M. A. (2019). Accurate vehicle counting approach based on deep neural networks. pages 1–5.

[Ahmed et al., 2019] Ahmed, G., Abdelwahab, Moataz, M., A.-Z., Tsubasa, M., Taniguchi, and Rin-ichiro (2019). Robust vehicle detection and counting algorithm employing a convolution neural network and optical flow. *Sensors*, 19:4588.

[Aires., 2013] Aires., U. B. (2013). Seminario de electrónica: Sistemas embebidos. *Argentina). Sistemas embebidos: Sistemas de tiempo real. Buenos Aires: Universidad de Buenos Aires, 2013*, 55.

[Arducam, 2021] Arducam (2021). https://www.amazon.com/-/es/Arducam-M [Online; accessed 07-Marzo-2021].

[Arinaldi et al., 2018] Arinaldi, A., Pradana, J. A., and Gurusinga, A. A. (2018). *Detection and classification of vehicles for traffic video analytics*, volume 144. Procedia Comput. Sci.

[Arróspide et al., 2012] Arróspide, J., Salgado, L., and Nieto, M. (2012). Video analysis based vehicle detection and tracking using an mcmc sampling framework. *EURASIP Journal on Advances in Signal Processing*, 2012.

[Bianco et al., 2018] Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). *Benchmark Analysis of Representative Deep Neural Network Architecttures*. IEEE Access.

[Bishop, 2006a] Bishop, C. (2006a). *Pattern Recognition and Machine Learning*. Singapur, Springer Science.

[Bishop, 2006b] Bishop, C. M. (2006b). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Coral, 2019] Coral (2019). https://coral.ai/docs/edgetpu/models-intro/#compiling. [Online; accessed 13-Marzo-2021].

[Danelljan et al., 2014] Danelljan, M., Hager, G., Khan, F. S., and Felsberg, M. (2014). *Accurate Scale Estimation for Robust Visual Tracking*. Proceedings of the British Machine Vision Conference.

[Darwin, 1936] Darwin, C. (1936). *The origin of species*. Everyman's library. Dent.

[Davida, 2018] Davida, B. (2018). https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb. [Online; accessed 08-Marzo-2021].

[Desealers et al., 2008] Desealers, T., Pimenidis, L., and Ney", H. (2008). Bag-of-visual-words models for adult image classification and filtering.

[Ferrero and Mora, 2008] Ferrero, J. A. and Mora, S. L. (2008). *Imagen Digital*. Universidad de Alicante, Alicante.

[Fielding and Taylor, 2000] Fielding, R. T. and Taylor, R. N. (2000). *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, Irvine.

[Foundation, 2006] Foundation, O. J. (2006). jquery learning center. https://learn.jquery.com/. [Online; accessed 06-Marzo-2021].

[Galeano, 2007] Galeano, G. (2007). *Programación de Sistemas Embebidos en C*. Alfaomega.

[Gonzalez and Woods, 2002] Gonzalez, R. and Woods, R. (2002). *Digital Image Processing*. New Jersey, 2nd Ed, Prentice Hall.

[Google, 2019] Google (2019). https://coral.ai/docs/dev-board/datasheet/#mechanical-dimensions. [Online; accessed 13-Marzo-2021].

[GoogleCoral, 2021] GoogleCoral (2021). https://coral.ai/products/dev-board/. [Online; accessed 07-Marzo-2021].

[GoogleResearch, 2019] GoogleResearch (2019). https://coral.ai/docs/edgetpu/compiler/#system-requirements. [Online; accessed 13-Marzo-2021].

[Halfacree, 2018] Halfacree, G. (2018). *The official Raspberry Pi Beginner's guide - How to use your new computer*. Raspberry Pi Trading Ltd.

[Haupt and Haupt, 1998] Haupt, R. L. and Haupt, S. E. (1998). *Practical Genetic Algorithms*. John Wiley and Sons, Inc., USA.

[Ho and Wookey, 2020] Ho, Y. and Wookey, S. (2020). The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *CoRR*, abs/2001.00570.

[Huttunen, 2019] Huttunen, A. P. H. (2019). Tau vehicle type recognition competition. *https://www.kaggle.com/c/vehicle*.

[JetsonNano, 2021] JetsonNano (2021). https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Online; accessed 07-Marzo-2021].

[Johnston and team, 2013] Johnston, S. and team (2013). https://www.docker.com/company. [Online; accessed 09-Marzo-2021].

[Jorge, 2001] Jorge, D. (2001). *Redes neuronales: conceptos básicos y aplicaciones*. Universidad Tecnológica Nacional – Facultad Regional Rosario, Rosario.

[Laroca et al., 2021] Laroca, R., Zanlorensi, L. A., Gonçalves, G. R., Todt, E., Schwartz, W. R., and Menotti, D. (2021). An efficient and layout-independent automatic license plate recognition system based on the yolo detector. *IET Intelligent Transport Systems*.

[Leelasantitham and Wongseree, 2008] Leelasantitham, A. and Wongseree, W. (2008). *Detection and classification of moving Thai vehicles based on traffic engineering knowledge*. 8th International Conference on ITS Telecommunications, Phuket.

[Li and Venkatesan, 2017] Li, B. and Venkatesan, R. (2017). *Convolutional Neural Networks in Visual Computing a concise guide*. CRC Press.

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. 9905:21–37.

[Lowe, 2004] Lowe, D. G. (2004). *Distictive Image Features from Scale-Invariant Keypoints*.

[Marcomini and Cunha, 2018] Marcomini, L. A. and Cunha, A. L. (2018). *A Comparison between Background Modelling Methods for Vehicle Segmentation in Highway Traffic Videos*. Preprint.

[Opencv-documentation, 2021] Opencv-documentation (2021). https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html. [Online; accessed 09-Marzo-2021].

[OpenJS, 2010] OpenJS (2010). About nodejs. https://nodejs.org/es/about/. [Online; accessed 06-Marzo-2021].

[Raspberry1.3, 2021] Raspberry1.3 (2021). https://es.dhgate.com/product/camera-module-board-rev-1-3-5mp-webcam-video/426839840.html. [Online; accessed 07-Marzo-2021].

[Raspberry2.1, 2021] Raspberry2.1 (2021). https://www.vistronica.com/board-de-desarrollo/raspberry-pi/camara-raspberry-pi-v2-de-8mpx-detail.html. [Online; accessed 07-Marzo-2021].

[RaspberryPi4, 2021] RaspberryPi4 (2021). https://www.raspberrypi.org/blog/8gb-raspberry-pi-4-on-sale-now-at-75/. [Online; accessed 06-Marzo-2021].

[RaspberryPiZeroW, 2021] RaspberryPiZeroW (2021). https://www.sparkfun.com/products/14277. [Online; accessed 06-Marzo-2021].

[Redmon and Farhadi, 2018] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement.

[Ronacher, 2006] Ronacher, A. (2006). Welcome to flask. https://flask.palletsprojects.com/en/1.1.x/. [Online; accessed 06-Marzo-2021].

[S. Aqel and Aarab, 2017] S. Aqel, A. Hmimid, M. A. S. and Aarab, A. (2017). *Road traffic: Vehicle detection and classification*. 2017 Intelligent Systems and Computer Vision (ISCV).

[S. Kul and Sayar, 2017] S. Kul, S. E. and Sayar, A. (2017). *A concise review on vehicle detection and classification*. International Conference on Engineering and Technology (ICET).

[Sandler et al., 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[Song et al., 2019] Song, H., Liang, H., Li, H., et al. (2019). Vision-based vehicle detection and counting system using deep learning in highway scenes. *Eur. Transp. Res. Rev. 11*, 11(51).

[Sun et al., 2017] Sun, M., Wang, Y., Li, T., Lv, J., and Wu, J. (2017). Vehicle counting in crowded scenes with multi-channel and multi-task convolutional neural networks. *Journal of Visual Communication and Image Representation*, 49:412–419.

[Vertebrar, 2021] Vertebrar (2021). https://vertebrar.com/servicios/inteligencia-artificial/aprendizaje-profundo/. [Online; accessed 08-Marzo-2021].

[Vrahatis et al., 2000] Vrahatis, M., Magoulas, G., Parsopoulos, K., and Plagianakos, V. (2000). Introduction to artificial neural network training and applications.

[Zheng and Casari, 2018] Zheng, A. and Casari, A. (2018). *Feature Engineering for Machine Learning*. O' Reilly Media, Inc.