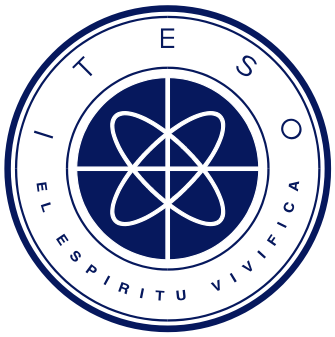
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

MAESTRÍA EN SISTEMAS COMPUTACIONALES



DETECCIÓN VISUAL DE VEHÍCULOS AUTOMOTRICES EN AMBIENTES REALES

Trabajo recepcional que para obtener el grado de

MAESTRO EN SISTEMAS COMPUTACIONALES

Presenta: Ing. Francisco de Jesús Méndez Pelayo

Director: Dr. Luis Enrique González Jiménez

Tlaquepaque, Jalisco. 22 de enero de 2018

Agradecimientos

A través de estas líneas, me permito enumerar mis agradecimientos para aquellos que han sido parte importante en mi vida y para este trabajo. Me gustaría expresar con alegría en primer lugar, mi agradecimiento por todas las bendiciones recibidas inmerecidamente por parte de Dios en cada uno de los momentos transcurridos, ¡tu presencia siempre ha sido tan real aun en las dificultades más grandes!

Con especial cariño a Sandra por tu motivación y compromiso en cada proyecto nuevo juntos, por soportar conflictos aun estando en completo desacuerdo. Especialmente, porque me has ayudado a descubrir que los compromisos con hechos se convierten en una forma esencial para expresar el amor y de tantas formas demostrarme tu amor.

A mis padres, que me han enseñado más de lo que creo, que siempre me mostraron la importancia del trabajo bien hecho desde las cosas más simples, que me ayudaron a seguir adelante en etapas nebulosas de mi vida.

A mis hermanos y hermana, que han sido también mis primeros amigos, de ellos también he aprendido cosas muy valiosas, hemos compartido experiencias que guardo y seguiré guardando en mi corazón, hemos descubierto que la fraternidad es en realidad un don que se experimenta y se vive cuando se decide hacerlo.

Al CONACYT, que ha permitido este proyecto y toda la maestría con el apoyo económico destinado a colegiaturas de las asignaturas, dicho apoyo fue el 424514.

Por último, no por su importancia sino por su cercanía al proyecto, a mi asesor Luis Enrique, fueron varios altibajos por los que pasamos, pero siempre tú postura positiva y tú apertura para ayudar en cada etapa me impulsaron a solucionar los problemas presentes en el trabajo. Algunas sesiones iniciábamos hablando de procesamiento de imágenes y terminábamos hablando de música, eso siempre dio dinamismo a la forma en cómo me compartías tus experiencias para aplicarlas en este trabajo. Solamente queda agradecerte los esfuerzos extra que realizaste para solucionar dificultades como el tiempo y las imposibilidades de asistir físicamente a sesiones, haciendo video llamadas a horas fuera del trabajo.

Dedicatoria

A mis padres, ustedes han sido mis primeros maestros, ustedes instalaron los principios más fundamentales en mi persona. Siempre recordaré con afecto y cariño el esfuerzo que realizaron por mí y mis hermanos.

Resumen

Entre los algoritmos de detección de objetos, cascade ha demostrado ser uno de los más robustos y flexibles siendo aplicado sobre un gran número de diferentes tipos de objetos. La detección de rostros fue la primera aplicación y hoy en día este algoritmo se sigue utilizando para muchos sistemas en producción. De igual forma, uno de los grandes objetivos buscados en los últimos años ha sido el de diseñar un vehículo completamente autónomo y donde la conducción se realice de forma automática sin intervención humana. Es por esto por lo que ha utilizado la combinación de algoritmos cascade y Adaboost para crear un sistema que sea capaz de detectar vehículos de forma eficiente.

Como base para este trabajo se ha utilizado la implementación de OpenCV, dicho software se distribuye bajo una licencia *open source*, la cual nos ha permitido realizar cambios en la implementación de las características tipo HAAR para agregar una serie de características capaces de aumentar el poder de reconocimiento de vehículos. Estas características en conjunto las que originalmente se encuentran implementadas por OpenCV, han permitido mejorar los niveles de detección de vehículos en secuencias de imágenes, además, con los entrenamientos realizados se pudo observar cierta reducción en el número de falsos negativos.

De acuerdo con la el esquema de este conjunto de algoritmos, adaboost es el encargado de realizar el entrenamiento, entonces, es durante el entrenamiento que se definen los tipos de características tipo HAAR que se utilizarán tanto en el entrenamiento como durante la etapa de detección. Durante el entrenamiento, dicho conjunto de características sirve únicamente como referencias para generar las ventanas de búsqueda durante el proceso de detección.

Índice

Agradecimientos 1
Dedicatoria 2
Resumen 3
Índice 4
Lista de Figuras 6
Lista de Tablas 7
1. Introducción 8
2. Objetivos 11
3. Estado del Arte 12
4. Marco Conceptual 12
4.1. Imágenes digitales y espacio de colores 13
4.2. Wavelet Haar 14
4.3. Características 15
4.3.1. Características tipo Haar 16
4.4. Imagen integral 17
4.5. Árboles de decisión 19
4.5.1. Métodos combinados de aprendizaje (ensemble) 20
4.6. Algoritmos de aprendizaje y clasificación 21
4.6.1. Adaboost 21
4.6.2. Cascade 23
4.7. Estructura de OpenCV 26
4.7.1. Estructura del archivo xml de entrenamiento 28
5. Metodología de Trabajo 32
5.1. Preparación de los datos y elección de parámetros 32
5.2. Estrategias de evaluación 33
5.2.1. Método Holdout 33
5.2.2. Validación cruzada de k iteraciones (K-fold cross validation) 33
5.2.3. Validación cruzada dejando uno fuera (Leave one out cross validation) 33
6. Esquema De Clasificación Visual De Objetos Propuesto 34
6.1. Características propuestas 35
6.1.1. Características tipo esquina 35
6.1.2. Características tipo T 36

6.1.3. Características tipo semi-borde 37
6.2. Implementación de las características 38
6.3. Pruebas y comparación con las características por defecto en OpenCV 39
6.3.1. Proceso de validación de cada imagen 39
6.3.2. Validación del sistema 39
6.4. Resultados encontrados 40
7. Conclusiones y Trabajo Futuro 44
Bibliografía 45
Apéndice A: Ejemplo de Implementación de la Imagen Integral en C++ 47
Apéndice B: Código de las Nuevas Características Propuestas 48
Apéndice C: Licencia de OpenCV 50

Lista de Figuras

FIGURA 1: CLASIFICACIÓN DE LOS AGENTES EN LA ZONA CAPTURADA POR LA CÁMARA 8
FIGURA 2: MODELO ADITIVO DE COLORES EN RGB 13
FIGURA 3: VISTA DE UNA IMAGEN FORMADA POR PIXELES RGB 14
FIGURA 4: IMAGEN CON PIXELES MUY VISIBLES 14
FIGURA 5: LA FUNCIÓN WAVELET HAAR 15
FIGURA 6: CARACTERÍSTICAS TIPO HAAR 16
FIGURA 7: ÁREA SUMADA EN UNA IMAGEN INTEGRAL 17
FIGURA 8: REPRESENTACIÓN DE IMAGEN EN VALORES ENTEROS Y ESCALA DE GRISES 18
FIGURA 9: IMAGEN INTEGRAL CALCULADA 18
FIGURA 10: CALCULAR SUMA EN CUALQUIER SUB RECTÁNGULO 19
FIGURA 11: EJEMPLO DE UN ÁRBOL DE DECISIÓN 20
FIGURA 12: PROCESO DEL ALGORITMO ADABOOST 21
FIGURA 13: PROCEDIMIENTO DE ADABOOST 23
FIGURA 14: PROCESO DEL ALGORITMO CASCADE 24
FIGURA 15: REDIMENSIONADO SUB VENTANAS DE BÚSQUEDA 25
FIGURA 16: DETECCIÓN UTILIZANDO CASCADE Y LAS CARACTERÍSTICAS TIPO HAAR 26
FIGURA 17: ESTRUCTURA DE ARCHIVOS DE OPENCV 27
FIGURA 18: ESTRUCTURA COMPLETA DEL ARCHIVO DE ENTRENAMIENTO 28
FIGURA 19: REPRESENTACIÓN DE LAS ETAPAS 29
FIGURA 20: REPRESENTACIÓN DE LAS CARACTERÍSTICAS 30
FIGURA 21: CARACTERÍSTICA 1 SEGÚN EL ARCHIVO DE ENTRENAMIENTO 30
FIGURA 22: CARACTERÍSTICA 2 SEGÚN EL ARCHIVO DE ENTRENAMIENTO 31
FIGURA 23: ESQUEMA COMPLETO DEL SISTEMA 34
FIGURA 24: CARACTERÍSTICAS TIPO HAAR IMPLEMENTADAS EN OPENCV 35
FIGURA 25: CARACTERÍSTICAS TIPO HAAR PARA DESCRIBIR ESQUINAS 35
FIGURA 26: CARACTERÍSTICAS DE ESQUINA SOBRE UN AUTOMÓVIL 36
FIGURA 27: CARACTERÍSTICAS TIPO T 36
FIGURA 28: CARACTERÍSTICAS TIPO T EN UN VEHÍCULO REAL 37
FIGURA 29: CARACTERÍSTICAS TIPO SEMI BORDE 37
FIGURA 30: CARACTERÍSTICAS TIPO SEMI BORDE EN UN VEHÍCULO REAL 38
FIGURA 31: FLUJO DE LLAMADAS HASTA LA GENERACIÓN DE CARACTERÍSTICAS 38
FIGURA 32: PROCESO DE VALIDACIÓN 40
FIGURA 33: GRAFICA DE LAS 10 ITERACIONES USANDO LAS CARACTERÍSTICAS DE OPENCV 41
FIGURA 34: GRAFICA DE LAS 10 ITERACIONES USANDO LAS CARACTERÍSTICAS DE OPENCV MAS LAS PROPUESTAS 41
FIGURA 35: EJEMPLO 1 DE DETECCIÓN CON CARACTERÍSTICAS DE OPENCV (IZQUIERDA) Y LAS PROPUESTAS (DERECHA) Y EL MISMO NÚMERO DE ETAPAS DE ENTRENAMIENTO. 42
FIGURA 36: EJEMPLO 2 DE DETECCIÓN CON CARACTERÍSTICAS DE OPENCV (IZQUIERDA) Y LAS PROPUESTAS (DERECHA) Y EL MISMO NÚMERO DE ETAPAS DE ENTRENAMIENTO. 42
FIGURA 37: SECUENCIA DE IMÁGENES DE PRUEBA OBTENIDAS DE UN VIDEO DEL FLUJO VEHICULAR EN UNA AVENIDA. 43

Lista de Tablas

TABLA 1: MUESTRA LOS DISTINTOS TAMAÑOS DE LAS SUB VENTANAS 25
TABLA 2: PARÁMETROS USADOS EN EL ENTRENAMIENTO 33

1. Introducción

La visión computacional representa una gran oportunidad de sensar características diversas del ambiente. Para realizar tareas como vigilancia, evasión de obstáculos, empaletado de productos, entre otras, es necesario poder identificar de qué tipo y cuantos elementos se encuentran en determinado momento en el ambiente. Esto implica extraer características visuales de elementos desordenados de un conjunto de imágenes. Debido a esto, la visión se ha convertido en parte fundamental para poder desarrollar sistemas capaces de tomar decisiones en base a los objetos que se han reconocido en el entorno. Pero la exactitud es un factor que debe ser considerado, ya que un error en el reconocimiento podría desencadenar situaciones desastrosas. Entonces, los algoritmos de clasificación deben ser mejorados de modo que la posibilidad de tener errores en la detección sea prácticamente inexistente.

Otro aspecto importante es que los problemas de tráfico a nivel mundial son cada vez más grandes y diversos. Con el tiempo se ha ido incrementando el número de agentes que intervienen en una situación común y real, esta es la razón por la que dar a los automóviles la capacidad de tener más información del entorno que lo rodea se ha convertido en una necesidad. Pero se debe mencionar que un sistema así de complejo requiere de un gran poder de procesamiento, además de una gran cantidad de sensores que en tiempo real informen al sistema sobre posibles situaciones de riesgo, indicaciones de ruta a seguir basado en señales de tránsito, semáforos, obstáculos, peatones, vehículos, entre otros. Gracias a estas funcionalidades, se puede pensar en crear vehículos auto asistidos que hagan a los conductores más sencilla dicha tarea y con menos riesgo de accidentes e incluso que no tengan que intervenir en lo absoluto en las tareas de conducción.

Se han propuesto muchos algoritmos para detección de objetos, pero estos en su mayoría funcionan bien para cierto tipo de objetos. Es por esto por lo que el objetivo principal ha sido tomar un algoritmo que haya demostrado tener buenos resultados de detección en tiempo real además de haber sido utilizado en muchas aplicaciones. Una vez elegido el algoritmo, hacer los ajustes necesarios para mejorar su funcionamiento en el reconocimiento de vehículos de la vialidad utilizando una secuencia de imágenes de entrada capturadas del ambiente que rodea al vehículo.

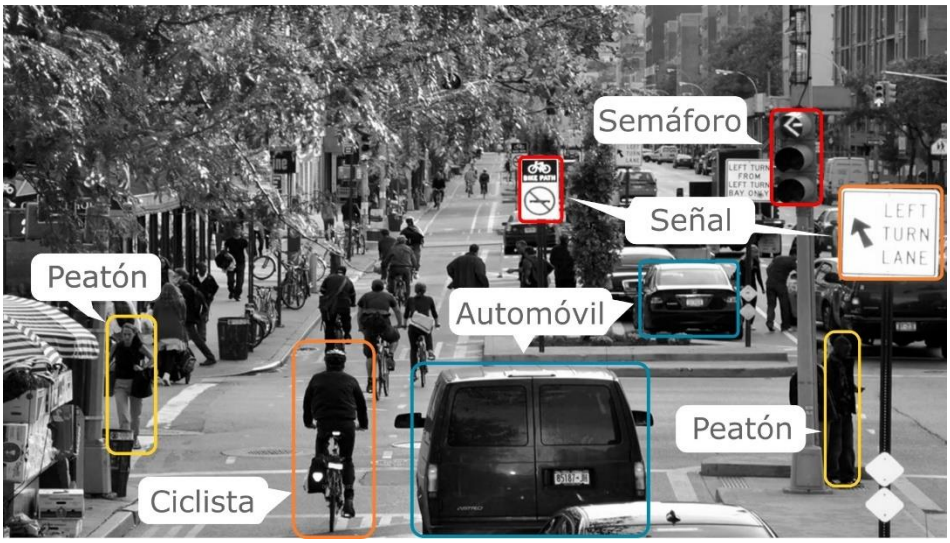


Figura 1: Clasificación de los agentes en la zona capturada por la cámara

Como podemos observar en la Figura 1, existen muchos agentes que rodean el vehículo entre los que se encuentran, por mencionar los más importantes, otros vehículos, peatones, ciclistas y señales de tránsito. Es importante mencionarlos, ya que estos, son solamente los objetos básicos para realizar un escaneo adecuado del ambiente y por lo tanto la lista de objetos podría crecer de forma que podría ser inmanejable.

Además de lo anterior, se ha analizado que no todos y cada uno de los objetos en la escena son importantes, ya que algunos están demasiado lejos o algunos son peatones que están fuera del alcance del vehículo. Por otro lado hay otro nivel de complejidad; existen algunos objetos que deberían ser clasificados como pertenecientes al mismo conjunto, aunque esto difiere del hecho de que ellos estén siquiera cerca de ser homogéneos, de modo que dos peatones aunque compartan ciertas características visuales, tienen otras que los hacen distintos, como podría ser el tipo de ropa, las características fisiológicas relacionadas a sus dimensiones, además del ángulo en que el peatón se encuentra con respecto a la cámara, si este lleva consigo otros objetos que puedan introducir ruido en la detección y esto sólo por mencionar algunos de ellos. Es esta una de las razones por las que la detección de objetos en ambientes reales es un problema que dista mucho de tener una solución trivial.

Dotar a un sistema vehicular con la capacidad de análisis respecto a qué y cuantos objetos tiene alrededor, le dará la capacidad de llegar a tomar decisiones importantes y, sobre todo, críticas. Por ejemplo, para el caso de un vehículo que se dirige a toda velocidad por una calle, cuando el conductor no se da cuenta que tiene un obstáculo enfrente (posiblemente un tope) el sistema sería capaz de lanzar una advertencia al conductor de que dicho obstáculo puede ser peligroso. Un caso más crítico podría ser el de detener un vehículo que está propenso a chocar con algún objeto que se encuentre en un "punto ciego" para el conductor. Y por qué no, poder llegar a hacer posible el vehículo auto conducible, el cual no requiera de la más mínima interacción con los tripulantes más que la indicación del punto al cual quieren dirigirse.

Aunque los puntos anteriores puedan entenderse como simples, conceptualmente hablando, el presente trabajo se centra en trabajar con un algoritmo que sea capaz de detectar objetos en la vialidad. De hecho, para ser más precisos, el presente trabajo se ha desarrollado pensando en encontrar las optimizaciones a un algoritmo de modo que sea capaz de dar mejores resultados en la detección de vehículos y, por lo tanto, no se plantea hacer detecciones de otro tipo de objetos como peatones o señales de tránsito, y tampoco es parte de este trabajo tomar decisiones con la información obtenida con el detector. Sin embargo, es importante aclarar que el proceso seguido para que el sistema detecte otros vehículos puede aplicarse sin mayores modificaciones para identificar otro tipo de patrones como el de personas, bicicletas o señales de tránsito.

En los últimos años, se han implementado varios sistemas que han permitido un avance en el desarrollo de algoritmos o técnicas que permiten reconocer algunos elementos particulares. Incluso se han propuesto algunas soluciones que hacen uso de varios sensores, lo cual de alguna manera ayuda a mejor la confiabilidad de detección. Pero, al mismo tiempo, hace que la complejidad general del sistema también se vea incrementada, dificultando el mantenimiento e incrementando los costos del sistema. Es por esto por lo que el algoritmo elegido solamente utiliza una cámara monocular simple de bajo costo, en comparación con otras soluciones más complejas.

La decisión ha sido tomar como base el par de algoritmos Adaboost y cascade utilizando características del tipo HAAR, el cual fue primeramente propuesto para detección de caras [1]. Este ha ofrecido resultados cada vez mejores, y debido a que se han realizado trabajos donde se han intentado usar este par de algoritmos para detección de vehículos, dando resultados buenos y en tiempo real [2]. De forma análoga a los trabajos originales, se ha utilizado Adaboost para llevar a cabo el entrenamiento

y cascade para hacer la detección en tiempo real.

2. Objetivos

Han surgido varias motivaciones detrás de este trabajo, entre los más importantes se puede mencionar que, aun cuando han surgido muchas implementaciones para detección de vehículos en la vialidad, algunos algoritmos actuales siguen utilizando exactamente varios elementos que originalmente fueron propuestos, como es el caso de los algoritmos Adaboost y cascade. Estos han sido utilizados para detección de vehículos, incluso se han hecho combinaciones con otros algoritmos como *kmeans*, los cuales han arrojado resultados con buenas mejoras. Pero algo que nos hemos dado cuenta durante la investigación, ha sido que las características han permanecido exactamente igual a las propuestas originalmente por Viola y Jones [1]. Es por esto por lo que un objetivo principal de este trabajo ha sido analizar las características originales y el tipo de objetos para el cual fueron diseñadas en primera instancia. Así, hemos explorado nuevas características que pudieran ser capaces de describir de mejor manera algunos aspectos de nuestro objeto a clasificar, en este caso vehículos. Una vez con una propuesta de las nuevas características, hacer comparaciones en la detección del sistema originalmente propuesto contra el sistema con las nuevas características y validar desempeños.

Otro objetivo importante era el uso de un *framework* potente y ampliamente extendido como es OpenCV [3], que además de lo anterior es *open source*. Esto nos ha permitido adentrarnos en la implementación de cada módulo, para de esta forma encontrar los puntos optimizables para nuestro reconocedor de vehículos. Por otro lado, el hecho de que OpenCV sea uno de los *frameworks* más ampliamente utilizados, y con más colaboración de la comunidad y de la industria, ha facilitado la asimilación rápida de sus implementaciones.

Al final de la investigación, el principal motivador de este trabajo ha sido el tener un sistema relativamente simple, que se comporte mejor para detectar vehículos que los trabajos anteriormente propuestos de modo que en un futuro se puedan proponer nuevas combinaciones de ideas previas como el caso de uno de los últimos trabajos en esta problemática [2]. Esto permitirá, utilizando las nuevas características propuestas aquí, que se puedan conseguir aún mejores resultados.

Por último, aunque proponer nuevas características aumente el tiempo de procesamiento en el entrenamiento, lo cual es inevitable, ya que mientras más características se deban procesar, más combinaciones de las mismas de deben crear; el tiempo de detección una vez entrenado, no debe verse afectado por las nuevas características. Así, la comparación de un entrenamiento con y sin nuestras características, debería dar resultados similares en cuanto a rendimiento se refiere.

3. Estado del Arte

La detección visual de objetos ha sido un campo de trabajo muy explotado durante muchos años, es por esto por lo que existen grandes avances, y aunque estos avances no han demostrado resultados óptimos, muchos de ellos han permitido muchas aplicaciones automatizadas de muchas áreas cómo la medicina [4], otros en temas de demografía y políticas socioeconómicas [5]. Es importante mencionar como el trabajo descrito por Viola y Jones [1] fue el primero en proponer una solución que fuera factible para sistemas que trabajaran en tiempo real, con las propuestas de dicho documento fue posible por primera vez detectar caras de manera muy eficiente en unos cuantos milisegundos. Existen también otras propuestas utilizando otro tipo de características, tal es el caso de las características LBP [6], las cuales en general tienen mejores tiempos de respuesta, pero la robustez de los clasificadores también se ve afectada, aunque investigaciones más recientes han demostrado que no existe mayor diferencia [7]. Existen algunos otros algoritmos que han demostrado ser muy eficientes, aunque el entrenamiento de muchos de ellos puede requerir de muchas horas de procesamiento [8], lo cual también es un factor importante que considerar cuando se busca crear un clasificador, algunos de ellos son las redes neuronales jerárquicas profundas que son un método no supervisado [9].

Entre otros trabajos de mucho interés, también se encuentras aquellos que buscan la detección de peatones. Se han propuesto algoritmos los cuales también emplean características de tipo HAAR que tienen tiempos de respuesta muy aceptables al ser combinado con un extractor de características como es el caso de HOG (histograma de gradientes orientados), el cual ciertamente es más lento, pero al mismo tiempo es más preciso dando como resultado un número bajo de falsos positivos. Esto ha permitido tener un clasificador de peatones bastante robusto [10].

Además de la detección de peatones, uno de los problemas iniciales fue el de detectar otros vehículos, esto debido a la necesidad de asegurar que un vehículo se mantenga a una distancia segura de otros vehículos [11] y así evitar accidentes, es por esto por lo que la detección de vehículos sigue siendo especialmente importante y uno de los tipos objetos en los cuales se sigue concentrando mucha investigación [5].

Hoy en día contamos con una gran variedad de sensores, cámaras que, junto a otros sensores, permiten calcular profundidad en las imágenes, pero estos siguen siendo de un alto costo, por lo cual las imágenes mono-oculares continúan siendo, la base de la visión [12]. Es por esto por lo que la gran mayoría de los trabajos aún en la actualidad siguen utilizando este sensor.

Entre los algoritmos de clasificación más interesantes se encuentra cascade, que en conjunto con algunas de las características antes mencionadas (HAAR o HOG) ha demostrado ser muy eficiente en el reconocimiento de rostros humanos [1] [2] [7], Es una de las razones por la cual se ha decidido optar por la implementación de OpenCV, agregando nuevas características tipo HAAR de modo que estas describan de forma más precisa a un vehículo. La implementación ofrecida por OpenCV ha sido utilizada ampliamente por muchos otros trabajos anteriormente [13] [14].

4. Marco Conceptual

En esta sección se hablará de aquellos conceptos que previamente han sido definidos, y que es necesarios conocerlos antes del planteamiento de este trabajo. Dichos conceptos requieren entender las imágenes digitales, como se almacenan y representan en una computadora. También se explica la transformada *wavelet*, concepto sumamente importante para las características tipo Haar, las cuales

serán explicadas después.

Además, es importante entender cómo se emplean las imágenes integrales para calcular sumas de los valores en cierta región en una imagen en escala de grises, lo cual sirve como base para explicar la implementación de Adaboost y cascade propuestas por Viola & Jones [1]. Por último, se explica la estructura y organización de OpenCV, así como la forma en que el algoritmo de entrenamiento genera los archivos de salida, esto se explica ya que este trabajo se basa completamente en la implementación proporcionada por dicha biblioteca.

4.1. Imágenes digitales y espacio de colores

Una imagen digital es una representación matricial de una imagen, la cual es almacenada en forma de matriz de pixeles (px), dichos pixeles representan la unidad mínima de una imagen, una imagen debe ser representada utilizando un espacio de colores, el cuál básicamente es un modelo matemático abstracto que describe la forma de representación de las imágenes utilizando tuplas de valores, normalmente tres o cuatro valores. Entre los espacios de color más utilizados se encuentras el RGB, CMYK, HSV y HSL. De estos tres, el más usado es el RGB, el cual utiliza una tripleta de valores que representa los niveles de los tres colores primarios rojo, verde y azul para representar cada pixel.

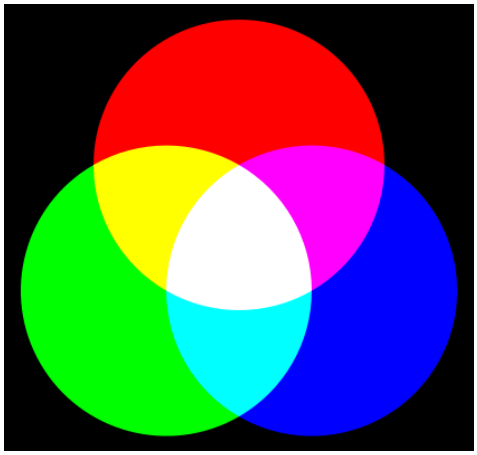


Figura 2: Modelo aditivo de colores en RGB

La Figura 2 muestra el modelo en el que se basa este espacio de colores, el cual mezcla por adición los tres colores primarios y en la Figura 3 se puede observar una imagen conformada por un conjunto de pixeles RGB.

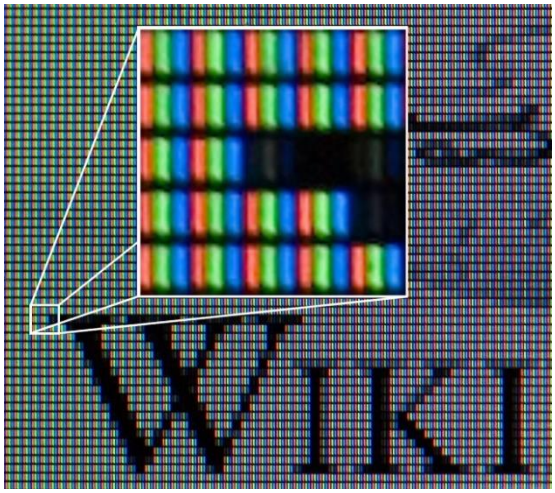


Figura 3: Vista de una imagen formada por pixeles RGB

Las imágenes que se almacenan directamente en forma de matriz se conocen como imágenes matriciales, aunque existen otro tipo de imágenes llamadas imágenes vectoriales, las cuales, en lugar de guardar pixeles, almacenas funciones y descripciones de elementos básicos como líneas, polígonos, arcos, entre otros. Este tipo de imágenes deben ser interpretadas por un programa informático que sea capaz de convertir su contenido en una matriz de pixeles para poder ser desplegada. La Figura 4 muestra un ejemplo de una imagen digital con baja resolución la cual nos permite visualizar cada uno de los pixeles que la conforman. Las imágenes digitalizadas (obtenidas desde mundo real y almacenadas en un archivo) solamente pueden ser almacenadas como imágenes matriciales. Los métodos utilizados para digitalizar imágenes son los *scanners* y las cámaras digitales.

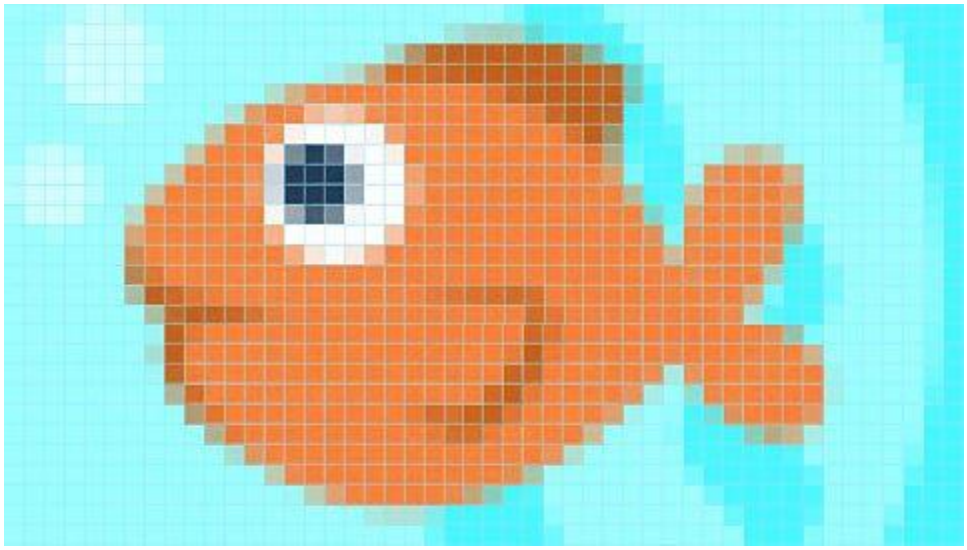


Figura 4: Imagen con pixeles muy visibles

4.2. Wavelet Haar

La transformada *wavelet* es una transformada matemática que representa señales en términos

de traslación y cambio de escala de una función conocida como "*wavelet* madre". Las wavelets son dilatadas cuando $s > 1$ y son contraídos cuando $s < 1$, donde s es la escala. Una gran ventaja que ofrece las *wavelets* es que son un tipo de transformada con ventana, dicha ventana cambia su anchura de forma proporcional al cambio de la frecuencia [15]. Existen muchas funciones para generar la "*wavelet* madre", tal es el caso de las funciones *Daubechies, coiflets*, así como la función *Haar.*

Las funciones *wavelet Haar*, fueron las primeras wavelets conocidas y propuestas por Alfred Haar en 1909. La Figura 5 muestra los valores en forma gráfica para la función Haar. Esta está definida por la siguiente función explicita

$$s(t) = \begin{cases} 1, & 0 < t < 0.5 \\ -1, & 0.5 < t < 1 \\ 0 & en\ otro\ caso \end{cases}.$$

El hecho de que las funciones Haar sean discretas ha incentivado su uso en el procesamiento de imágenes gracias a que, permite detectar variaciones abruptas en la señal. De esta forma, se pueden utilizar como características que ayuden a detectar objetos en una imagen.

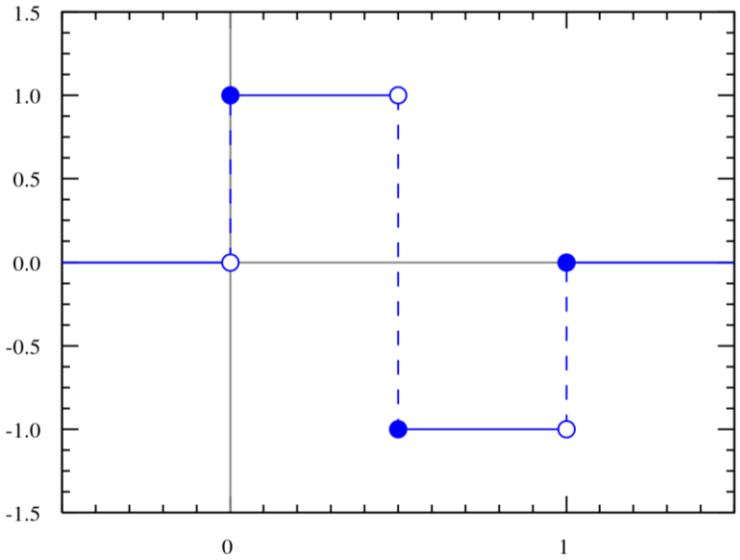


Figura 5: La función wavelet Haar

4.3. Características

Una característica es un elemento que contiene información de una aplicación o contexto específico como parte de su construcción, de modo que esta pueda describir componentes predefinidos antes de la creación de dicha característica. Esto permite vincular una estructura que en ocasiones puede llegar a ser compleja con un descriptor simple que ayuda a identificarlo. Así, dada la generalidad conceptual, la selección de características para cada problema específico se convierte en una tarea primaria. En el campo de la visión computacional se han propuesto una gran cantidad de tipos de características, las cuales varían en el modelo y la estructura de las mismas. Entre los dos tipos de características más utilizadas en el campo de la detección de objetos se encuentran las características tipo Haar [1] y las características *Local Binary Pattern* (LBP) [16].

4.3.1. Características tipo Haar

La idea de las características tipo Haar fueron originalmente pensadas en el trabajo de Papageorgiou y Oren [17], posteriormente fueron adaptadas y formalizadas por Viola & Jones [1]. Una de las principales ventajas de utilizarlas es que permite analizar pixeles de forma directa. Para esto, se utilizan rectángulos para definir la estructura de estas características. La principal motivación para utilizar rectángulos, en lugar de pixeles, es que éstos funcionan como un mecanismo que en sí mismo permite codificar datos que pueden llegar a ser complicados de aprender. Los elementos más importantes de un objeto son aquellos que permiten diferenciar dicho objeto de otros. Es de mucho interés encontrar dichos diferenciadores en aplicaciones donde las imágenes a analizar sean grandes. Dichos diferenciadores permitirán detectar rápidamente si una región de la imagen contiene algún objeto de interés.

Las características del tipo Haar a pesar de ser muy simples, son calculadas de forma muy eficiente gracias al concepto de imagen integral, de la cual hablaremos más adelante en detalle. De modo que, como en la mayoría de los trabajos previos, podemos darnos cuenta de que siempre se presentan tres tipos:

- Basadas en dos rectángulos. Estas principalmente sirven para distinguir zonas oscuras adyacentes a zonas más claras y pueden ser horizontales o verticales.
- Basadas en tres rectángulos. Sirven para encontrar zonas con componente de alta frecuencia. Ya que, se caracteriza por que el elemento del centro presenta alto contraste respecto a los elementos de los lados.
- Basadas en cuatro rectángulos. Estas se utilizan mayormente para buscar diagonales y otras componentes de la imagen con esta orientación.

La Figura 6 muestra la representación gráfica de estas características.

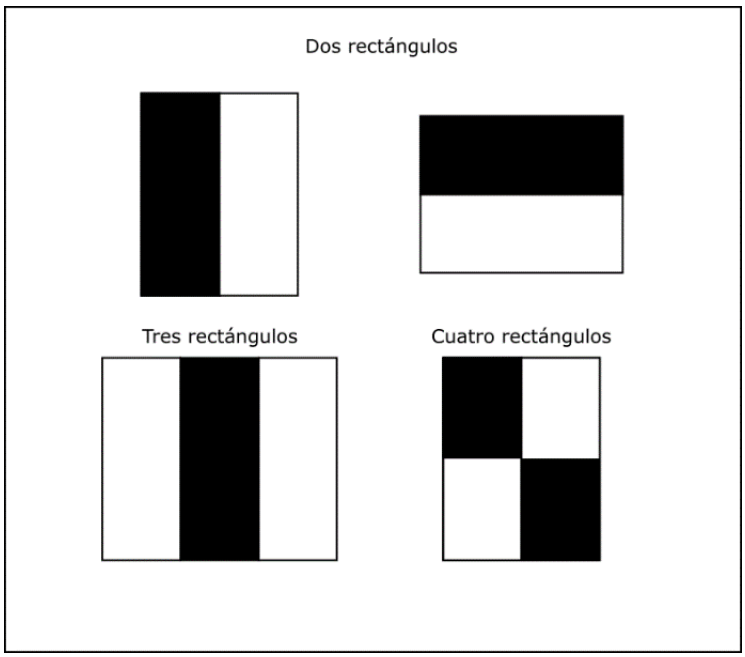


Figura 6: Características tipo HAAR

4.4. Imagen integral

Este método, proporciona una solución muy eficiente para computar rectángulos de características, la idea consiste en crear una "imagen integral", $ii(\ \ )$, de una imagen original, $i(\ \ )$. Esta imagen integral sólo se calcula una vez, y cuando se requiere conocer el valor de la imagen para un punto (x, y), esta contenga la suma de los pixeles hacia arriba y hacia la izquierda (incluyendo el punto x, y). Dicha consulta se hace de una forma bastante eficiente con las ecuaciones que adelante se describen. Este concepto se encuentra representada en la Figura 7.

La imagen integral ha de ser calculada de la siguiente forma:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

donde $ii(x, y)$ es la imagen integral e $i(x', y')$ es la imagen original. Esta expresión también puede ser vista con las siguientes recurrencias

$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

donde $s(x, y)$ hacer referencia la suma acumulada de cada fila. Además, para terminar con la recursividad son necesarias las siguientes condiciones, $s(x, -1) = 0$ e $ii(-1, y) = 0$. En el "Apéndice A: Ejemplo de Implementación de la Imagen Integral en C++**"** se adjunta una posible solución para la implementación, pero de forma iterativa.

Figura 7: Área sumada en una imagen integral

Para ejemplificar este proceso, considere la matriz relativamente pequeña de enteros que se muestra en la Figura 8. Considere también que cada valor representa la intensidad en escala de grises de un pixel.

5	10	4	2	1	6	8	4	7	2	7	7
6	7	5	3	7	10	6	3	3	8	4	5
5	4	2	6	2	2	2	9	1	1	8	4
6	9	5	7	8	5	6	9	3	6	1	2
4	10	2	2	9	2	5	4	4	1	10	9
4	6	9	9	9	8	5	7	8	2	8	10
9	8	10	6	7	2	8	3	3	5	7	9
4	1	4	7	3	6	2	7	5	6	7	1
9	10	8	5	6	4	3	3	10	6	10	3
6	2	10	3	6	1	8	3	7	2	10	5
1	10	1	4	9	3	7	2	3	9	2	10
4	1	1	2	3	9	1	2	7	4	8	7

Figura 8: Representación de imagen en valores enteros y escala de grises

La imagen integral asociada, utilizando las ecuaciones antes descritas, se muestra en la Figura 9.

5	15	19	21	22	28	36	40	47	49	56	63
11	28	37	42	50	66	80	87	97	107	118	130
16	37	48	59	69	87	103	119	130	141	160	176
22	52	68	86	104	127	149	174	188	205	225	243
26	66	84	104	131	156	183	212	230	248	278	305
30	76	103	132	168	201	233	269	295	315	353	390
39	93	130	165	208	243	283	322	351	376	421	467
43	98	139	181	227	268	310	356	390	421	473	520
52	117	166	213	265	310	355	404	448	485	547	597
58	125	184	234	292	338	391	443	494	533	605	660
59	136	196	250	317	366	426	480	534	582	656	721
63	141	202	258	328	386	447	503	564	616	698	770

Figura 9: Imagen integral calculada

Como se puede apreciar, el cálculo de la imagen integral es muy simple, pero es un mecanismo muy eficiente ya que sirve como una tabla de búsqueda con la cual se pueden calcular la suma de un sub-rectángulo de forma casi instantánea. De hecho, con el uso de imágenes integrales es posible computar la suma de cualquier rectángulo dentro de la imagen con cuatro puntos de referencia. Por lo tanto, para calcular la suma de dos rectángulos se requieren ocho puntos, pero si se tratara de dos rectángulos adyacentes (características de dos rectángulos), en realidad sólo se necesitan seis puntos. De esto se concluye que para tres rectángulos adyacentes (características de tres rectángulos) sólo se necesitan ocho puntos. Entonces, para las características de cuatro rectángulos (en forma de matriz de $2\ x\ 2$) sólo se requieren nueve puntos. Como se muestra en la Figura 10, el valor de la imagen integral

en el punto 1 es la suma de los pixeles en el rectángulo A, el valor en el punto 2 es la suma de los rectángulos $A + B$, en el punto 3 es la suma de los rectángulos $A + C$ y, finalmente, el punto 4 es la suma $A + B + C + D$. Por lo tanto, para calcular la suma de los pixeles en el rectángulo D se debe utilizar la simple fórmula $4 + 1 - (2 + 3)$.

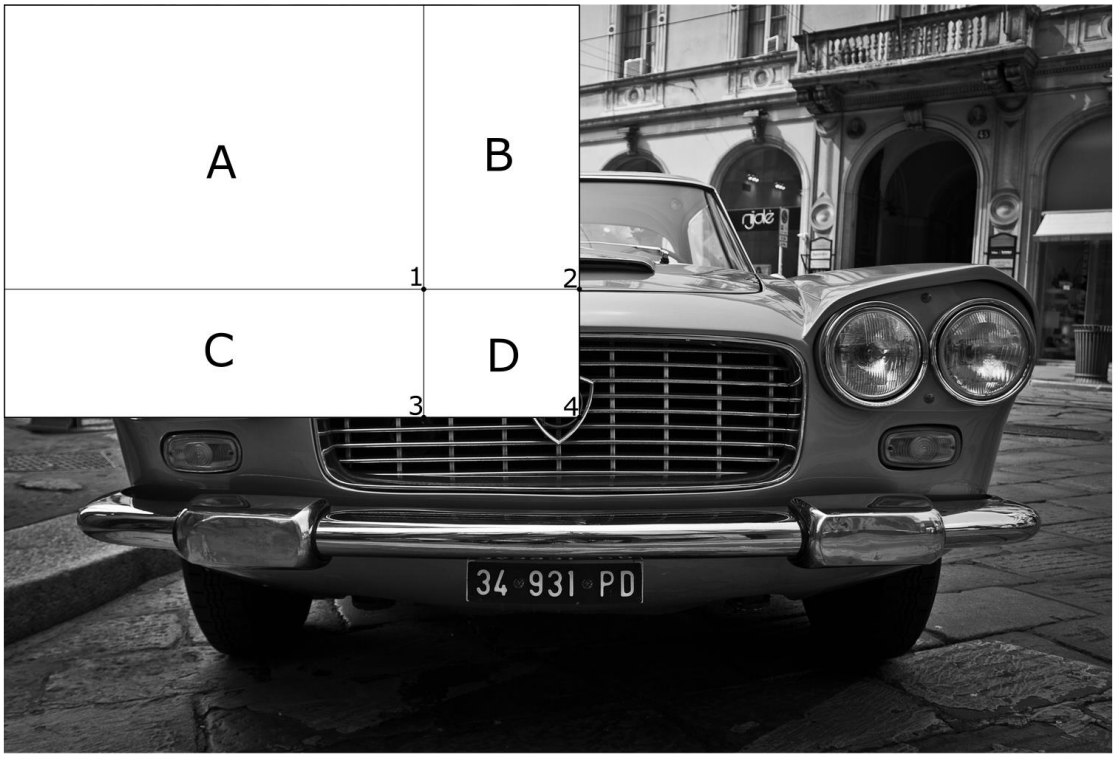


Figura 10: Calcular suma en cualquier sub rectángulo

4.5. Árboles de decisión

Los árboles de decisión son un modelo de predicción que usa como estructura un árbol N-ario para representar flujos de decisión basados en condiciones previamente definidas. Estos son muy útiles para establecer las diferentes opciones en un escenario de acuerdo con ciertos datos de entrada. En ocasiones es posible combinar un árbol dado con otros árboles para generar un nuevo modelo más robusto y con mayor poder de discriminación de datos. En la Figura 11 se muestra un ejemplo de un árbol de decisión para determinar el tipo de flor (de un conjunto de 3) de acuerdo con el ancho y largo del pétalo. Como se puede apreciar, los nodos en forma de óvalo son las condiciones, los vértices son los resultados de las condiciones y los nodos en forma de rectángulos son las decisiones tomadas. De modo que para evaluar dicho árbol se parte de la raíz y se evalúa la condición, de acuerdo con la evaluación se elige un vértice para seguir la evaluación. Este proceso re realiza de forma recursiva hasta llevar a un nodo de decisión (nodo hoja).

Existen árboles de decisión de un solo nivel, a los que se le conoce como *stub*. Estos se emplean con otro método que permita encadenarlos de alguna manera y así crear un modelo más grande utilizando un conjunto de reglas que no solo depende del árbol de decisión. Este es el caso de los métodos de conjunto (*ensemble*).

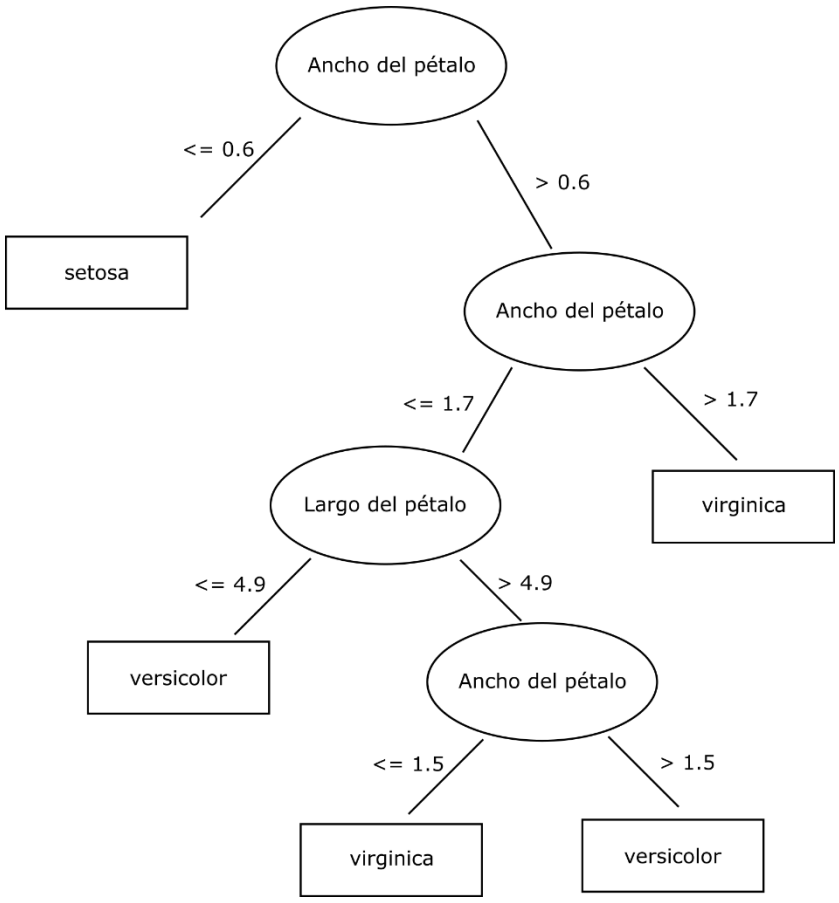


Figura 11: Ejemplo de un árbol de decisión

4.5.1. Métodos combinados de aprendizaje (ensemble)

Los algoritmos combinados forman parte de los algoritmos de aprendizaje supervisado, los cuales realizan una búsqueda sobre un conjunto de hipótesis para encontrar aquella que sea capaz de hacer predicciones sobre los datos y el problema particular. La complejidad de recorrer todas y cada una de estas hipótesis es un problema no polinomial (NP) y, de hecho, recorrer toda ellas no asegura encontrar una solución óptima.

La idea de un método combinado consiste en considerar varias hipótesis de forma simultánea, para así formar una nueva hipótesis que se comporte mejor que cada una de las hipótesis que la componen. Es un hecho que llevar a cabo una combinación de varias hipótesis incrementa los costes computacionales. Debido a esto es común utilizar algoritmos rápidos o que requieran poco poder computacional como hipótesis base, el más común de ellos son los árboles de decisión.

Los algoritmos de *ensemble* más comunes son *bagging* [18] y *boosting* [19]. Los algoritmos de *boosting* trabajan sobre el conjunto de datos de entrada de entrenamiento completo, entonces se manipulan los pesos de los datos para generar modelos. Es importante mencionar que en cada paso del algoritmo se va incrementando el peso de aquellos objetos que han sido clasificados de forma incorrecta, de modo que estos objetos mal clasificados se convierten en los más importantes en las próximas iteraciones.

4.6. Algoritmos de aprendizaje y clasificación

Durante las últimas décadas, se han desarrollo una gran variedad de algoritmos que permiten crear sistemas cada vez más “inteligentes” y capaces de solucionar problemas más complejos. Esto ha sido en gran parte gracias a una rama de la inteligencia artificial conocida como aprendizaje automático o *machine learning* por su nombre en inglés. El objetivo de dichos algoritmos es ser capaces de generalizar comportamientos y patrones sobre un conjunto de datos, es decir, dado un conjunto de datos que comparten ciertas características o comportamientos, el algoritmo puede generar el “conocimiento” suficiente para describir dicho conjunto de datos. En otras palabras, dichos algoritmos permiten obtener una generalización a partir de un conjunto de instancias particulares.

4.6.1. Adaboost

El algoritmo de aprendizaje Adaboost es una variante del tipo de algoritmos tipo *boosting*. Adaboost fue propuesto originalmente por Yoav and Robert [19], pero posteriormente Viola y Jones [1] describieron una nueva implementación. Dicha implementación fue utilizada exitosamente como una etapa de su sistema de detección de objetos. Este sistema planteaba tener como entrada un conjunto de características predefinidas y un conjunto de imágenes positivas (que incluyera el objeto a clasificar) y negativas (que no incluyera el objeto a clasificar) para realizar un entrenamiento del clasificador y este pudiera detectar el objeto para el que fue entrenado en nuevas imágenes que lo contengan. La idea original detrás de este algoritmo consiste en incrementar el poder de clasificación de un algoritmo de aprendizaje simple o también llamado “clasificador débil”.

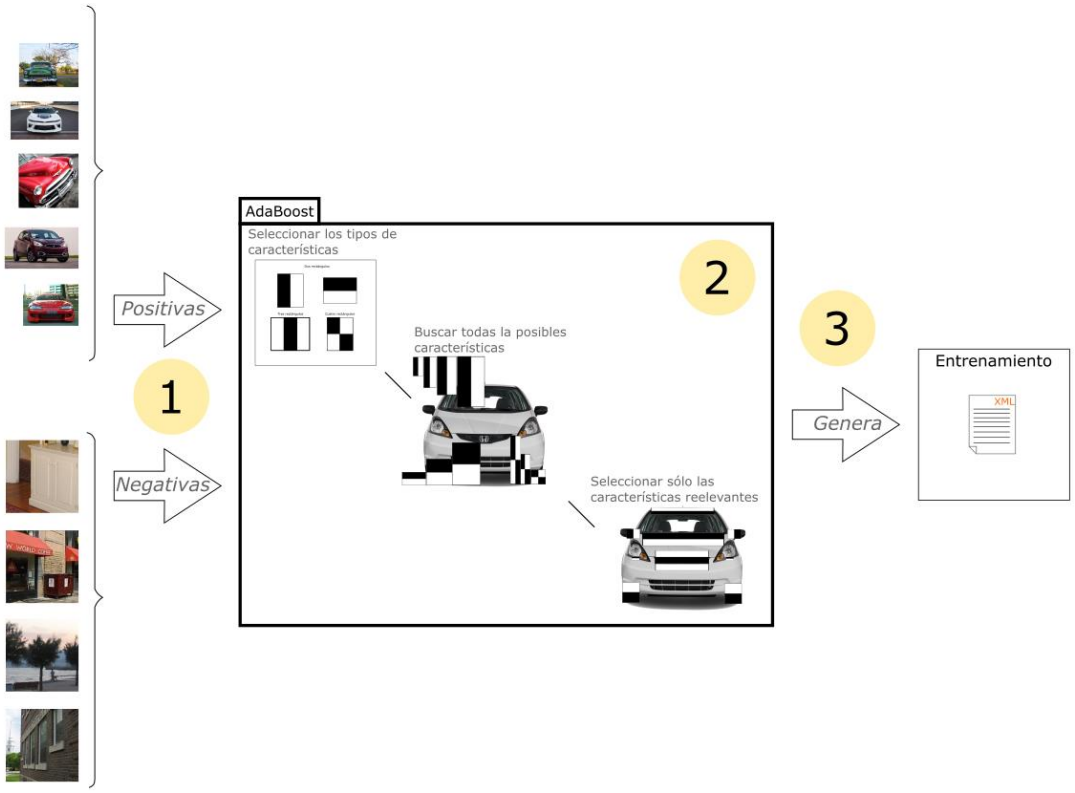


Figura 12: Proceso del algoritmo Adaboost

El proceso completo del algoritmo se muestra en la Figura 12, donde se puede observar cómo se inicia a partir de un conjunto de imágenes de entrada positivas y negativas. Una vez con las imágenes de entrada, se requiere conocer el tipo de características a utilizar, estas previamente han sido definidas en el código del algoritmo. Posteriormente se generan todas las posibles combinaciones de características al ir incrementando tanto el tamaño en x y y así como su posición dentro de la imagen. Por último, la parte central de *Adaboost*, debe ser capaz de seleccionar sólo aquellas características que sirvan para describir un vehículo y a la vez no sirvan para describir algo que no sea un vehículo. Por último, una vez terminado el entrenamiento, este es almacenado en un archivo que posteriormente será leído por el algoritmo *cascade*, que será descrito en la siguiente sección del documento.

El número de rectángulos de características que originalmente se diseñaron para el trabajo de Viola y Jones era mayor a 180,000. Es evidente que probar este conjunto por cada sub ventana en una imagen es una tarea imposible. De esto surgió la idea de que, de ese conjunto de características, sólo un pequeño sub conjunto es suficiente para que al combinarlo pueda formarse un clasificador funcional requiriendo un poder de procesamiento mucho más bajo. El gran problema que esto presenta es que encontrar ese pequeño sub conjunto es una tarea difícil.

Para el presente trabajo se ha utilizado como clasificador débil los arboles de decisión de un solo nivel, también conocidos como *stub*. Estos son rápidos de evaluar y, por si solos, son mejores que una aproximación completamente aleatoria. Entonces, la tarea de estos clasificadores es determinar en cada etapa del entrenamiento cual es el rectángulo de característica que mejor separa las muestras positivas de las negativas. Por cada característica, el clasificador débil calcula el umbral correcto para su función de clasificación. La siguiente expresión describe el funcionamiento de un clasificador débil.

$$h_j(x) = \begin{cases} 1 & if\ p_j f_j(x) < p_j \emptyset_j \\ 0 & en\ otro\ caso \end{cases}$$

donde:

$h_j(x) : Es\ un\ clasificador\ débil$

$f_j : Es\ una\ característica$

$\emptyset_j : Es\ el\ umbral$

$p_j : Indica\ el\ signo\ en\ la\ desigualdad$

$x : Es\ una\ sub\ ventana\ dentro\ de\ una\ imagen\ de\ 24\ x\ 24\ pixeles.$

En la práctica, se ha demostrado que durante las primeras etapas del proceso de entrenamiento se tienen errores de clasificación entre 0.1 y 0.3 a diferencia de los que se encuentran en etapas posteriores que tienen errores entre 0.3 y 0.5. El procedimiento de Adaboost se describe con detalle en la Figura 13.

Entradas

$(x_1, y_1), \dots, (x_n y_n) : donde\ x_i\ representa\ una\ imagen\ y$
$y_i = 0,1\ indica\ \ si\ es\ para\ negativas\ o$
$positivas\ respectivamente$

Inicialización

$Inicializar\ pesos\ w_{1,j} = \frac{1}{2m}, \frac{1}{2l}\ para\ y_i = 0, 1\ respectivamente$
$donde\ m\ y\ l\ son\ el\ número\ de\ imagenes\ negativas\ y\ positivas\ respectivamente$

$para\ t = 1 \ldots T$:

1. $Normalizar\ pesos$:

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

2. $Por\ cada\ característica\ j\ entrenar\ un\ clasificador\ h_j. El\ error\ se\ evalúa\ con\ respecto\ a\ w_t$

$$\in_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$$

3. $Obtener\ el\ clasificador\ h_t\ con\ error\ más\ bajo\ \in_t$
4. $Actualizar\ los\ pesos$

$$w_{t+1,\ i} = w_{t,i} \beta_t^{1-e_i}$$

$donde$:

$e_i = 0\ si\ el\ ejemplo\ x_i\ es\ clasificado\ correctamente$

$e_i = 1\ en\ caso\ contrario$

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$$

$El\ clasificador\ final\ es$

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & en\ otro\ caso \end{cases}$$

$donde$:

$$\alpha_t = \log \frac{1}{\beta_t}$$

Figura 13: Procedimiento de Adaboost

4.6.2. Cascade

Ahora se describe la implementación del algoritmo *cascade* basados en lo ya descrito por Paul Viola y Michael Jones [1]. Conceptualmente, este algoritmo utiliza un conjunto de características tipo HAAR [10], las cuales previamente fueron elegidas por el algoritmo adaboost durante la etapa de entrenamiento, para describir al objeto a clasificar. El flujo completo del proceso se muestra en el diagrama de la Figura 14, donde se muestran los componentes más importantes y los objetos de entrada y salida.

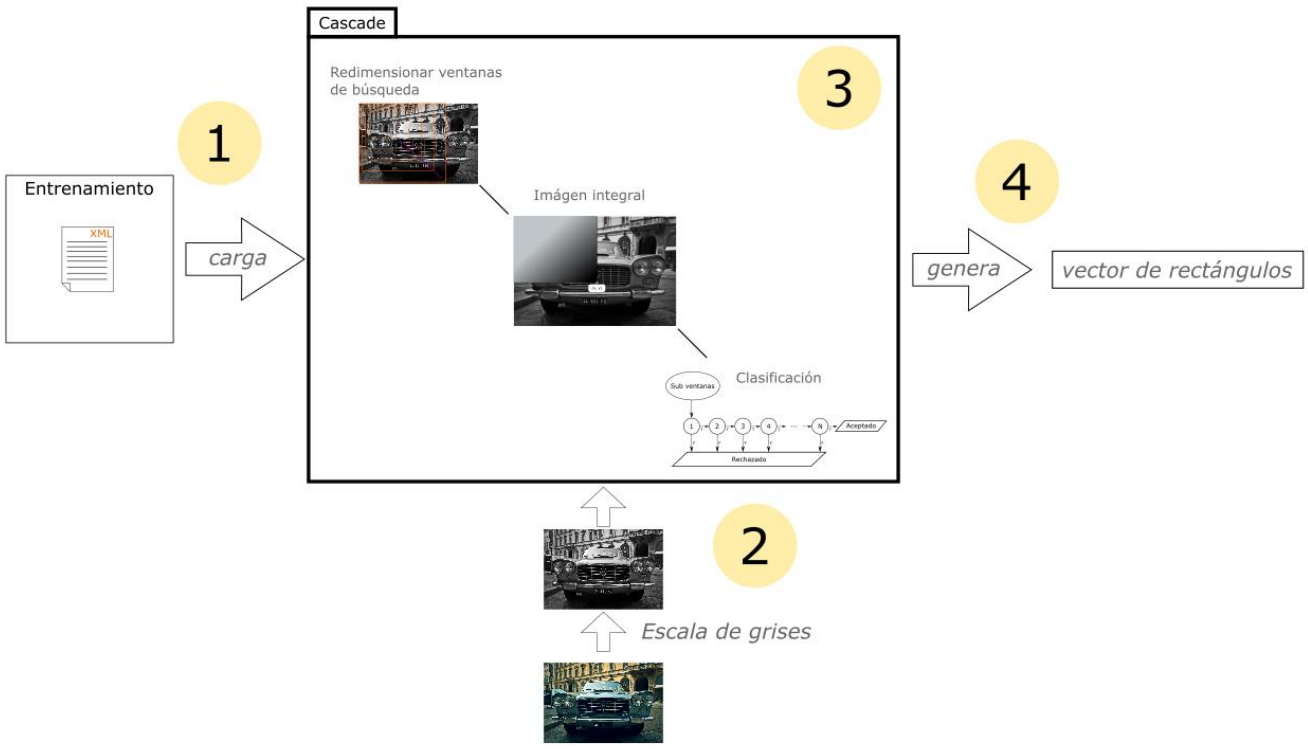


Figura 14: Proceso del algoritmo Cascade

La detección de objetos con el algoritmo *cascade* se realiza de la siguiente forma: como primer paso es necesario cargar los datos que adaboost previamente ha generado, estos datos contienen las características que se utilizarán para realizar la detección. Después, se cargan la o las imágenes de entrada, ya sea una sola imagen fija o de una secuencia de imágenes.

Una vez que el algoritmo tiene la primera imagen, se realiza un proceso de inicialización de las estructuras internas del clasificador. Es importante mencionar que en el caso de tener como entrada una secuencia de imágenes, esta inicialización se realiza únicamente cuando llega la primera imagen. Como parte de proceso de inicialización es necesario calcular el tamaño de búsqueda de cada ventana, de modo que la primera ventana de búsqueda es exactamente del mismo tamaño que la ventana utilizada durante el entrenamiento. Entonces, las ventanas siguientes van creciendo proporcionalmente de acuerdo con un factor hasta que el tamaño de la ventana supera el tamaño de la imagen de entrada, es en este momento donde se ha encontrado la última ventana. Es decir, si elegimos un factor de redimensionado de 1.1, y el tamaño de la ventana de entrenamiento fue de $20\ x\ 20$ pixeles, entonces es necesario calcular las siguientes sub ventanas, por lo tanto la segunda sub ventana tendría un tamaño de $22\ x\ 22\ px$ $(redondea(20 * 1.1)\ y\ redondea(20 * 1.1))$ y si continuamos con el proceso la siguiente sub ventana debería medir $24\ x\ 24\ px$ $(redondea(20 * 1.21)\ y\ redondea(20 * 1.21))$, por lo tanto la siguiente sería $27\ x\ 27\ px$ $(redondea(20 * 1.331)\ y\ redondea(20 * 1.331))$ y así sucesivamente hasta que se llegue a un tamaño de ventana mayor o igual que el tamaño de la imagen que se quiere analizar. Este último no debe ser utilizado debido a que supera el tamaño de la imagen, ya sea en anchura o altura. La Figura 15 ejemplifica como crece el tamaño de las sub ventanas hasta superar el tamaño de la imagen de una ventana inicial arbitraria.

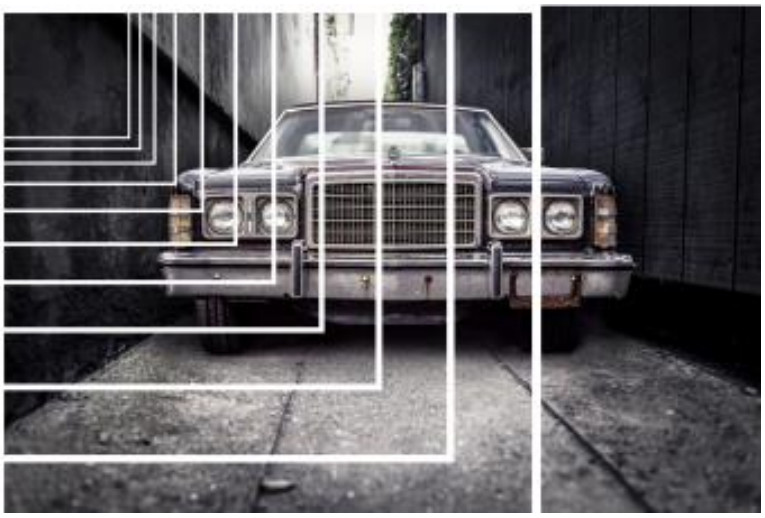


Figura 15: Redimensionado sub ventanas de búsqueda

Es importante mencionar que, aunque en la etapa de la generación de sub ventanas el tamaño de las sub ventanas nos sirve para determinar cuándo debe detenerse la generación de las sub ventanas, la información relevante a almacenar no es el tamaño de la sub ventana como tal, sino el factor de redimensionado, ya que este factor es utilizado para calcular los tamaños de los espacios de almacenamiento o *buffers* para el cálculo de la imagen integral.

La Tabla 1 muestra los factores de redimensionado generados para una imagen de $640\ x\ 480$ px y una ventana inicial un tamaño de $20\ x\ 20\ px$. Como se puede observar, la columna número 35, marcada de color rojo, indica que en realidad esa iteración será ignorada, esto es debido a que el tamaño de la sub ventana para ese factor de redimensionado sería de $511\ x\ 511\ px$ y ya que la imagen a analizar es solamente de $640\ x\ 480\ px$, se ha superado el tamaño de la altura ($511\ >\ 480$).

Iteración	Factor de redimensionado	Tamaño de ventana (pixeles)
1	1	20 x 20
2	1.1	22 x 22
3	1.21	24 x 24
…	…	…
33	21.1138	422 x 422
34	23.2252	465 x 465
35	25.5477	511 x 511

Tabla 1: Muestra los distintos tamaños de las sub ventanas

Una vez calculadas las sub ventanas de búsqueda, el siguiente paso es calcular la imagen integral para la imagen de entrada. Este proceso se realiza con el procedimiento anteriormente mencionado, de forma que una vez calculada la imagen integral se puede comenzar la detección. Entonces, por cada uno de los tamaños de las sub ventanas generadas anteriormente se realiza un barrido desde la esquina

superior izquierda hasta la inferior derecha, con un desplazamiento de un pixel en cada nueva búsqueda.

La Figura 16 muestra como por cada sub ventana de la imagen integral se hace la comprobación de las características de cada etapa. Entonces, si el resultado de la evaluación de estas características supera el umbral establecido para esta etapa, esta se evalúa como positiva y como negativa en caso contrario. Este proceso continúa hasta que la última etapa acepta la ventana o hasta que cualquiera de las etapas rechaza la ventana. Al final, este proceso se convierte en una decisión de aceptación o de rechazo de la sub ventana.

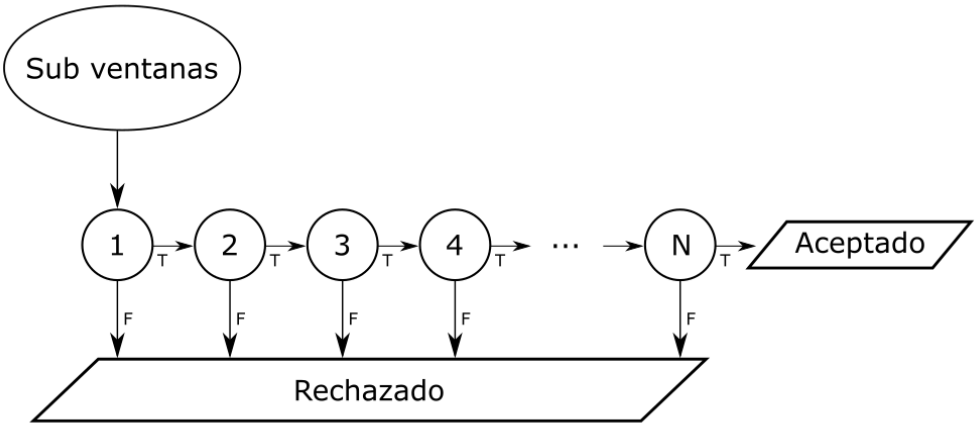


Figura 16: Detección utilizando cascade y las características tipo Haar

4.7. Estructura de OpenCV

OpenCV (Open Computer Vision) es un conjunto de librerías de acceso libre para el procesamiento de imágenes y aplicaciones de visión computacional. OpenCV se distribuye bajo una licencia *open source* muy similar a la licencia BSD (*Berkeley Software Distribution*) aunque con algunas pequeñas modificaciones. Siempre es posible descargar la última versión desde el repositorio de GitHub https://github.com/opencv/opencv.

Debido al tamaño de la biblioteca, y por contener un muy diverso conjunto de algoritmos y herramientas, se abordarán a detalle solamente los elementos importantes para este trabajo. Entre estos se encuentran aquellos empleados en detección de objetos, entrenamiento con Adaboost, y ejemplos para utilizar la funcionalidad de detección de objetos.

OpenCV divide su estructura de archivos de acuerdo con la categoría de elementos que contiene, pero sobre todo debemos poner más atención en las carpetas “apps”, “include”, “modules” y “samples”. En la Figura 17 se muestran las carpetas principales del proyecto y se encierran aquellas que tienen más relevancia.

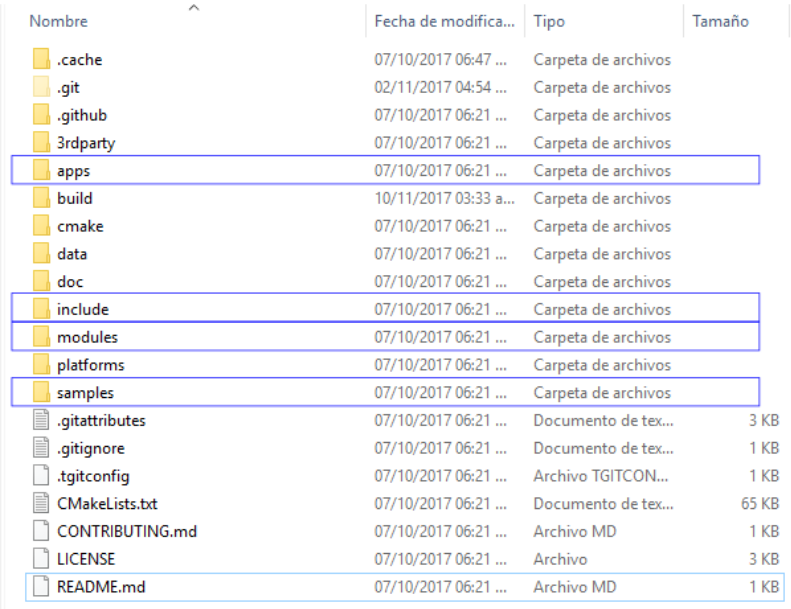


Figura 17: Estructura de archivos de OpenCV

- Modules

La carpeta de “Modules” contiene el núcleo de OpenCV, esto incluye todas las clases básicas, por ejemplo la clase cv::Mat, cv::Rect, cv::Size, además de las funciones cv::cvtColor, cv::imread, sólo por mencionar algunas. Además de esto, esta carpeta contiene el código con la implementación de los algoritmos de OpenCV, entre ellos se encuentra *cascade* en su implementación con Haar y Hog. También se encuentran muchos otros algoritmos como redes neuronales, procesamiento de imágenes, filtros, y otros componentes que forman el núcleo de OpenCV.

- Include

La carpeta “Include” contiene los archivos de cabecera de OpenCV. Debido a que esta librería está creada principalmente en lenguaje C++, todos los elementos básicos de él, como definiciones de clases, estructuras, enumeraciones, funciones, entre otras, se describen en esta carpeta. Esto es necesario ya que cuando se requiere escribir algún software utilizando las funcionalidades de OpenCV, debemos utilizar dichos archivos de cabecera para que el compilador sea capaz de determinar la estructura de los objetos que sea crean. Por lo tanto, esta es una de las carpetas que se le deben indicar al compilador para ser agregadas a las carpetas que contienen archivos de cabecera.

- Apps

Esta carpeta es muy importante para el uso de Adaboost en OpenCV, ya que aquí se encuentran varias herramientas necesarias para dicho algoritmo. Una de ellas es la herramienta para hacer anotaciones en las imágenes positivas ya que con dicha herramienta podemos seleccionar las coordenadas donde se encuentra realmente el objeto de interés en nuestras imágenes de entrada. Además, se encuentra la herramienta de creación de muestras la cual permite crear el archivo “.vec”. Este contiene las imágenes positivas exclusivamente, pero almacenadas en un solo archivo binario. Por

último, se encuentra la herramienta para realizar el entrenamiento Adaboost, por lo tanto, podemos decir que la implementación completa de Adaboost se encuentra en este directorio, de modo que la modificación de los tipos de características debe realizarse aquí.

- Samples

Esta carpeta contiene un gran número de ejemplos de las funcionalidades de OpenCV. Contiene ejemplos en C++, Java, Python e incluso para Android. Esta carpeta es de gran utilidad como primera aproximación para hacer funcionar algún algoritmo. En nuestro caso ha servido para analizar el ejemplo de cómo utilizar cascade, específicamente para el uso de la función "detectMultiScale".

4.7.1. Estructura del archivo xml de entrenamiento

Una de las partes principales del sistema es el archivo de entrenamiento. Por ello, es importante entender la estructura y el significado de cada uno de los elementos que lo componen. La Figura 18 muestra la estructura completa del archivo. Se puede observar que los campos "stages" y "features" se encuentran plegados ya que se explican con más detalle posteriormente.

La mayoría de los campos se explican por si solos, aunque vale la pena explicar algunos de ellos con un poco más de detalle. Por ejemplo, entre ellos se encuentra "stageType" que indica el tipo de entrenamiento a utilizar. La librería permite utilizar otros algoritmos además de los de *boosting*, sin embargo, para nuestro caso siempre definiremos este valor como "BOOST". Otro campo importante es "featureType", el cual indica el tipo de características utilizadas. En nuestra aplicación hemos empleado características tipo Haar, pero esta herramienta nos permite realizar entrenamientos con características del tipo LBP. Los siguientes campos son "height" y "width" que indican el tamaño de las imágenes positivas que se utilizaron durante el entrenamiento, de modo que este es el tamaño mínimo que el detector podrá utilizar. Los campos dentro de la sección "stageParams" indican los parámetros que fueron especificados para la herramienta de entrenamiento y se puede encontrar su descripción en [20]. Por último, el campo "stageNum" que indica con cuantas etapas se realizó el entrenamiento.

```
<?xml version="1.0"?>
- <opencv_storage>
  - <cascade>
      <stageType>BOOST</stageType>
      <featureType>HAAR</featureType>
      <height>24</height>
      <width>24</width>
    - <stageParams>
        <boostType>GAB</boostType>
        <minHitRate>9.9500000476837158e-01</minHitRate>
        <maxFalseAlarm>5.0000000000000000e-01</maxFalseAlarm>
        <weightTrimRate>9.4999999999999996e-01</weightTrimRate>
        <maxDepth>1</maxDepth>
        <maxWeakCount>100</maxWeakCount>
      </stageParams>
    - <featureParams>
        <maxCatCount>0</maxCatCount>
        <featSize>1</featSize>
        <mode>ALL</mode>
      </featureParams>
      <stageNum>30</stageNum>
    + <stages>
    + <features>
    </cascade>
</opencv_storage>
```

Figura 18: Estructura completa del archivo de entrenamiento

Ahora, en la Figura 19 se muestra un ejemplo de cómo se estructuran las etapas dentro del archivo de entrenamiento. El primer campo que considerar es “maxWeakCount” que indica la cantidad de clasificadores débiles que contiene la etapa. Recordemos que un clasificador débil está representado por una característica tipo Haar que se evalúa en un árbol de decisión tipo *stub*, el cual a su vez genera un resultado que se evaluará por un umbral. El valor “stageThreshold” es de hecho el valor del umbral que debe alcanzarse en conjunto con todos los clasificadores débiles para poder declarar la etapa como positiva. Recapitulando, en la Figura 16 se muestra como cada etapa debe ser evaluada como positiva para poder continuar con el proceso hasta llegar a la última etapa, y en caso de que alguna de ellas se evalúe como negativa, entonces la ventana actual se descarta.

```
- <stages>
        <!-- stage 0 -->
   - <_>
        <maxWeakCount>10</maxWeakCount>
        <stageThreshold>-1.1428397893905640e+00</stageThreshold>
      - <weakClassifiers>
         - <_>
              <internalNodes> 0 -1 229 -2.4443954229354858e-02</internalNodes>
              <leafValues> 8.2016348838806152e-01 -3.1434491276741028e-01</leafValues>
           </_>
         - <_>
              <internalNodes> 0 -1 881 3.6203805357217789e-02</internalNodes>
              <leafValues> 4.3552702665328979e-01 -6.8621689081192017e-01</leafValues>
           </_>
         - <_>
              <internalNodes> 0 -1 1142 2.8973307460546494e-02</internalNodes>
              <leafValues> -3.2711839675903320e-01 5.7417917251586914e-01</leafValues>
           </_>
         - <_>
              <internalNodes> 0 -1 28 -1.3194229453802109e-02</internalNodes>
              <leafValues> 5.4838293790817261e-01 -2.5290349125862122e-01</leafValues>
           </_>
```

Figura 19: Representación de las etapas

Después, se encuentran las definiciones de los clasificadores débiles, que estos cuentan con dos campos: “internalNodes” y “leafValues”. El primero sirve para representar los árboles de decisión donde el primer y segundo valor (0 y -1) significan el nodo hijo izquierdo y derecho respectivamente del nodo. Aunque parece extraño y redundante el hecho de que en todos los clasificadores aparecen siempre estos dos valores, esto sucede ya que la implementación del árbol de decisión ha sido reutilizada del árbol de decisión ya implementado anteriormente en OpenCV, de modo que estos dos valores hacen referencia a dicho árbol de decisión de un solo nivel.

El tercer valor dentro del campo “internalNodes” hace referencia al índice de la característica que debe ser utilizada. Dichas características se mencionan en la sección posterior “features” como se ve en la Figura 20. Por último, el cuarto valor es el umbral de decisión a considerar durante la evaluación para cada clasificador.

Los valores del campo “leafValues” hacen referencia a los valores hoja para el *stub,* de modo que este par de valores son el resultado de la evaluación del árbol de decisión, el primero corresponde al nodo izquierdo y el segundo al nodo derecho.

La última parte del archivo de entrenamiento se compone por las características en sí, estas se presentan en la Figura 20. En ella se puede apreciar que la estructura de las mismas es muy simple, ya que se compone de los campos “rect” y “tilted”. El campo “rect” representa los rectángulos que definen cada característica, de modo que para una característica que contenga dos rectángulos, esta sólo

contendrá dos entradas en su respectivo campo “rect”, para las características que requieran tres campos, contendrá tres entradas y así sucesivamente. Cada uno de estos rectángulos re representa por cuatro valores que indican la posición del vértice superior izquierdo (x, y), así como la anchura y altura del rectángulo, respectivamente. Por último, el quinto parámetro es el peso que tiene dicho rectángulo, el cual sirve para establecer el color que es representado por cada rectángulo.

```
- <features>
  - <_>
    - <rects>
        <_> 0 0 4 4 1.</_>
        <_> 2 1 2 2 -4.</_>
      </rects>
      <tilted>0</tilted>
    </_>
  - <_>
    - <rects>
        <_> 0 0 2 18 -1.</_>
        <_> 0 0 1 9 2.</_>
        <_> 1 9 1 9 2.</_>
      </rects>
      <tilted>0</tilted>
    </_>
```

Figura 20: Representación de las características

Entonces, de acuerdo a la Figura 20, la representación gráfica de la primer característica es la que se puede observar en la Figura 21, donde el rectángulo con los valores $0, 0, 4, 4$ es en realidad el cuadro de $4\ x\ 4$ que contiene los *pixeles* negros. Después se superpone el rectángulo con los valores $2, 1, 2, 2$ que contiene los *pixeles* blancos.

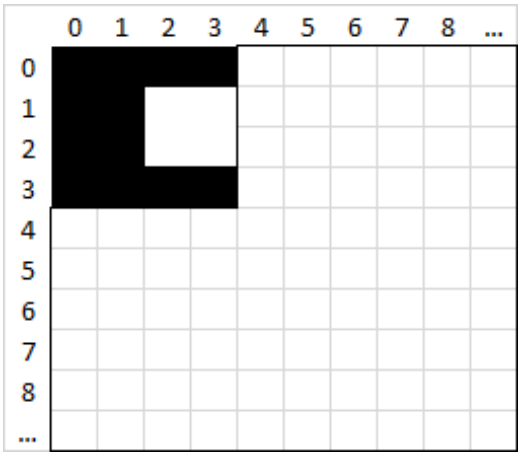


Figura 21: Característica 1 según el archivo de entrenamiento

En la Figura 22 se observa gráficamente la segunda característica. Esta se compone por tres rectángulos, el primero es el rectángulo con los valores $0, 0, 2, 18$ que corresponde al rectángulo exterior, el cual tiene una dimensión de $2\ x\ 18$. Después, aparece el segundo rectángulo con valores

$0, 0, 1, 9$ que representa el rectángulo negro que se encuentra en la parte superior izquierda. Por último, el rectángulo $1, 9, 1, 9$ que representa el rectángulo negro que se encuentra en la parte inferior derecha. Con lo que queda totalmente definida la característica deseada. Con este procedimiento es posible definir nuevas características adecuadas para otro tipo de aplicaciones.

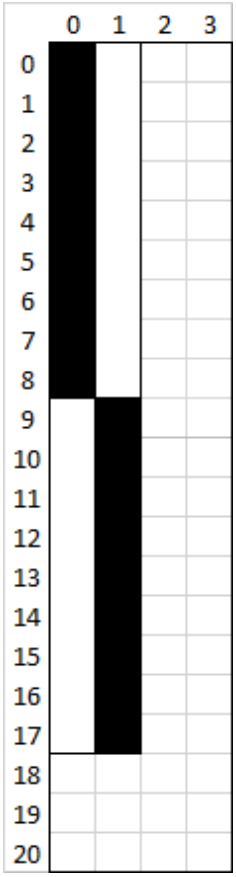


Figura 22: Característica 2 según el archivo de entrenamiento

5. Metodología de Trabajo

El desarrollo de este trabajo ha sido llevado a cabo con el objetivo principal de obtener un detector que sea capaz de conseguir mejores resultados utilizando adaboost y cascade. Para esto, ha sido necesario entender a profundidad ambos algoritmos y la información que ellos requieren de tal forma que sea posible realizar entrenamientos adecuados para el objeto de interés, en este caso vehículos automotrices.

De acuerdo con lo anterior, los elementos más importantes durante el desarrollo han sido la elección de imágenes de entrada suficientes y adecuadas. Después, la elección de características que sean capaces de mejorar la capacidad de detección del sistema, dichas características serán explicadas con más detalle en la siguiente sección. Por último, para verificar que verdaderamente dichas características demuestran beneficiar al sistema mejorando su capacidad de detección. De acuerdo con el problema a validar es importante utilizar alguna estrategia de evaluación que facilite la validación del detector.

5.1. Preparación de los datos y elección de parámetros

Uno de los pasos más importantes para el proceso de entrenamiento, es la selección de un buen conjunto de datos de entrenamiento. Este debe estar compuesto por un conjunto de imágenes positivas y otro de imágenes negativas. Para este trabajo se han tomado un total de 2861 imágenes positivas obtenidas del conjunto de datos "The LabelMe-12-50k dataset". Cuenta con un total de 3829 imágenes de entrenamiento, las cuales fueron filtradas para obtener un conjunto de datos acorde al entrenamiento esperado. Una de las técnicas utilizadas para incrementar el número de imágenes positivas fue aplicar la operación de espejo (*flop*) a todas las imágenes, creando 2861 imágenes más, dando un total de 5722 imágenes.

Para la selección de imágenes negativas, se utilizó un subconjunto de imágenes también del *data set* anteriormente mencionado, el cual fue de 2000 imágenes. Además, este sub conjunto se combinó con un set de imágenes obtenidas de Google compuesto por 1000 imágenes de entre $550x550\ px$ hasta $1512x1512\ px$. Dicho sub conjunto se conformó exclusivamente de imágenes con contenido de la vialidad, donde no aparecía ningún vehículo.

Una vez seleccionado el conjunto de datos, se hizo el etiquetado de las imágenes positivas para generar el archivo ".info" y el archivo "bg" de acuerdo con la documentación de OpenCV [20]. El archivo ".info" fue generado con la ayuda de la herramienta para hacer anotaciones que OpenCV provee "opencv_annotation", la cual permite ir imagen por imagen en forma gráfica y elegir las coordenadas exactas donde se encuentra nuestro objeto a detectar. Este proceso fue realizado con las 2861 imágenes originales y para las otras 2861 imágenes se invirtieron las coordenadas en base a las ya obtenidas.

Después de haber generado el archivo ".info", se procedió a generar el archivo de muestras con la aplicación también proporcionada por OpenCV "opencv_createsamples" [20], la cual toma todas las imágenes definidas en el archivo ".info" y crea un archivo binario (archivo ".vec") con el contenido de ellas pero solamente con la sección de cada imagen que contiene un vehículo y con las dimensiones elegidas para realizar el entrenamiento, en este caso $24x24\ px$.

Una vez creados los archivos "bg" y ".vec", el siguiente paso fue elegir los parámetros de entrenamiento a utilizar con la herramienta de OpenCV "opencv_traincascade". Para esto se ha seguido la serie de recomendaciones descrita en el capítulo "Parameter selection when training an object model" en [21]. Basándose en los valores antes mencionados, los parámetros más relevantes para el entrenamiento elegidos se describen en la Tabla 2, dichos paramentos han sido probados y propuestos después de pruebas, demostrando ser los más eficientes.

Nombre del parámetro	Valor
data	*data*
numPos	*3000*
numNeg	*2000*
numStages	*20 - 40*
bg	*Bg.txt*
vec	*posImages.vec*
featureType	*HAAR*
minHitRate	*0.995*

Tabla 2: Parámetros usados en el entrenamiento

5.2. Estrategias de evaluación

Estas permiten validar el desempeño funcional del objetivo de un sistema. Es una herramienta de gran importancia, ya que, determinar la precisión de un algoritmo siempre permite detectar puntos débiles en dicho sistema y de esta forma hacer propuestas cada vez con mejores resultados. Para esto es común utilizar estrategias de validación, entre ellas, la más simple es *holdout*, además de las conocidas como estrategias de validación cruzada. Las cuales consisten en repetir la validación sobre diferentes particiones y calcular la media aritmética.

5.2.1. Método Holdout

Es la estrategia más simple de validación, consiste en separar el conjunto de datos en dos subconjuntos, uno llamado conjunto de entrenamiento y otro conjunto de pruebas. Es importante notar que el algoritmo de entrenamiento solamente trabaja con los datos del conjunto de entrenamiento lo cual hace que el conjunto de pruebas sea completamente desconocido para dicho algoritmo. Como ventaja de esta estrategia se tiene que es muy rápido de realizar y no es muy demandante en cuanto a poder de cómputo. Su gran desventaja es la dependencia que existe entre los resultados obtenidos y la elección de los conjuntos de datos, es decir, conjuntos de datos distintos generan resultados muy distintos. Esta será la estrategia de evaluación utilizada en este trabajo.

5.2.2. Validación cruzada de k iteraciones (K-fold cross validation)

En esta estrategia, los datos de entrada se dividen en k subconjuntos. Después, uno de estos subconjuntos es usado como datos de prueba y el resto como datos de entrenamiento. Este proceso se realiza k veces cambiando el conjunto de pruebas en cada una de ellas. Al final se obtiene la media aritmética de los resultados obtenidos en cada iteración. La gran ventaja de esta estrategia es su gran precisión gracias a las combinaciones creadas, aunque su gran desventaja es el tiempo requerido para realizar el análisis debido a que el entrenamiento y la evaluación deben realizarse k veces.

5.2.3. Validación cruzada dejando uno fuera (Leave one out cross validation)

Esta estrategia consiste en separar los datos de modo que para cada iteración se obtenga únicamente una muestra para realizar pruebas y todos los demás para el entrenamiento. Al final, la evaluación como tal se arrastra en el valor del error. Esta estrategia permite obtener más precisión en los resultados, pero tiene la desventaja de ser más costoso en cuanto a capacidad de cómputo, incluso más que la estrategia anterior, ya que deben realizarse tantas iteraciones como datos de entrada.

6. Esquema De Clasificación Visual De Objetos Propuesto

El esquema de clasificación propuesto para la detección de vehículos en imágenes tiene la misma estructura que utiliza OpenCV. Es decir, utiliza el algoritmo AdaBoost (en su versión GAB – *Gentle AdaBoost*) para el entrenamiento del clasificador utilizando una secuencia de imágenes de entrada positivas (que contiene el objeto a detectar) y negativas (que no lo contiene). Luego, utiliza el resultado del entrenamiento, condensado en un archivo *xml*, para realizar la detección por medio del algoritmo *Cascade*. La interacción de estos dos elementos del sistema se puede apreciar en la Figura 23 donde se describe el flujo de la información desde que se ingresa el conjunto de imágenes positivas y negativas.

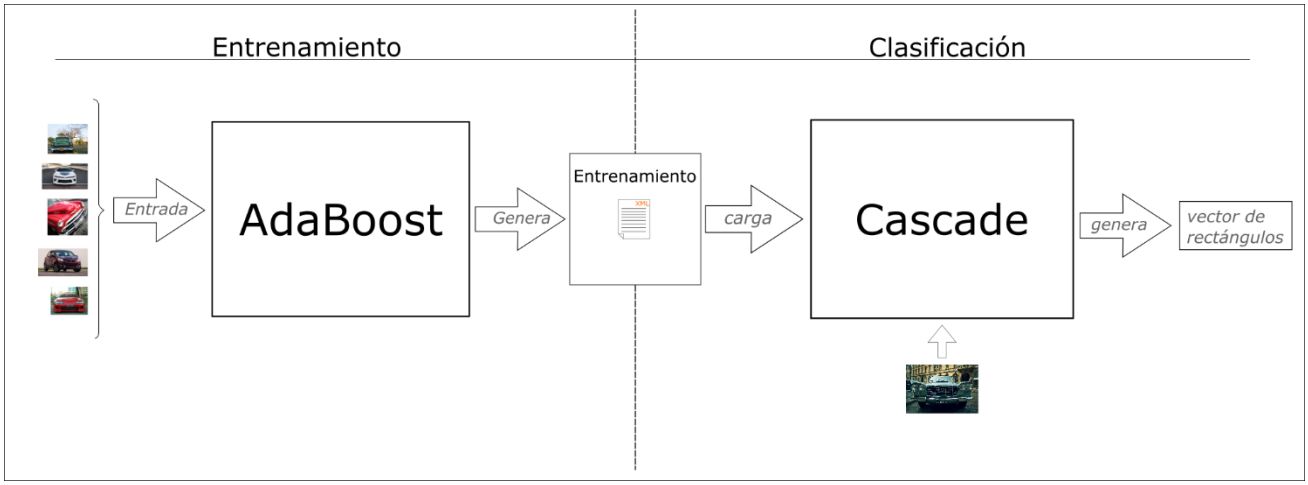


Figura 23: Esquema completo del sistema

Este esquema basado en estos dos algoritmos, ambos implementados en la biblioteca de OpenCV, ha mostrado ser muy versátil y se han hecho aplicaciones de detección de diversos patrones, entre ellos el de vehículos automotrices, con resultados diversos. De la experiencia de utilizar este esquema de detección para vehículos, surge la idea de proponer un nuevo conjunto de características que permitan mejorar el desempeño del sistema de detección para vehículos automotrices. Es evidente que estas determinan fundamentalmente el éxito y desempeño del sistema de detección ya que impactan directamente la etapa de entrenamiento y, consecuentemente, la de clasificación.

Para esto, es necesario analizar la implementación de las características existentes en la librería de OpenCV e identificar la forma de implementar nuevas características que se consideren necesarias para mejorar el desempeño del sistema de detección global.

Las características por defecto en la librería de OpenCV se muestran en la Figura 24. La principal aplicación para la que se plantearon estas características es la detección de rostros. Es por esto por lo que se diseñaron originalmente para caracterizar líneas horizontales, verticales y líneas inclinadas (*tilted)*. Para poder mejorar la capacidad de descripción del conjunto de características para detección de vehículos, se optó por mantener el conjunto original de características de OpenCV y agregar las que permitieran describir patrones que describen más adecuadamente un vehículo automotriz. Para poder utilizar el marco de trabajo de la librería de OpenCV, fue necesario entender cómo se expresan las características en la librería y cómo interactúan las mismas con los demás elementos que la conforman.

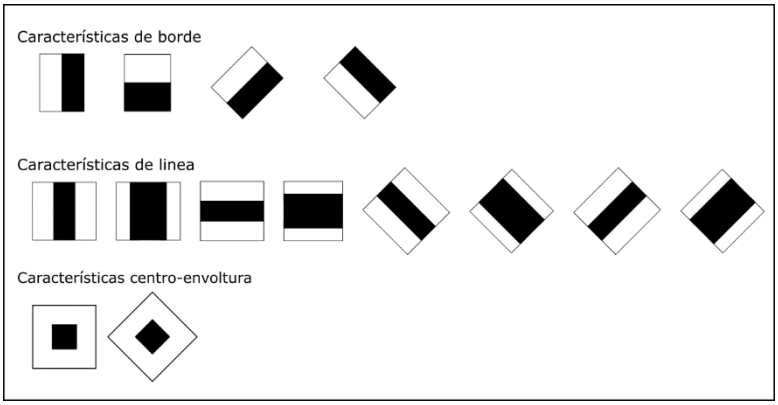


Figura 24: Características tipo Haar implementadas en OpenCV

6.1. Características propuestas

Las nuevas características, que se agregaron a las ya existentes en OpenCV, se pueden clasificar en 3 grupos: las características tipo esquina, las características tipo “T” y, por último, las características tipo semi-borde. Todas orientadas a describir de mejor manera los patrones que suelen componer a un vehículo automotriz en una imagen.

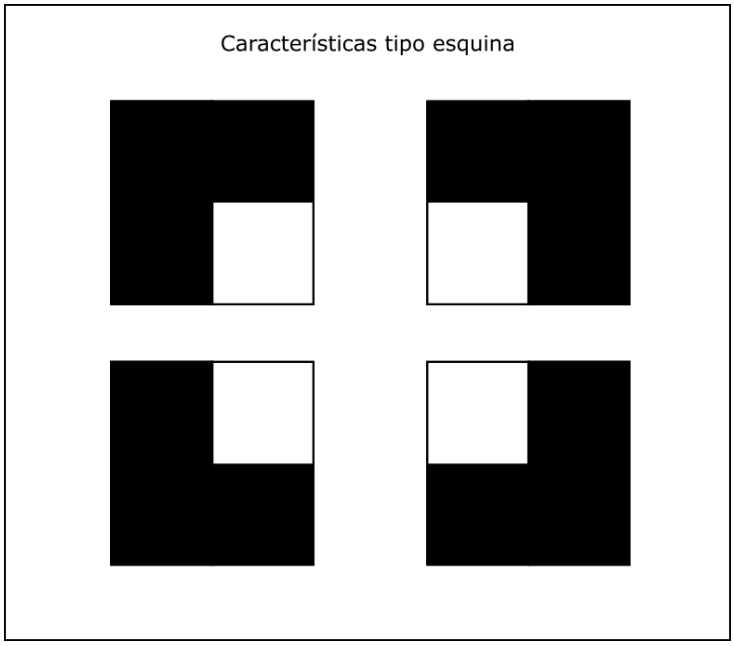


Figura 25: Características tipo HAAR para describir esquinas

6.1.1. Características tipo esquina

Las características de tipo esquina son un complemento a las características originalmente propuestas como líneas verticales y horizontales. La ventaja del tipo esquina es que ayudan a potencializar la detección de zonas donde las esquinas sean un elemento importante y muy visible a simple vista en un vehículo. Se planteó el definir una característica por cada esquina que define la

máscara cuadrada base sobre la que ésta se expresa. La Figura 25 muestra la representación gráfica de dichas características y la Figura 26 muestra las zonas de una imagen de un vehículo que aumentarían la probabilidad de ser mejor descritas por las mismas.

Figura 26: Características de esquina sobre un automóvil

6.1.2. Características tipo T

Este tipo de características ayudan a describir pares de líneas perpendiculares, las cuales, aunque no es muy común encontrar en un vehículo, pueden aportar información que ayuden a detectar vehículos cuya imagen fue capturada desde una perspectiva lateral. La Figura 27 muestra las características en forma de T que fueron propuestas para este trabajo y en la Figura 28 se muestra una fotografía de un vehículo con las características tipo T que permiten identificar algunas zonas particulares del vehículo.

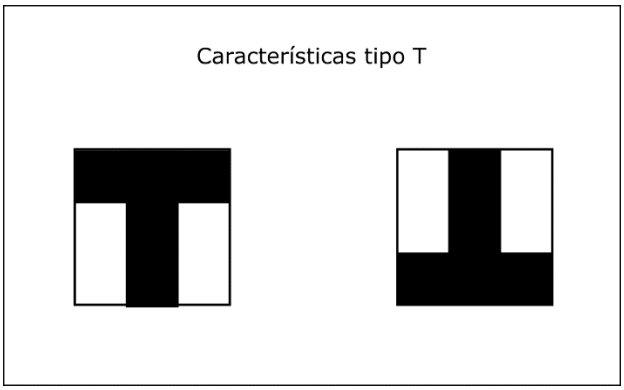


Figura 27: Características tipo T

Figura 28: Características tipo T en un vehículo real

6.1.3. Características tipo semi-borde

El último tipo de conjunto de características propuestas es el de semi-borde, las cuales ayudan a detectar algunas partes del vehículo que se caractericen por bordes semi cuadrados. Existen muchas zonas de la imagen de un vehículo que pueden describirse fuertemente por este tipo de características. Una gran ventaja de utilizar estas características es su simplicidad, ya que dichas características se pueden representar únicamente por 2 rectángulos. La Figura 29 muestra las cuatro variantes de este tipo de características que fueron implementadas en el sistema propuesto.

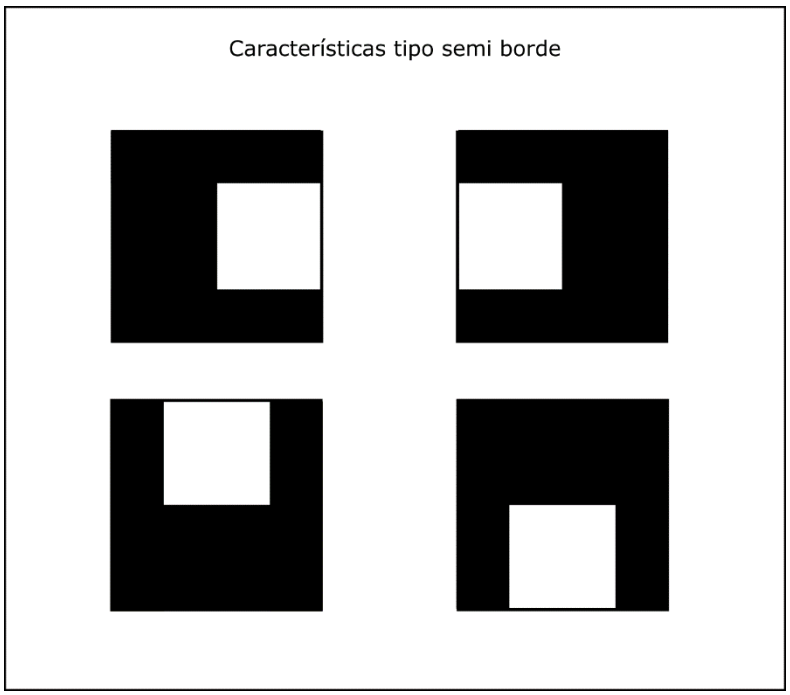


Figura 29: Características tipo semi borde

Como se muestra en la Figura 30, los elementos más visibles para ser representados por estas características son las llantas desde una perspectiva lateral, los retrovisores y las luces del vehículo.

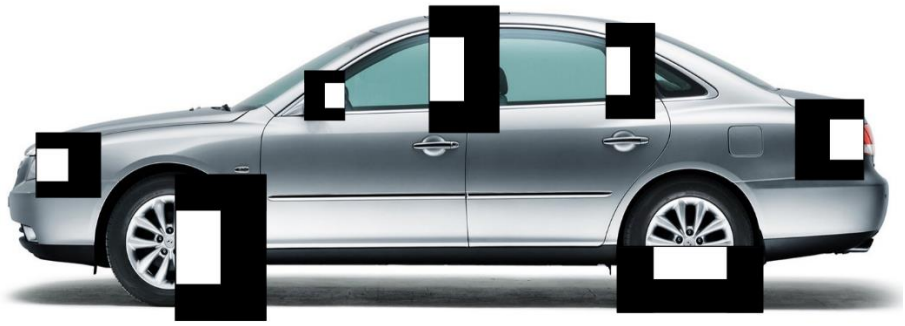


Figura 30: Características tipo semi borde en un vehículo real

6.2. Implementación de las características

El primer paso en la implementación de nuevas características, que serán integradas al marco de trabajo de OpenCV, es entender como OpenCV implementa y utiliza las ya existentes en un esquema de detección. Como ya ha sido mencionado en la sección "Estructura de OpenCV", la carpeta "apps" es un elemento clave para entender este proceso.

Del conjunto de aplicaciones que pueden ser encontradas en esta carpeta, ***traincascade*** es la que realiza el entrenamiento con el algoritmo AdaBoost, y es aquí mismo donde inicialmente se calculan las características con las cuales se realiza el entrenamiento. Dentro de este proyecto se encuentra el archivo ***traincascade.cpp***, el cual es el punto de entrada para la aplicación, ya que contiene la función principal y es aquí donde se hace el mapeo de los parámetros introducidos desde la línea de comandos a las variables que configuran la ejecución. Al final de este archivo, el proceso de entrenamiento es delegado a un objeto del tipo ***CvCascadeClassifier***. Dicho objeto ejecuta la plantilla de entrenamiento, esto implica ejecutar las etapas seleccionadas y, por cada una de ellas, delegar el entrenamiento a un objeto evaluador que previamente es creado de acuerdo con el tipo de características a implementar. En este caso se utiliza un objeto del tipo ***CvHaarEvaluator***.

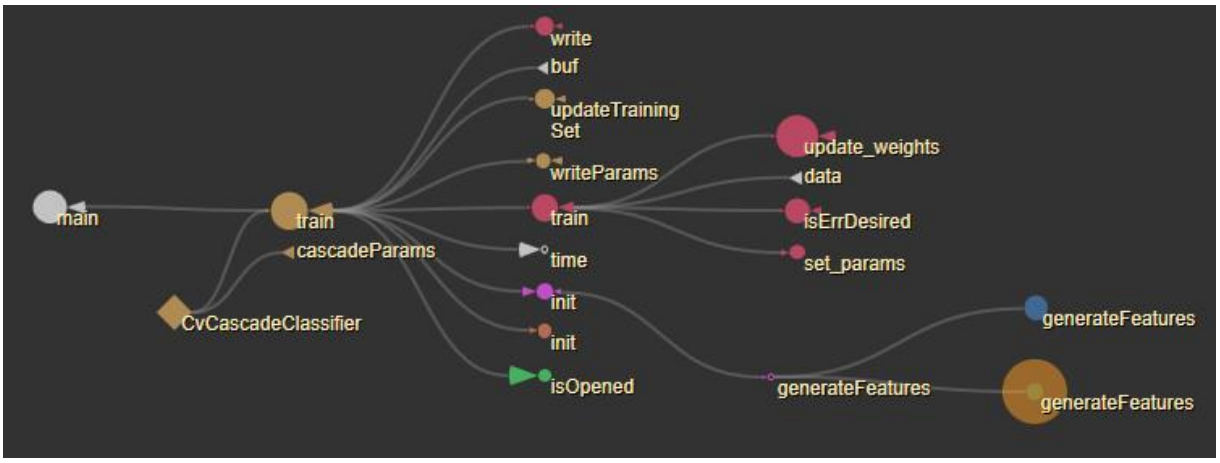


Figura 31: Flujo de llamadas hasta la generación de características

Este objeto evaluador es, en realidad, nuestro objeto de interés, ya que durante su inicialización genera todas las posibles combinaciones de características tipo Haar que soporta. Esto se logra variando

el tamaño de las ventanas, así como la posición de las mismas. Dicho universo de posibles características es calculado usando el método ***generateFeatures***. La Figura 31 muestra el flujo de ejecución de la aplicación hasta la llamada del método antes mencionado.

6.3. Pruebas y comparación con las características por defecto en OpenCV

La validación del detector se ha llevado a cabo utilizando como metodología la estrategia *holdout*. La razón para utilizar esta estrategia ha sido el poder comparar de forma simple y efectiva dos esquemas de detección que utilizan los mismos algoritmos, las mismas imágenes de entrada y los mismos parámetros de entrenamiento, pero que consideran diferentes conjuntos de características de entrenamiento y donde se varía el número de etapas de entrenamiento.

Para la validación se han utilizado 500 imágenes positivas, es decir, que contienen un vehículo en ella. Dichas imágenes son completamente ajenas al conjunto de imágenes utilizadas durante el entrenamiento y que fueron etiquetadas previamente para indicar las coordenadas donde se localizaban los vehículos.

6.3.1. Proceso de validación de cada imagen

Para realizar la validación del sistema se optó por, primeramente, automatizar la validación para cada imagen de prueba. De esta forma, es posible repetir el proceso con todas las imágenes y determinar la cantidad de imágenes detectadas correctamente, así como la cantidad de imágenes no detectadas o aquellas donde se detectaron falsos positivos. Los argumentos de entrada para este proceso automático eran la ruta de la carpeta donde se encontraban almacenadas las imágenes y las coordenadas del vehículo dentro de cada imagen. Para acceder a la información de los pixeles de las imágenes se utilizó la función ***imread***. Dicha imagen sería utilizada como parámetro de entrada de cascade, a través del método ***detectMultiScale***.

Después de realizar la detección, el valor de retorno es un vector de rectángulos. Estos rectángulos indican la posición de un vehículo dentro de la imagen. El siguiente paso es la comprobación del resultado. La forma de realizarla considera los siguientes pasos:

- Comparar el área del rectángulo esperado contra el área del rectángulo detectado. las áreas no deberían variar en más de 0.5 del área del rectángulo esperado.
- Comprobar la coordenada superior izquierda de ambos rectángulos. Estas coordenadas no deberían estar más alejadas de 0.25 del valor de la anchura o la altura del rectángulo esperado.
- Comprobar el ancho y algo del rectángulo detectado. Estos valores no deberían variar más de 0.5 del valor del rectángulo esperado al compararlo contra el rectángulo esperado.

Después de realizar las validaciones anteriores, se asegura que el rectángulo obtenido por el detector está dentro de un rango razonable para considerarlo similar al rectángulo esperado. En otro caso, pueden ocurrir dos cosas: la primera es que el detector haya regresado más de un rectángulo, entonces se realiza el proceso anterior para los rectángulos restantes. La segunda es que el rectángulo detectado sea el único, y en este caso la imagen es marcada como “no detectada” (falso negativo).

6.3.2. Validación del sistema

Como se describió, el primer objetivo es ser capaz de evaluar si una detección es correcta o no. Después de esto, el proceso de validación del sistema se describe a continuación.

Uno de los puntos a considerar es el número de etapas con las cuales el clasificador debe ser entrenado para entregar los mejores resultados, de modo que se ha decidido realizar entrenamientos

con los valores 20, 22, 24, 26, 28, 30, 32, 34, 36, 38 y 40 para el número de etapas. El proceso completo para validar el sistema se describe en la Figura 32.

Variables:
detecciones←0
falsosPositivos←0
falsosNegativos←0

Desde i = 20 hasta i = 40
    Entrenar sistema con i etapas
    Por cada imagen img para validación
        Validar imagen img utilizando el detector de i etapas
        Si se detecta sólo un objeto en la imagen y el objeto corresponde al esperado
            detecciones←detecciones + 1
        si no, si el objeto corresponde al esperado y se detecta más de un objeto
            detecciones←detecciones + 1
            falsosPositivos←falsosPositivos + 1
        si no
            falsosNegativos←falsosNegativos + 1

Figura 32: Proceso de validación

6.4. Resultados encontrados

Después de realizar el procedimiento de validación, se obtuvieron resultados tanto para el detector entrenado con las características propuestas y el entrenado con las características originales de OpenCV. Tales resultados se pueden apreciar en la Figura 33 y la Figura 34. En dichas figuras se puede apreciar que los resultados obtenidos con 20, 22, 24 y 26 etapas de entrenamiento tienen un desempeño bajo. De hecho, es a partir de la etapa 28 donde el detector comienza a ofrecer mejores resultados. Se puede observar que la etapa 30 es la que ofrece mejores resultados en cuanto a detecciones correctas, pero aún el número de falsos negativos es alto.

La relación inversamente proporcional entre el número de etapas de entrenamiento y el número de falsos positivos del detector es evidente en todos los experimentos. De igual forma, el número de imágenes detectadas de forma correcta también comienza a descender. Este efecto es conocido como sobre entrenamiento, lo cual indica que el sistema se ha entrenado a un punto donde el algoritmo de entrenamiento ha intentado ajustar demasiado las características y sus umbrales de detección. Por ello, dicho algoritmo comienza a rechazar imágenes aun cuando ésta contiene vehículos en ellas. Esto hace que la detección se haga cada vez más difícil para *Cascade*. De esto se puede decir que el número de etapas optimas se encuentra entre 30 y 34 para nuestro sistema.

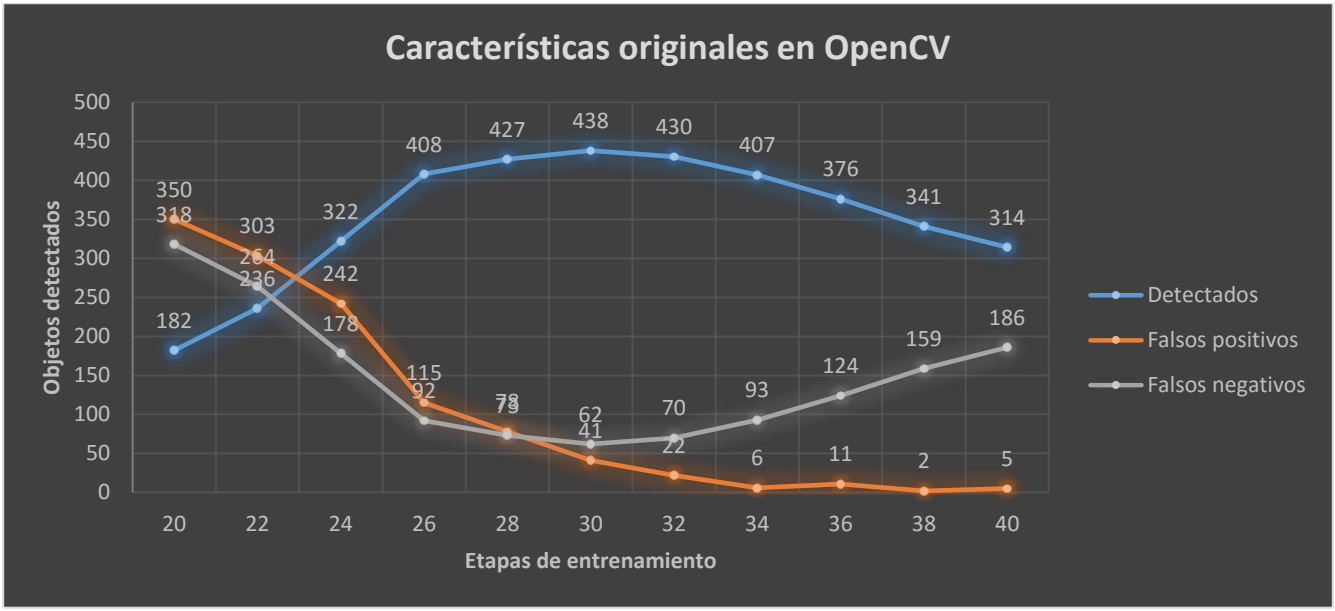


Figura 33: Grafica de las 10 iteraciones usando las características de OpenCV

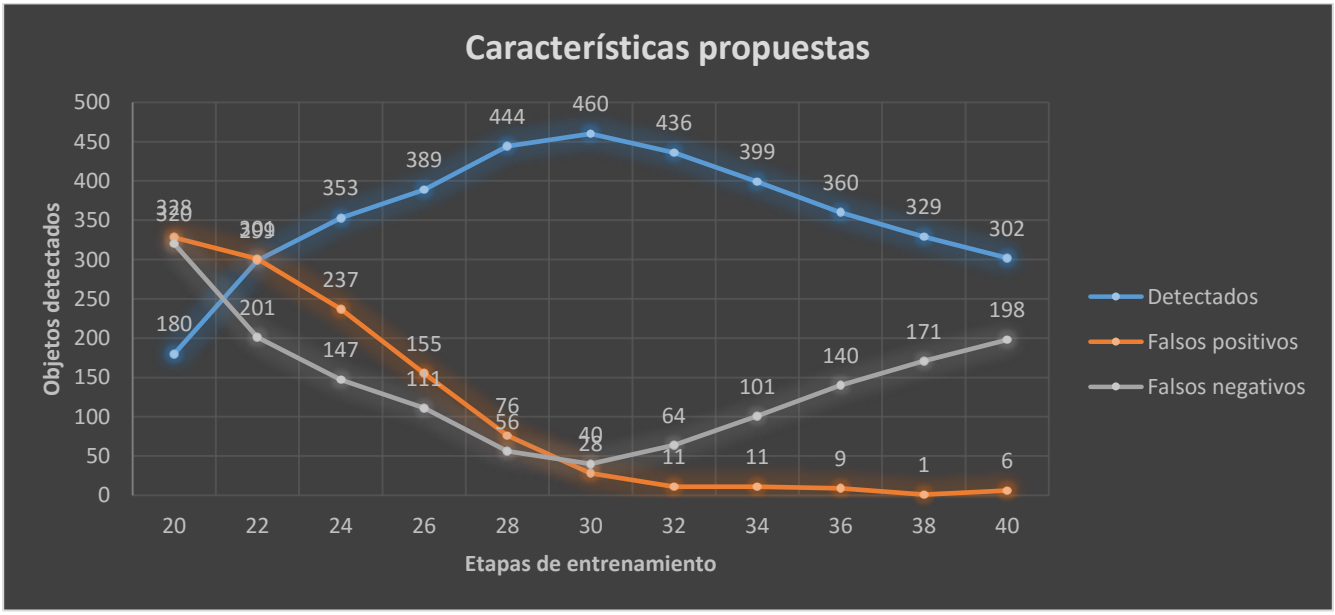


Figura 34: Grafica de las 10 iteraciones usando las características de OpenCV mas las propuestas

En la Figura 35 se puede observar una muestra de aplicación del sistema de detección utilizando el conjunto de características de OpenCV y el conjunto de características propuesto. Se aprecia como el detector con las características de OpenCV no logra detectar el vehículo en la imagen, y el sistema con el conjunto propuesto si tiene éxito. Otra muestra de la aplicación de los sistemas de detección se puede ver en la *Figura 36*, donde la efectividad del sistema con las características propuestas se muestra superior. Cabe destacar que, en esta última comparación, las imágenes no corresponden con la

perspectiva del vehículo con la que se entrenaron, ya que las imágenes de prueba muestran una vista lateral del vehículo.

Figura 35: Ejemplo 1 de detección con características de OpenCV (izquierda) y las propuestas (derecha) y el mismo número de etapas de entrenamiento.

Figura 36: Ejemplo 2 de detección con características de OpenCV (izquierda) y las propuestas (derecha) y el mismo número de etapas de entrenamiento.

Otra muestra de las pruebas que se han realizado el sistema con las características propuestas se puede apreciar en la Figura 37. En esta se muestra una secuencia de imágenes obtenida de un archivo de video del flujo vehicular de una ciudad. La detección realizada con el sistema y características propuestas es buena ya que se caracteriza por un nivel muy bajo de falsos negativos y falsos positivos prácticamente inexistentes.

Cabe destacar que, como a cualquier sistema de detección basado en características Haar, las escalas en los vehículos de las imágenes de prueba afectan dramáticamente la detección.

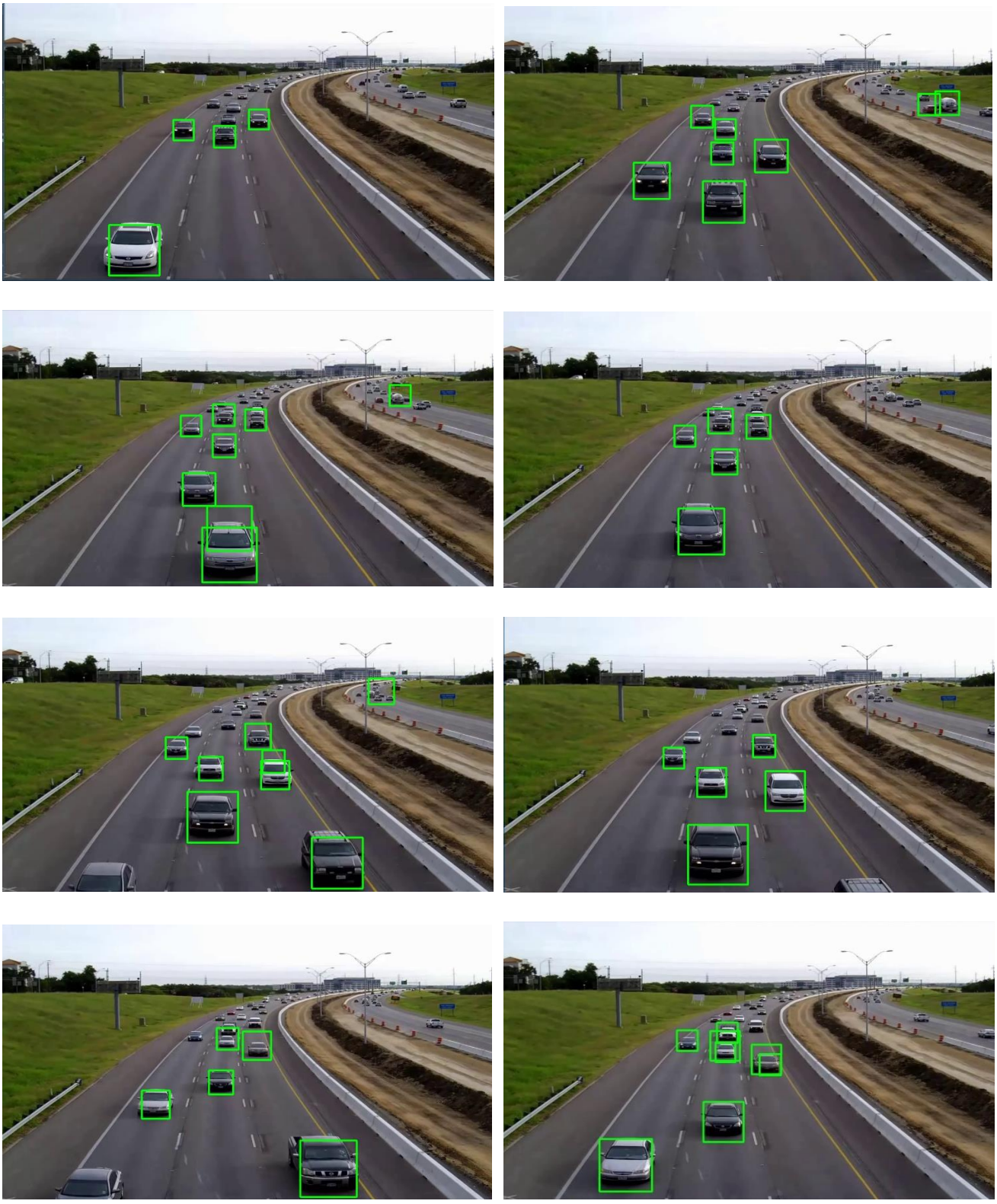


Figura 37: Secuencia de imágenes de prueba obtenidas de un video del flujo vehicular en una avenida.

7. Conclusiones y Trabajo Futuro

En este trabajo se ha propuesto un conjunto de características visuales para la detección de vehículos en imágenes, el cual es diferente al conjunto definido en las librerías de OpenCV. El conjunto propuesto se compone de las 14 características de OpenCV más 10 características nuevas. La experiencia de introducir estas nuevas características en el mecanismo de detección de OpenCV fue particularmente difícil pero enriquecedora.

El sistema de detección completo incluye, además de las características, al algoritmo de entrenamiento AdaBoost y al algoritmo de clasificación Cascade, el cual es un esquema muy utilizado en la bibliografía. Este sistema ha demostrado ser capaz de detectar vehículos en imágenes donde el conjunto original de características de OpenCV ha fallado.

Por otro lado, se ha demostrado que la efectividad en la detección se encuentra directamente relacionada con las etapas de entrenamiento que hayan sido realizadas para el sistema completo. La diferencia más grande encontrada entre los dos grupos de características fue con 30 etapas de entrenamiento, donde el conjunto de características propuesto mostró un 92% de efectividad contra un 87% del conjunto de OpenCV.

En la mayoría de las pruebas, el desempeño de nuestras características fue similar o superior que las obtenidas con el conjunto de OpenCV. Particularmente, en el rubro de falsos positivos nuestras características demostraron un desempeño superior.

Aunque las características propuestas han permitido obtener mejores detecciones con respecto al conjunto original de OpenCV, existen varias alternativas que pudieran ayudar a obtener mejores resultados. Especialmente, si se quiere realizar la detección con imágenes que tengan perspectivas diferentes respecto al vehículo. Es por esto, que se puede identificar un trabajo futuro basado en otras propuestas como las de Quichang, Sakrapee y Chunhua [2] donde se realiza la separación de las imágenes desde diferentes perspectivas aplicando algoritmos de *clustering*. Esto permitiría robustecer el desempeño del esquema general usado en este trabajo.

Bibliografía

[1] V. Paul y J. Michael, «Rapid Object Detection using a Boosted Cascade of Simple Features,» de *Computer Vision and Pattern Recognition*, 2001.

[2] Q. Hu, S. Paisitkriangkrai, C. Shen, A. v. d. Hengel y F. Porikli, «Fast detection of multiple objects in traffic scenes,» p. 13, 2015.

[3] C. d. OpenCV, «OpenCV sitio web,» 3 Agosto 2017. [En línea]. Available: https://opencv.org/. [Último acceso: 27 Noviembre 2017].

[4] W. Song, X. Zhou y H. Chen, «Automatic localization of solid organs on 3D CT images by a collaborative majority voting decision based on ensemble learning,» *Computerized Medical Imaging and Graphics,* 2012.

[5] T. Gebru, J. Krause, Y. Wang, D. Chen, J. Deng y L. Fei-Fei, «Fine-Grained Car Detection for Visual Census Estimation,» 2017.

[6] T. Ahonen, A. Hadid y M. Pietikäinen, «Face Recognition with Local Binary Patterns».

[7] S. Guennouni, A. Ahaitouf y A. Mansouri, «A Comparative Study of Multiple Object Detection Using Haar-Like Feature Selection and Local Binary Patterns in Several Platforms,» *Modelling and Simulation in Engineering,* p. 8, 2015.

[8] A. S. I. &. H. G. E. Krizhevsky, «Imagenet classification with deep convolutional neural networks,» de *In Advances in neural information processing systems (pp. 1097-1105).*, 2012.

[9] D. M. U. &. S. J. Ciresan, «Multi-column deep neural networks for image classification,» de *In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on (pp. 3642-3649). IEEE.*, 2012.

[10] P. &. S. G. Geismann, «A two-staged approach to vision-based pedestrian recognition using Haar and HOG features,» de *Intelligent Vehicles Symposium, 2008*, 2008.

[11] M. Z. T. N. D. B. M. &. V. S. W. Schwarzinger, «Vision-based car-following: detection, tracking, and identification,» de *In Intelligent Vehicles' 92 Symposium., Proceedings of the (pp. 24-29). IEEE.*, 1992.

[12] S. M. O. M. R. F. &. P. N. P. Gupte, «Detection and classification of vehicles,» de *Intelligent Transportation Systems, IEEE Transactions on, 3(1), 37-47.*, 2002.

[13] S. Reinius, «Object recognition using the OpenCV Haar cascade-classifier on the iOS platform,» 2013.

[14] V. Singh, V. Shokeen y B. Singh, «Face detection by HAAR Cascade classifier with simple and complex backfround images using OpenCV implementation,» 2013.

[15] S. Mallat, «A theory for multiresolution signal decomposition: The wavelet representation,» 1989.

[16] T. Ojala, M. Pietikainen y D. Harwood, «Performance evaluation of texture measures with classification based on Kullback discrimination of distributions,» *Pattern recognition,* 1994.

[17] C. Papageorgiou, M. Oren y T. Poggio, «A general framework for object detection,» *Computer Vision, 1998. Sixth International Conference on,* 1998.

[18] L. Breiman, «Bagging Predictors,» p. 18, 1996.

[19] Y. Freoud y R. E. Schapire, «A decision-theoretic generalization of on-line learning and an

application to boosting,» *In Computational Learning Theory,* 1995.

[20] S. Puttemans, «Cacade Classifier training,» OpenCV, 2017. [En línea]. Available: https://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html. [Último acceso: 10 2017].

[21] J. Howse, S. Puttemans, Q. Hua y U. Sinha, OpenCV 3 Blueprints, United kingdom: Packt Publishing Ltd., 2015.

[22] A. Gepperth, E. Johann y Thoma, «Real-time detection and classification of cars in video sequences,» de *Intelligent Vehicles Symposium, 2005*, 2005.

[23] T. C. C. E. J. I. C. K. D. L. I. .. &. V. S. W. Bücher, «Image processing and behavior planning for intelligent vehicles,» de *Industrial Electronics, IEEE Transactions on, 50(1), 62-75*, 2003.

[24] V. &. C. P. Wojcik, «Algorithms for speedy visual recognition and classification of patterns formed on rectangular imaging sensors,» de *Neurocomputing, 74(1), 140-154.*, 2010.

[25] P. I. Wilson y J. D. Fernandez, «Establishing a Face Recognition Research Environment Using Open Source Software,» 2005.

Apéndice A: Ejemplo de Implementación de la Imagen Integral en C++

Utilizando como representación básica de una imagen la siguiente definición:

```
using Mat = vector<vector<float>>;
```

Podemos calcular la imagen integral de una imagen con la siguiente función:

```
Mat integral(Mat &img) {
    Mat n_img(img.size(), vector<float>(img[0].size(), 0));
    float sum;
    vector<float> rSum(n_img[0].size(), 0);
    for (size_t i = 0; i < n_img.size(); ++i) {
        sum = 0.0f;
        for (size_t j = 0; j < n_img[0].size(); ++j) {
            sum = img[i][j] + sum;
            n_img[i][j] = sum + rSum[j];
            rSum[j] += sum;
        }
    }

    return n_img;
}
```

Apéndice B: Código de las Nuevas Características Propuestas

```
// esq sup iz
if ((x + dx * 3 <= winSize.width) && (y + dy * 3 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 3, dy * 3, -2,
            x + dx, y, dx * 2, dy, 3,
            x, y, dx, dy * 3, 4));
}


// esq inf iz
if ((x + dx * 3 <= winSize.width) && (y + dy * 3 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 3, dy * 3, -2,
            x + dx, y + dy * 2, dx * 2, dy, 3,
            x, y, dx, dy * 3, 4));
}


// esq sup de
if ((x + dx * 3 <= winSize.width) && (y + dy * 3 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 3, dy * 3, -2,
            x, y, dx * 2, dy, 3,
            x + dx * 2, y, dx, dy * 3, 4));
}


// esq inf de
if ((x + dx * 3 <= winSize.width) && (y + dy * 3 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 3, dy * 3, -2,
            x, y + dy * 2, dx * 2, dy, 3,
            x + dx * 2, y, dx, dy * 3, 4));
}


// T
if ((x + dx * 3 <= winSize.width) && (y + dy * 3 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 3, dy * 3, -2,
            x, y, dx * 3, dy, 4,
            x + dx, y + dy, dx, dy * 2, 3));
}
```

```
// T invertida
if ((x + dx * 3 <= winSize.width) && (y + dy * 3 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 3, dy * 3, -2,
            x, y + dy * 2, dx * 3, dy, 4,
            x + dx, y, dx, dy * 2, 3));
}


// media semi borde derecha
if ((x + dx * 4 <= winSize.width) && (y + dy * 4 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 4, dy * 4, 1,
            x + dx * 2, y + dy, dx * 2, dy * 2, -4));
}


// media semi borde izquierda
if ((x + dx * 4 <= winSize.width) && (y + dy * 4 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 4, dy * 4, 1,
            x, y + dy, dx * 2, dy * 2, -4));
}


// media semi borde abajo
if ((x + dx * 4 <= winSize.width) && (y + dy * 4 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 4, dy * 4, 1,
            x + dx, y + dy * 2, dx * 2, dy * 2, -4));
}


// media semi borde arriba
if ((x + dx * 4 <= winSize.width) && (y + dy * 4 <= winSize.height))
{
      features.push_back(Feature(offset, false,
            x, y, dx * 4, dy * 4, 1,
            x + dx, y, dx * 2, dy * 2, -4));
}
```

Apéndice C: Licencia de OpenCV

OpenCV es distribuido bajo una licencia *open source*, cualquier uso, distribución, modificación o copia de dicho software debe contener la siguiente leyenda que explica su forma de uso.

IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.

By downloading, copying, installing or using the software you agree to this license.
If you do not agree to this license, do not download, install,
copy or use the software.

License Agreement
For Open Source Computer Vision Library

Copyright (C) 2008-2013, Itseez Inc., all rights reserved.
Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

** Redistribution's of source code must retain the above copyright notice, this list of conditions and the following disclaimer.*

** Redistribution's in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.*

** The name of Itseez Inc. may not be used to endorse or promote products derived from this software without specific prior written permission.*

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the copyright holders or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.