TRABAJO ESPECIAL DE GRADO

DISEÑO DE UN SISTEMA DE RECONOCIMIENTO DE MATRÍCULAS VEHICULARES A TRAVÉS DEL PROCESAMIENTO DE IMÁGENES Y MACHINE LEARNING

Presentado ante la ilustre
Universidad Central de Venezuela
por el Br. Natalia Sofia Molina Ramos
para optar al título de
Ingeniero Electricista.

Caracas, agosto de 2020

TRABAJO ESPECIAL DE GRADO

DISEÑO DE UN SISTEMA DE RECONOCIMIENTO DE MATRÍCULAS VEHICULARES A TRAVÉS DEL PROCESAMIENTO DE IMÁGENES Y MACHINE LEARNING

TUTOR ACADÉMICO: William La Cruz.

Presentado ante la ilustre
Universidad Central de Venezuela
por el Br. Natalia Sofia Molina Ramos
para optar al título de
Ingeniero Electricista.

Caracas, agosto de 2020

Dedico el éxito de este trabajo de grado principalmente a mí. *Este trabajo no solo me permitió contribuir al campo de investigación siendo un gran honor, sino que me dejó un invaluable crecimiento personal y profesional.*

A mi Padre ***Jaime Molina****, cuyo sueño fue verme pisar la gran aula magna de la UCV, aunque no podré por fuerzas ajenas a nuestra voluntad, entrego todo el amor y éxito que este trabajo representó a tu sueño. Gracias infinitamente por toda la sabiduría que me entregaste.*

A mi hermano ***Ricardo Molina****, quien siempre me motivó y creyó en mis capacidades para alcanzar grandes metas.*

Todo está al alcance de nuestras manos si nos damos la oportunidad de aprender con dedicación, disciplina y paciencia; dándonos la oportunidad de fallar en el proceso.

RECONOCIMIENTOS Y AGRADECIMIENTOS

A mi equipo de laboratorio, a mi compañero de estudio, a mi amigo y pareja **Marco Rodríguez**, quien siempre creyó en mí, en mis fortalezas y capacidades para alcanzar éxito de este proyecto y muchos más.

A mí tutor el Prof. **William La Cruz** por guiarme en todo el proceso de creación de este importante proyecto, desde la conceptualización hasta la implementación. Agradezco infinitamente todo el aprendizaje profesional y en especial el crecimiento personal que obtuve durante el desarrollo de este trabajo junto a usted.

A mí Madre Yamilca Ramos por todo el apoyo incondicional que siempre he recibido, gracias a tí y mi padre estoy en este importante momento de mi vida. A todos mis familiares y amigos que me apoyaron, fueron una pieza importante en este proceso.

Finalmente, quiero agradecer a todos los profesores que contribuyeron en algún momento en mi formación profesional e integral durante la carrera y a la UCV, la casa que vence las sombras, por entregarme todo lo que pudo para hoy ser una Ingeniera Electricista.

Natalia Molina.

DISEÑO DE UN SISTEMA DE RECONOCIMIENTO DE MATRÍCULAS VEHICULARES A TRAVÉS DEL PROCESAMIENTO DE IMÁGENES Y MACHINE LEARNING

Tutor Académico: William La Cruz. Tesis. Caracas, Universidad Central de Venezuela. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Mención Electrónica. Año 2021, xvii, 144 pp.

Palabras Claves: Procesamiento de imágenes, Máquina de soporte Vectorial, Machine learning.

Resumen.- La presente investigación propone el diseño de un sistema de Reconocimiento Automático de una matrícula vehicular mediante un programa que integra dos procedimientos, con la capacidad de localizar y reconocer la matrícula vehicular en un ambiente con condiciones controladas. El diseño propuesto se basa en seguir la siguiente secuencia de pasos: 1) detectar la placa; 2) segmentar los caracteres; y 3) reconocer la matrícula mediante un algoritmo de Machine Learning o de aprendizaje automático. El algoritmo seleccionado para el reconocimiento de caracteres fue la Máquina de Soporte Vectorial, que es un algoritmo de aprendizaje supervisado que ha demostrado ser muy eficienciente en una gran variedad de aplicaciones en Ingeniería. La implementación del sistema propuesto se realizó de manera modular, en el que cada módulo representa cada uno de los pasos que conforman el sistema. Para lograr lo anterior, se realizó una investigación documental acerca de las técnicas de procesamiento de imágenes y de la Máquina de Soporte Vectorial, así como del lenguaje de programación Python y las librerías de código abierto especializadas en el procesamiento de imágenes y Machine Learning. Se seleccionó la Máquina de Soporte Vectorial por su capacidad de trabajar en dimensiones grandes y su efectividad para el reconocimiento de patrones. Se seleccionaron las técnicas de procesamiento de imágenes más adecuadas para los requerimientos de la detección de placa y la segmentación de caracteres, considerando herramientas optimizadas y ampliamente utilizadas en la visión por computador. En el trabajo se presenta un conjunto de pruebas numéricas con distintas placas para mostrar el comportamiento computacional del sistema propuesto. Los resultados obtenidos indican que el sistema de reconocimiento diseñado es efectivo y eficiente en el reconocimiento de las matrículas procesadas.

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS

ANPR: Automatic Number Plate Reconigtion, Reconocimiento Automático de Número de Placa.

SVM: Suport Vector Machine, Maquina de Soporte Vectorial.

ML: Machine Learning.

INTRODUCCIÓN

Con el pasar de los años y los avances de la tecnología la automatización de procesos ha ganado importancia como la innovación que ha dado paso a la nueva sociedad tecnológica que hoy conocemos. En diferentes campos industriales, comerciales y militares la automatización de procesos ha reemplazado la labor del ser humano en tareas repetitivas en un principio. Conforme el desarrollo de la tecnología avanza las aplicaciones modernas han ganado complejidad exigiendo la utilización de sensórica más complejas y sistemas de control con capacidad de toma de decisiones. Así, la innovación ha evolucionado desde cámaras de vigilancia a sistemas de seguridad con reconocimiento de objetos y movimiento. Los sensores ópticos, en especial las cámaras de fotográfica, destacan como instrumentación eficiente para generar sistemas modernos que desde la captura de imagen pueden extraer información de interés que permita la toma de decisiones, como lo son los sistemas de Reconocimiento Automático de Número de Placa (ANPR, por sus siglas en inglés). No obstante, emplear sensores como las cámaras fotográficas requieren el diseño de sistemas computacionales capaces de procesar y utilizar la información contenida en tales imágenes mediante diferentes procedimientos matemáticos, geométricos y estadísticos, que en conjunto se conoce como Visión por Computador.

Con lo anterior expuesto, este trabajo plantea el diseño de un sistema de Reconocimiento Automático de Matrícula vehícular, cuyo propósito es automatizar la identificación del vehículo que posteriormente se podrá integrar a un sistema de entrada y salida en un estacionamiento o sistemas de vigilancia, sistemas de rastreo, entre otros. El diseño propuesto se basó en seguir la siguiente secuencia de pasos: 1) detectar la placa; 2) segmentar los caracteres; y 3) reconocer la matrícula

mediante un algoritmo de Machine Learning o de aprendizaje automático. Así, se llevó a cabo una investigación documental acerca de las técnicas de procesamiento de imágenes y de la Máquina de Soporte Vectorial, así como del lenguaje de programación Python y las librerías de código abierto especializadas en procesamiento de imágenes y Machine learning.

La selección se caracterizó por obtener herramientas optimizadas y ampliamente utilizadas en la visión por computador que se ajustan a las necesidades de la detección de la placa, segmentación de caracteres, garantizando la información necesaria para la compresión de sus conceptos. Para la implementación del sistema propuesto se utilizó el lenguaje de programación Python, que es libre y ofrece gran variedad de librerías de código abierto. En particular, Python cuenta con librerías optimizadas para el procesamiento de imágenes, las cuales se basan en técnicas matemáticas como *Histogramas de Gradientes Orientados*. Se seleccionó la biblioteca de código abierto OpenCV por su amplia utilización en la visión por computador, con una gran variedad de técnicas y aplicaciones en el campo de investigación. Por otro lado, se seleccionó la Maquina de Soporte Vectorial por su capacidad de trabajar en grandes dimensiones y su efectividad para el reconocimiento de patrones, siendo uno de los algoritmos más importantes en el campo de Machine Learning. Para su implementación se seleccionó Scikit-Learn, biblioteca de basto desarrollo en los campos de aprendizaje automático de software libre para el lenguaje de programación Python.

El desarrollo del módulo de procesamiento de imagen da salida a la entrada a la Máquina de Soporte Vectorial que se encargará, mediante el modelo generado, de reconocer la identificación de la matrícula. Dada a la forma en que se plantea el procesamiento de imágenes, la Máquina de Soporte Vectorial tiene la capacidad de reconocer rápidamente el caracter que se le ha entregado como entrada y, debido a las fortalezas de este algoritmo, el modelo reconoce nuevas muestras. Las principa-

les debilidades de esta estructura se encuentran en el procesamiento de imágenes dónde la posición de captura incide directamente en el desempeño general y la disposición de muestras para la construcción de una base de datos que se ajuste a la estructura del sistema. Esta idea general busca formar parte del conjunto de aplicaciones del campo de investigación reconocimiento de texto, utilizando algoritmos de aprendizaje automático basados en la detección y reconocimiento de patrones.

En el trabajo se realizaron un conjunto de pruebas numéricas con diferentes placas para mostrar el comportamiento computacional del sistema propuesto. Las pruebas evaluaron el desempeño del modelo generado con placas que formaron parte de la base de datos de entrenamiento y con placas que no formaron parte de la base de datos, cuyos resultados demostraron más de 98 % y 93 % de tasa de éxito, respectivamente. Así mismo, las pruebas mostraron que el módulo de procesamiento de imagen, así como las condiciones físicas de la matrícula, inciden significativamente en el módulo de reconocimiento, por lo que es posible optimizar los resultados obtenidos mejorando el módulo del procesamiento de imágenes. Las pruebas mostradas en este trabajo se realizaron sobre imágenes capturadas en ambientes con condiciones controladas en iluminación y posición. Los resultados obtenidos indican que el sistema de reconocimiento diseñado es efectivo y eficiente en el reconocimiento de matrículas procesadas.

Dentro de los alcances de este proyecto se encuentra solo el diseño y desarrollo de un sistema de reconocimiento de matrículas vehículares, empleando procesamiento de imágenes y Machine Learning. Este diseño posteriormente podrá ser llevado a un prototipo de implementación física.

El resto del trabajo está organizado de la siguiente manera:

En el Capítulo 1, se expone el planteamiento del problema, junto con los

objetivos general y específicos del presente trabajo.

En el Capítulo 2, se muestra el marco referencial o teórico, en donde se explican los conceptos, herramientas y métodos utilizados.

En el Capítulo 3, se ofrece una descripción general del proyecto, donde se explica el funcionamiento del sistema propuesto y los diferentes módulos que lo conforman, mostrando una descripción detallada del software y bibliotecas utilizadas.

En el Capítulo 4, se reportan las diferentes pruebas realizadas para validar el funcionamiento de la rutina, como también sus respectivos resultados. Por último, se presentan las conclusiones y algunas recomendaciones para futuros proyectos.

CAPÍTULO I

CONCEPTUALIZACIÓN DEL PROYECTO

1.1. PLANTEAMIENTO DEL PROBLEMA

La aparición del transporte vehicular ha causado gran impacto en el desenvolvimiento diario de la sociedad. Los vehículos, en mayor o menor medida, se han convertido en una necesidad, permitiendo a las personas movilizarse a distintos sitios, como el trabajo, sedes de estudios, recreación, entre otros. Además, la integración de los vehículos a la vida cotidiana implica el cumplimiento de una serie de normas para su correcta utilización y prevención de accidentes, es decir, para la seguridad de los usuarios y personas alrededor.

Entonces, cualquier ciudadano común, con los permisos oficiales necesarios, puede adquirir y conducir un vehículo personal. En consecuencia, las personas regularmente aparcan sus vehículos en estacionamientos, públicos o privados, a lo largo de sus actividades diarias. Sin embargo, el robo de vehículos se ha incrementado, originando a los usuarios la inseguridad de visitar distintos lugares debido al riesgo a su bien privado.

Actualmente, los métodos que se disponen para la recuperación del vehículo representan un proceso complejo que puede tomar mucho tiempo y pudiera incluso no alcanzar su objetivo, implicando grandes costos en tiempo y dinero. De este modo, surge la necesidad de crear un sistema que permita monitorear y localizar

vehículos de manera económica y eficiente. Aunque, existen distintas opciones de vigilancia o rastreo vehicular que funcionan para mejorar la seguridad,

Los avances de la tecnología ha permitido simplificar tareas repetitivas o resolver tareas de gran complejidad. Entre los avances significativos destacan los sistemas de seguridad. Existen dispositivos como los GPS que permiten localizar con exactitud los vehículos desaparecidos, a pesar de esto, sus costos de instalación y servicio suelen ser muy elevados y es necesario instalarlos en cada vehículo. Para cubrir las necesidades de control y vigilancia se diseñaron sistemas como el reconocimiento automático de matrícula, los cuales están compuestos por una cámara de video para capturar la matrícula del vehículo y un computador donde se encuentra el cerebro del sistema.

Los ANPR, no solo eliminan la necesidad de tener más personal de vigilancia en el área, sino su costo a largo plazo es mucho menor comparado a otras alternativas y no requiere una intervención directa en los vehículos. Adicionalmente, es posible aumentar el área de vigilancia incorporando más cámaras de video a la red. Por otro lado, los ANPR pueden desempeñar también aplicaciones orientadas a la detección de infracciones, control de acceso, entre otros.

Debido a lo anterior expuesto, existe la necesidad de diseñar un sistema que sea capaz de obtener la información de una matrícula vehicular de manera automática a partir de la captura de su imagen.

1.2. OBJETIVOS

1.2.1. OBJETIVO GENERAL

Diseñar un sistema de reconocimiento de matrícula vehicular a través del procesamiento de imágenes y Machine Learning.

1.2.2. OBJETIVOS ESPECÍFICOS

1. Analizar las técnicas para el procesamiento de imágenes.
2. Analizar las técnicas de aprendizaje supervisado para la clasificación y/o reconocimiento de patrones, en particular, la Máquina de Soporte Vectorial.
3. Diseñar un procedimiento para la adquisición y localización de la matrícula en la imagen procesada, a través de técnicas de procesamiento de imágenes.
4. Aplicar la Máquina de Soporte Vectorial para el reconocimiento de los caracteres de la matrícula.
5. Implementar en un lenguaje de programación de libre acceso, la propuesta de sistema que permita reconocer los caracteres que conforman la matrícula.
6. Realizar experiencias computacionales dirigidas a la verificación del sistema de reconocimiento de matrícula diseñado, utilizando para ello imágenes de matrículas de prueba.

1.3. ANTECEDENTES

Los ANPR son tecnologías de procesamiento de imágenes que permite obtener el número de la matrícula vehicular de una captura de imagen de cámara digital. A continuación se presentan algunos trabajos de referencias:

El Ing. Reyders Morales en su trabajo de grado "Diseño de un sistema de reconocimiento de embarcaciones en medio marítimo mediante el procesamiento de imágenes" [16], pretendía satisfacer la necesidad en el ámbito civil y militar de detectar embarcaciones en el medio marítimo. En su propuesta solo se probó el concepto de un programa prototipo de este sistema y sus pruebas fueron realizadas sobre imágenes tomadas en ambientes controlados. Su metodología se basó en un descriptor morfológico denominado Histograma de Gradientes Orientados. Con dicho descriptor, se opera una Máquina de Soporte Vectorial (SMV) que permite realizar la clasificación del objeto de interés. Entre sus pruebas destacó que la similitud y los solapamientos entre los objetos de interés presentan problemas de detección, la mejor región de ajuste de la SVM no es lineal y la base de datos de entrenamiento no se ajustó a los requerimientos. Por lo tanto, su principales recomendaciones fueron construir una base de datos con tratamientos estadísticos que permita entrenar la SVM exitosamente y profundizar los conocimientos de la SVM para mejorar los ajustes.

El Ing. José Bracho en su trabajo de grado "Diseño e implementación de un sistema de reconocimiento de matrícula vehículares" [3], tenía como objeto desarrollar una solución para el robo de vehículos que se desenvuelve en Venezuela mediante un ANPR. Su solución se basó en las herramientas OpenSource OpenCV, para la adquisición de la imagen y la localización de la matrícula, y TesseracORC para la detección y reconocimiento de caracteres. Llevó a cabo su implementación

en un módulo de Raspberry Pi. Entre sus principales resultados se obtuvo que su tasa de éxito para la detección se encontraba alrededor de 50 %, así mismo, para el reconocimiento de los caracteres, aunque presentaba una especial dificultad para los caracteres numéricos. Según lo anterior, entre sus recomendaciones se encuentran evaluar los diferentes algoritmos de OpenCV para la detección de objetos, aumentar el entrenamiento de la clasificadora para obtener mejores resultados, implementar el sistema en un computador o dispositivo de mayor capacidad, entre otros.

Dhiraj y Pramod [8] describen los ANPR como un proceso que se pueden separar en las siguientes etapas: la detección del vehículo y la captura del mismo en la parte frontal o trasera, la localización y la extracción del número de matrícula de la imagen. El último paso lo denominan como técnica de segmentación, la cual se puede realizar a través de diferentes métodos como en una red neuronal, morfología matemáticas, análisis de colores o análisis de histograma. Explican en sus resultados que el rendimiento fue mejor en estático y su sistema funcionó para diferentes condiciones y placas. Así mismo, resaltan la importancia de la cámara, la cual influye en la velocidad de respuesta y sus sensibilidades ante las vibraciones. Además, el sistema desarrollado con Reconocimiento Óptico de Caracteres fue sensible a las desaliniaciones y los diferentes tamaños de placas.

El Ing. Arcadio Fernández en su trabajo de grado "Desarrollo de un modelo computacional para un sistema de reconocimiento de matrículas a través de una imagen proveniente de una cámara de tráfico" [13], realizó el estudio de diferentes algoritmos y técnicas para el desarrollo de las etapas que componen el reconocimiento de caracteres, con énfasis en el procesamiento de imágenes; para su posterior selección y programación. Se basó en la red neuronal Perceptrón multicapa para el reconocimiento de los caracteres y su implementación la llevó a cabo en Scilab. Alcanzó realizar el modelo computacional deseado satisfactoria-

mente, pudiendo reconocer matrículas antiguas y actuales. Sin embargo, señala la importancia de un buen entrenamiento para la clasificación de caracteres. Entre sus recomendaciones más importantes se encuentra mejorar el entrenamiento de las redes y recibir dimensiones de las placas lo más estables posibles.

CAPÍTULO II

MARCO REFERENCIAL

Para el desarrollo de este trabajo, es necesario manejar conceptos básicos del procesamiento de imagen y aprendizaje automático.

2.1. Imagen digital

Una imagen se define, en este contexto, como la representación visual de un objeto real a través de técnicas como la fotografía, la pintura, el video, entre otras técnicas. Entonces, una imagen digital puede ser definida como una función bidimensional $f(x, y)$, dónde (x, y) son coordenadas espaciales y $f(x, y)$ es la intensidad de la imagen en ese punto. Las imágenes digitales están compuestas por un número finito de elementos llamados *pixel*. Las imágenes digitales dependiendo de si es dinámica o estática se pueden clasificar en dos tipos: *imagen matricial* o *gráfico vectorial*.

El gráfico vectorial o la imagen vectorial, es una imagen digital formada por entidades geométricas independientes (segmentos, polígonos, arcos, entre otros), cada uno de ellos definidos por fórmulas matemáticas. Se construyen a partir de vectores y no se dividen en unidades mínimas de información como los pixeles, sino en manchas de color y lineas [2] .

Por otro lado, la imagen matricial o mapa de bits es una estructura que

representa una rejilla rectangular compuesta de pixeles o puntos de color. Estos, se suelen definir por su altura y grosor (en pixeles) por su profundidad de color. Esto determina el número de colores distintos que se pueden almacenar en cada punto individual. La calidad de las imágenes rasterizadas está definida por el total de pixeles que posee (Resolución) y la cantidad de información por pixel (Profunidad de color, bits por pixel). [1]

Los pixeles guardan información de color en un determinado punto, es decir, una representación numérica de color y esta se ve limitada por la cantidad de bits utilizados para representarla, esto se conoce como profunidad de color. Normalmente, cada pixel es representado por tres valores numéricos.

2.2. Espacios de color

El espacio de color define un modelo de composición de color. Por lo general, un espacio de color se compone de N vectores cuya combinación lineal puede generar todo el espacio de color. Generalmente, lo espacios de color intentan representar todos los colores que el ojo humano puede percibir, mientras que otros aislan un subconjunto específico de colores. Los espacios de color pueden ser: una dimensión (Escala de grieses), dos dimensiones (RGB, CIEXYZ, CIELAB, YIQ) o cuatro dimensiones (CMYK). Los espacios de color de tres dimensiones son, normalmente, los más usados. Es decir, un color se especifica usando tres coordenadas, la cual determina su ubicación en este espacio.

2.3. Procesamiento de imágenes digitales

El procesamiento digital de imágenes se define como el conjunto de técnicas y métodos desarrollados para manipular la información contenida en una imagen

digital. Estas técnicas consisten en aplicar diferentes operadores a la imagen con los siguientes objetivos [23]:

- *Restauración de la imagen*: mejorar la calidad de la imagen de forma objetiva, como lo es reducir el ruido.
- *Mejoramiento de la imagen*: mejorar la calidad de la imagen de forma subjetiva, como incrementar el contraste, crear distorsión, entre otros.
- *Compresión de la imagen*: consiste en representar la imagen con la menor cantidad de bits posible, sin afectar críticamente la calidad de la imagen, como lo es la reducción de dimensión, la binarización, entre otros.
- *Extracción de objetos*: resaltar explícitamente algunas características en la imagen que permitan la detección de objetos, tales como la utilización de algoritmos para detección y reconocimiento de contornos.

2.3.1. Mascaras derivativas discretas

El proceso de filtrado de una imagen se realiza mediante la convolución entre los distintos pixeles que componen la imagen y una matriz de convolución. Esta matriz es denominada "núcleo"del filtro. Dependiendo de los valores que componen al núcleo y su distribución, se obtienen diferentes resultados de filtrado en la imagen. Las mascaras derivativas discretas no son más que núcleo cuyos elementos representan una aproximación de la derivada [4]. En la figura 2.1 se puede apreciar el proceso de convolución sobre un pixel.

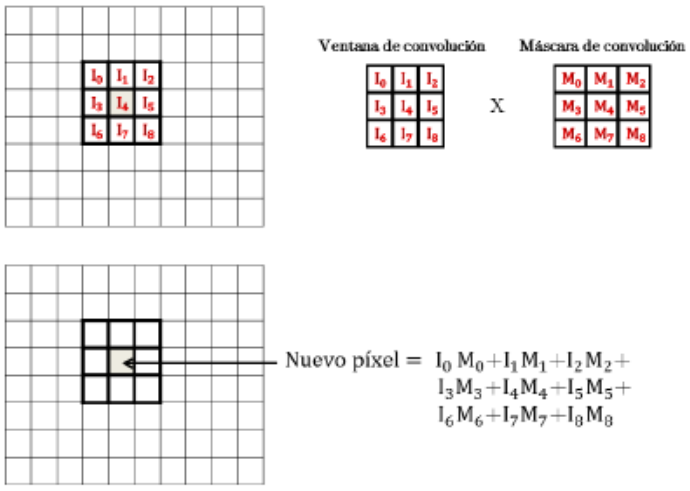

Figura 2.1: Operación de convolución sobre un pixel

Las mascaras derivativas son utilizadas para calcular el gradiente de una imagen, normalmente con la intención de detectar los contornos. Entre los más utilizados se encuentran: Sobel, Prewitt, Roberts y Laplaciano [14]. En la Figura 2.2 se muestra el efecto obtenido después de aplicar el operador de Sobel en una imagen.

Figura 2.2: Operador Sobel

Bibliotecas de software libre orientadas hacia la visión computarizada como

OpenCv, ofrecen funciones que utilizan estos operadores para determinar el gradiente de la imagen y así detectar los contornos mediante la umbralización como puede apreciarse en las Figuras 2.3 y 2.4:

Figura 2.3: Umbralización adaptativa de librería OpenCV

Figura 2.4: Filtro de sobel vs Umbralización adaptativa

Existen otras funciones especializadas en la detección de contornos como lo es Canny de OpenCV (Ver el ejemplo de la Figura 2.5).

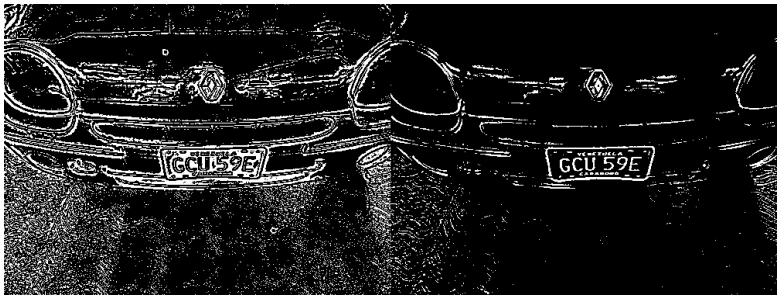

Figura 2.5: Función Canny vs Umbralización adaptativa

2.3.2. Binarización

La binarización es una técnica que consiste en la realización de un barrido en la matriz de la imagen digital, por medio de bucles o recursividad, con el fin de que el proceso produzca la reducción de la escala de grises a dos únicos valores. Negro(= 0) y blanco (= 255), o lo que es lo mismo, un sistema binario de ausencia y presencia de color 0-1. La comparación de cada píxel de la imagen viene determinada por el umbral de sensibilidad (valor T = Threshold). Por ejemplo, los valores que sean mayores que el umbral toman un valor 255 (blanco) y los menores 0 (negro).

Figura 2.6: Imagen a color binarizada.

En base a las particularidades entre algoritmos categorizan los métodos de umbralización en seis grupos. Aquí añadimos uno más, los métodos globales [15]:

- *Histograma*: métodos basados en el análisis de los picos máximos y mínimos de las curvas del histograma del suavizado de la imagen.
- *Clustering*: métodos basados en discernir como las muestras de los niveles de gris se agrupan o alternativamente se modelan como una mezcla de dos gaussianas.
- *Entropía*: métodos basados en el análisis de los resultados de la aplicación

de algoritmos que utilizan la entropía de las regiones frontal y de fondo, la entropía cruzada entre la imagen original y binarizada.

- *Similitud*: métodos basados en la búsqueda de una similitud entre las escalas de grises, como la tonalidad difusa, los bordes de la imagen, etc.
- *Espaciales*: métodos analíticos que usan el orden de distribución, la probabilidad y/o la correlación entre los diferentes píxeles.
- *Globales*: métodos cuyo valor del umbral es estático.
- *Locales*: métodos que adaptan el valor del umbral, de forma manual o automática, a cada píxel dependiendo

2.4. Machine Learning y Máquina de Soporte Vectorial

"*Machine Learning*" (ML) o aprendizaje automático (por su traducción al español) es una rama de la inteligencia artificial que está constituida por un conjunto de algoritmos que automatizan la construcción de modelos analíticos a partir del análisis de datos. La ML se fundamenta en la idea de que los sistemas pueden aprender de los datos, identificar patrones y tomar decisiones con una mínima intervención humana. La ML también se puede definir como el proceso de resolver un problema práctico mediante ([24]): 1) la recopilación de un conjunto de datos y 2) la construcción algorítmica de un modelo estadístico basado en ese conjunto de datos.

Por lo general, los métodos de aprendizaje automático se pueden clasificar de múltiples maneras bajo múltiples paradigmas. En el presente trabajo utilizamos la clasificación basada en la cantidad de supervisión humana en el proceso de aprendizaje, a saber:

a. *Aprendizaje supervisado*

b. *Aprendizaje no supervisado*

c. *Aprendizaje semi-supervisado*

d. *Aprendizaje reforzado*

2.4.0.1. Aprendizaje Supervisado

Los métodos o algoritmos de aprendizaje supervisado incluyen algoritmos de aprendizaje que toman muestras de datos (conocidas como datos de entrenamiento) y salidas asociadas (conocidas como etiquetas o respuestas) con cada muestra de datos durante el proceso de entrenamiento del modelo. El objetivo principal es aprender un mapeo o asociación entre las muestras de datos de entrada x y sus correspondientes salidas y, basándose en múltiples instancias de datos de entrenamiento. Este conocimiento aprendido se puede utilizar en el futuro para predecir una salida y' para cualquier nueva muestra de datos de entrada x', que antes se desconocía o no se veía durante el proceso de entrenamiento del modelo. Estos métodos se denominan supervisados porque el modelo aprende sobre muestras de datos donde las respuestas/etiquetas de salida deseadas ya se conocen de antemano en la fase de entrenamiento ([24]).

Existen dos clases principales de métodos de aprendizaje supervisado, según el tipo de tareas de aprendizaje automático que pretenden resolver:

- Regresión
- Clasificación

El objetivo principal de los métodos de aprendizaje supervisado para regresión es la estimación de un valor. Los métodos para regresión se entrenan en muestras

de datos de entrada que tienen respuestas de salida que son valores numéricos continuos. Los modelos de regresión hacen uso de atributos o características de los datos de entrada (también llamados variables explicativas o independientes) y sus correspondientes valores numéricos continuos de salida (también llamados respuesta, dependiente o variable de resultado) para aprender relaciones y asociaciones específicas entre las entradas y sus salidas correspondientes. Con este conocimiento, se puede predecir la respuesta de salida para instancias de datos nuevas y no vistas similares a la clasificación pero con salidas numéricas continuas.

En cambio, el objetivo principal de los métodos de aprendizaje supervisado para clasificación es predecir etiquetas de salida o respuestas que son de naturaleza categórica para los datos de entrada en función de lo que el modelo ha aprendido en la fase de entrenamiento. Las etiquetas de salida aquí también se conocen como clases o etiquetas de clase, ya que son de naturaleza categórica, lo que significa que son valores discretos y desordenados. Por lo tanto, cada respuesta de salida pertenece a una categoría o clase discreta específica. Un método de aprendizaje supervisado para clasificación es la *Máquina de Soporte Vectorial* (SVM, por sus siglas en inglés: Support Vector Machine) el cual describiremos más adelante.

2.4.0.2. Aprendizaje No Supervisado

Los métodos de aprendizaje no supervisado extraen conocimientos o información significativa de los datos en lugar de intentar predecir algún resultado basado en datos de entrenamiento supervisado previamente disponibles. Hay más incertidumbre en los resultados del aprendizaje no supervisado, pero también puede obtener mucha información de estos modelos que antes no estaba disponible para ver con solo mirar los datos sin procesar. A menudo, el aprendizaje sin supervisión podría ser una de las tareas involucradas en la construcción de un enorme sistema de inteligencia ([24]). Los métodos de aprendizaje no supervisado

se pueden clasificar en las siguientes áreas de la ML de mucha importancia para el aprendizaje no supervisado:

- Agrupación
- Reducción de dimensionalidad
- Detección de anomalías
- Asociación de minería de reglas

2.4.0.3. Aprendizaje Semi-Supervisado

Los métodos de aprendizaje semi-supervisados generalmente se encuentran entre los métodos de aprendizaje supervisados y no supervisados. Estos métodos suelen utilizar una gran cantidad de datos de entrenamiento que no están etiquetados (que forman el componente de aprendizaje no supervisado) y una pequeña cantidad de datos previamente etiquetados y anotados (que forman el componente de aprendizaje supervisado). Hay múltiples técnicas disponibles en forma de métodos generativos, métodos basados en gráficos y métodos basados en heurística. ([24])

2.4.0.4. Aprendizaje Reforzado

Los métodos de aprendizaje reforzado son un poco diferentes de los métodos convencionales supervisados o no supervisados. En este contexto, se cuenta con un agente que se desea capacitar durante un período de tiempo para interactuar con un entorno específico y mejorar su desempeño durante un período de tiempo, con respecto al tipo de acciones que realiza sobre el entorno. Normalmente, el agente comienza con un conjunto de estrategias o políticas para interactuar con el

entorno. Al observar el medio ambiente, toma una acción particular basada en una regla o política y observando el estado actual del medio ambiente. Según la acción, el agente obtiene una recompensa, que podría ser beneficiosa o perjudicial en forma de penalización. Actualiza sus políticas y estrategias actuales si es necesario y este proceso iterativo continúa hasta que aprende lo suficiente sobre su entorno para obtener las recompensas deseadas. Los pasos principales de un método de aprendizaje por refuerzo se mencionan a continuación. ([24])

1. Prepare al agente con un conjunto de políticas y estrategias iniciales.
2. Observe el entorno y el estado actual.
3. Seleccione la política óptima y realice una acción.
4. Obtenga la recompensa (o penalización) correspondiente.
5. Actualice las políticas si es necesario.
6. Repita los pasos 2 a 5 de forma iterativa hasta que el agente aprenda las políticas más óptimas.

2.4.1. Máquina de Soporte Vectorial

La SVM es un algoritmo de aprendizaje supervisado normalmente empleado en problemas de clasificación y, recientemente, en problemas de regresión. Los fundamentos matemáticos de la SVM han sido desarrollados por Vapnik [27] y están ganando popularidad debido a muchas características atractivas y un rendimiento empírico prometedor.

El problema de clasificación puede restringirse a la consideración del problema de dos clases sin pérdida de generalidad. Muchos problemas de clasificación pueden ser resueltos por la SVM si los grupos son linealmente separables. En este

problema, el propósito es separar las dos clases mediante una función que se induce a partir de los datos disponibles. El objetivo principal es producir un clasificador que funcione bien en ejemplos invisibles, es decir, que generalice bien. Considere el ejemplo de la Figura 2.7. Aquí hay muchos clasificadores lineales posibles que pueden separar los datos, pero solo hay uno que maximiza la distancia entre él y el conjunto de datos más cercano de cada clase. Este clasificador lineal se denomina *hiperplano de separación óptimo*. Intuitivamente, esperaríamos que este límite se generalizara bien en oposición a los otros límites posibles.

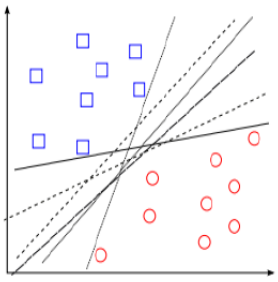

Figura 2.7: Hiperplanos de separación.

En caso que los datos no sean linealmente separables, existen procedimientos que emplean funciones denominadas *núcleos*, las cuales permiten usar la SVM para la separación de las clases. Pasemos a dar una breve descripción matemática de la SVM.

2.4.1.1. Datos Linealmente Separables

Asumamos que los datos a clasificar pueden ser separados linealmente en dos clases,

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_p, y_p), \quad \mathbf{x} \in \mathbb{R}^n, \quad y_i \in \{-1, +1\},$$

donde el $(\mathbf{x}_i, y_i)$ es un dato de entrenamiento, con $\mathbf{x}_i$ el *vector de atributos* e y_i la *etiqueta*. Se desea encontrar una función $f : \mathbb{R}^n \to \{-1, +1\}$, denominada *función*

de clasificación, que separe los dos clases de datos, es decir,

$$f(\mathbf{x}_i) = y_i, \quad i = 1, 2, \ldots, p.$$

Ahora bien, como los datos son linealmente separables, existen $\mathbf{w} \in \mathbb{R}^n$ y $b \in \mathbb{R}$ tales que el hiperplano

$$\{\mathbf{x} \in \mathbb{R}^n : \mathbf{w}^\top \mathbf{x} + b = 0\}$$

separa las dos clases, esto es, los datos de clases opuestas están en lados opuestos del hiperplano.

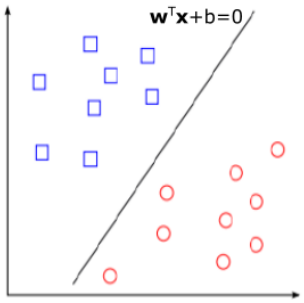

Figura 2.8: Datos e hiperplano $\mathbf{w}^\top \mathbf{x} + b = 0$

El hiperplano de la Figura 2.8 se construye a partir de los datos $\mathbf{x}_i$ y de sus etiquetas y_i. En este caso, la función $f(\mathbf{x})$ puede ser expresada como:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^\top \mathbf{x} + b),$$

donde $\text{sgn}(\cdot)$ es la función signo. De esta forma, el problema se reduce a encontrar $\mathbf{w} \in \mathbb{R}^n$ (*vector de pesos* que ajusta el modelo), y un escalar b (conocido como sesgo) tales que

$$y_i(\mathbf{w}^\top \mathbf{x} + b) > 0, \qquad \text{para } i = 1, 2, \ldots, p.$$

Pero, existen infinitos hiperplanos que satisfacen esta condición (ver Figura 2.7).

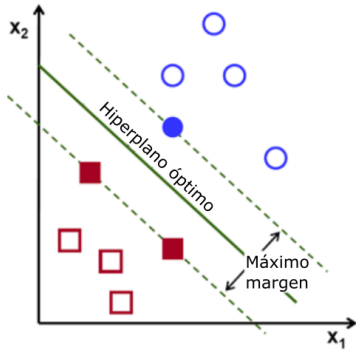

Figura 2.9: Hiperplano de separación óptimo. Máximo margen de separación.

El problema anterior se resuelve determinando el hiperplano de separación óptimo. Este hiperplano es aquel cuyo margen de separación entre los vectores de dos clases es máximo. La propiedad fundamental del hiperplano de separación óptimo es que este equidista del dato más cercano de cada clase [5] (ver Figura 2.9). En otras palabras, para determinar el hiperplano de separación óptimo se debe resolver el siguiente problema de optimización con restricciones:

$$\begin{aligned} \max_{\mathbf{w},b,\|\mathbf{w}\|=1} \quad & M \\ \text{sujeto a:} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq M, \qquad \text{para } i = 1, 2, \ldots, p, \end{aligned} \tag{2.1}$$

donde M es la distancia del hiperplano a cada lado. El margen máximo que se muestra en la Figura 2.9 tiene $2M$ unidades de ancho. Sin pérdida de generalidad, podemos suponer que $\|\mathbf{w}\| = 1/M$. Por lo que el problema (2.1) es equivalente a:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{sujeto a:} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \qquad \text{para } i = 1, 2, \ldots, p. \end{aligned} \tag{2.2}$$

El problema (2.2) se puede expresar en forma matricial como:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} \\ \text{sujeto a:} \quad & AX^\top \mathbf{w} + b\mathbf{y} \geq \mathbf{1}, \end{aligned} \tag{2.3}$$

donde $A = \text{diag}(y_1, y_2, \ldots, y_p) \in \mathbb{R}^{p\times p}$ es una matriz diagonal, $\mathbf{y} = (y_1, y_2, \ldots, y_p)^\top \in \mathbb{R}^p$ es el *vector de etiquetas*, $\mathbf{1} \in \mathbb{R}^p$ es el vector de unos, y $X = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \end{pmatrix} \in \mathbb{R}^{n\times p}$ es la matriz de atributos, cuyas columnas son los datos $\mathbf{x}_i$.

Utilizando los multiplicadores de Lagrange, el problema (2.3) puede expresarse en la siguiente forma, denominada *problema dual de Wolfe*,

$$\begin{aligned} \min_{\boldsymbol{\mu}} \quad & \frac{1}{2}\boldsymbol{\mu}^\top Q\boldsymbol{\mu} - \boldsymbol{\mu}^\top \mathbf{1} \\ \text{sujeto a:} \quad & \boldsymbol{\mu}^\top \mathbf{y} = 0, \\ & \boldsymbol{\mu} \geq 0. \end{aligned} \tag{2.4}$$

donde $Q = AX^\top XA$, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^\top \in \mathbb{R}^p$, y μ_i son los multiplicadores de Lagrange. El problema dual de Wolfe puede resolverse eficientemente, ya que es un problema cuadrático y existen en la literatura algoritmos eficientes para la optimización cuadrática.

De todo lo anterior, para obtener la función $f(\mathbf{x}) = \text{sgn}(\mathbf{x}^\top \mathbf{w}^* + b^*)$ que separe los p pares $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_p, y_p)$ de datos etiquetados, se debe realizar el siguiente procedimiento, que hemos denominados Procedimiento de Separación:

Procedimiento de Separación

1. Resolver el problema (2.4) para obtener los multiplicadores de Lagrange $\boldsymbol{\mu}^*$.

2. Calcular el vector $\mathbf{w}^*$ por:

$$\mathbf{w}^* = XA\boldsymbol{\mu}^* = \sum_{i=1}^{p} \mu_i^* y_i \mathbf{x}_i.$$

3. Calcular b utilizando la ecuación:

$$b^* = y_{i_{max}} - \mathbf{x}_{i_{max}}^\top \mathbf{w}^*,$$

donde $i_{max} = \underset{i=1,2,\ldots,p}{\text{argmax}}\{\mu_i\}$.

Los **vectores de soporte** son los datos $\mathbf{x}_i$ tales que sus multiplicadores de Lagrange μ_i^* son positivos. Generalmente los vectores de soporte representan una pequeña porción de todos los datos de entrenamiento y son aquellos datos que se encuentran más cercanos al hiperplano de separación óptimo. La mayoría de los algoritmos para resolver el problema (2.4), tienen como estrategia identificar los vectores de soporte para reducir el tamaño del problema durante el entrenamiento. Todo este procedimiento de cálculo está implementado de manera eficiente en algunas librerías de Python, en especial en la librería **Scikit-Learn**, de la cual hablaremos más adelante.

2.4.1.2. Datos No Linealmente Separables

Cuando los datos $\mathbf{x}_i$ no son linealmente separables, existe una función no lineal $\phi : \mathbb{R}^n \to \mathbb{R}^n$ tal que los datos definidos como

$$\mathbf{z}_i = \phi(\mathbf{x}_i), \quad i = 1, 2, \ldots, p$$

son linealmente separables [6]. Usando este hecho se puede encontrar un separador no lineal para los datos $\mathbf{x}_i$ con SVM aplicando la misma técnica usada en el caso del separador lineal, pero con los datos $\mathbf{z}_i$. En este caso, la función de clasificación viene dada por:

$$f(\mathbf{x}) = \text{sgn}(\phi(\mathbf{x})^\top \mathbf{w}^* + b^*),$$

donde $\mathbf{w}^*$ y b^* se obtienen resolviendo el problema (2.4) y aplicando el Procedimiento de Separación, tomando $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_p$ como datos de entrenamiento. La función ϕ, en general, no se conoce explícitamente, además, es poco práctico calcularla. En la práctica, se usan las funciones llamadas *núcleo*, $\mathcal{K} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, tales que

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z}), \qquad \text{para } \mathbf{x}, \mathbf{z} \in \mathbb{R}^n.$$

Con un núcleo $\mathcal{K}$ dado, se construye la matriz $Q = (q_{ij})$ del problema (2.4) como:

$$q_{ij} = y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j), \qquad \text{para } i, j = 1, 2, \ldots, p.$$

Por lo que, la función de clasificación en el caso no separable linealmente, es:

$$f(\mathbf{x}) = \operatorname{sgn}(\sum_{i=1}^{p} y_i \mu_i^* \mathcal{K}(\mathbf{x}, \mathbf{x}_i) + b^*),$$

donde

$$b^* = y_j - \sum_{i=1}^{p} y_i \mu_i^* \mathcal{K}(\mathbf{x}_j, \mathbf{x}_i), \qquad \text{para algún } j \text{ tal que } \mu_j^* > 0.$$

No toda función $\mathcal{K}$ es un núcleo. La funciones núcleo deben ser simétricas, esto es $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}(\mathbf{z}, \mathbf{y})$, además, la matriz $K = (k_{ij})$ definida como $k_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ debe ser simétrica semidefinida positiva[1]. A continuación se listan las funciones núcleos más usadas en las aplicaciones.

1. *Lineal*: $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$

2. *Polinomio de grado d*: $\mathcal{K}(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^\top \mathbf{z} + r)$, con parámetros $\gamma, r \in \mathbb{R}$.

3. *Base radial (RBF)*: $\mathcal{K}(\mathbf{x}, \mathbf{z}) = e^{-\gamma \|\mathbf{x} - \mathbf{z}\|^2}$, con parámetro γ $in\mathbb{R}$.

[1] Una matriz $A \in \mathbb{R}^{n \times n}$ es simétrica semidefinida positiva si $A^\top = A$ y $\mathbf{x}^\top A \mathbf{x} \geq 0$, para todo $\mathbf{x} \in \mathbb{R}^n$.

4. *Red neuronal (Sigmoid)*: $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \tanh(\gamma \mathbf{x}^\top \mathbf{z} + r)$, con parámetros $\gamma, r \in \mathbb{R}$.

El procedimiento de separación cuando los datos son no linealmente separables, además del caso cuando hay multiples clases, también está implementado en la librería **Scikit-Learn**.

En resumen, el objetivo de la SVM es idear un método de aprendizaje computacionalmente eficiente. Los hiperplanos optimizan el límite de generalización, por lo tanto el algoritmo es capaz de manejar grandes dimensiones ([7]). Algunas de las ventajas de las SVMs frente a otros algoritmos son las siguientes ([25]):

- Efectividad en espacios de grandes dimensiones.
- Sigue siendo efectivo incluso en casos donde el número de dimensiones es más grande que el número de ejemplos de entrenamiento.
- Usa un subconjuto de los datos de entrenamiento en la función de decisión (los vectores de soporte), por lo tanto es eficiente computacionalmente, en especial en el uso de la memoria.
- Es versátil: diferentes funciones núcleo pueden emplearse para definir función de clasificación. Los núcleos comunes son fáciles de personalizar.

2.5. Lenguaje de programación Python

Python es un lenguaje de programación multiparadigma, multiplataforma, dinámico e interpretado, es decir, se ejecuta directamente mediante un intérprete dejando a un lado la compilación. Actualmente, Python es administrado por

Python Software Foundation. El software posee una licencia de código abierto denominada Python Software Fondation License.

Python se caracteriza por su amplia aplicabilidad en diferentes campos del desarrollo de nuevas tecnologías como: Machine Learning, Data Science, Vision por Computador, entre otros. Python soporta diferentes librerías como OpenCV y Scikit-Learn, las cuales cuentan con extensa documentación acerca de sus aplicaciones e implementaciones. La versión que se utilizó para la implementación de este trabajo fue Python 3.1.

2.6. OpenCV

Open Computer Vision (OpenCV, por sus siglás en inglés) es una librería libre de visión artificial desarrollada originalmente por Intel. Especializada en la visión por computador y el procesamiento de imágenes cuenta con diferentes herramientas que permiten manipular la información dentro de una imagen. Para mayor información visitar https://opencv.org/. La versión que se utilizó para la implementación de este trabajo fue OpenCV 3.4.

Específicamente, las funciones de OpenCV empleadas en la implementación son:

- `cv2.imread`: permite llamar una imagen digital al ambiente de desarrollo empleado para así poder manipularla mediante todas las otras herramientas que ofrece la librería. [10]
- `img.shape`: lee y devuelve las dimensiones de una imagen digital en función del ancho y alto. [9]
- `cv2.resize`: redimensiona el ancho y alto de una imagen de acuerdo a los

parámetros de entrada especificados. [18]

- `cv2.GaussianBlur`: es un filtro que suaviza la definición de la imagen a través de un kernel Gaussiano. El filtrado gaussiano se realiza convolucionando cada punto de la matriz de entrada con un núcleo gaussiano y luego sumándolos a todos para producir la matriz de salida. [11]

- `cv2.cvtColor`: cambia el espacio de color de una imagen digital a otro. [20]

- `cv2.adaptiveThreshold`: el método adaptativetreshold binariza una imagen de acuerdo a un valor de umbral que cálcula para pequeñas regiones de la imagen digital. Así, el filtro gaussiano realiza una convolución con cada punto de la matriz generando una matriz de salida. El valor de umbral es calculado como la suma ponderada de los valores del vecindario donde los pesos son una ventana gaussiana. Las dimesiones del kernel representa el tamaño del vecindario de pixeles usados para calcular el valor umbral. [19]

- `cv2.dilate`: Dilate es un método de transformación morfológica. Las operaciones morfológicas aplican un elemento estructurante a la imagen de entrada y generan una de salida con características morfológicas diferentes. Dilate utiliza un kernel B el cual se escanea sobre la imagen, calcula el valor máximo de pixeles superpuestos por B y reemplazamos el píxel de la imagen en la posición del punto de anclaje con ese valor máximo. Como se puede deducir, esta operación de maximización hace que las regiones brillantes dentro de una imagen “ crezcan". [21]

- `cv2.findcontours`: definiendo contorno como una curva donde todos los puntos continuos tienen el mismo color o intensidad, este método encuentra los contornos definidos en una imagen, preferiblemente binarizada. Este método requiere diferentes parámetros de entrada que definen su modo de funcionamiento como le modo de recuperación de contorno y método de

aproximación de contorno. Ambos parámetros son explicados en las fuentes de documentación. [22]

- `cv2.boundingRect`: esta función calcula y devuelve el rectángulo delimitador de un contorno o un conjunto de píxeles de la misma intensidad y distintos de cero. Esta función cálcula un rectangulo recto, es decir, no considera ángulo de rotación por lo tanto el área del rectángulo no será mínima. Esta función devuelve las coordenadas de la esquina superior izquierda del rectángulo, su ancho y alto. [12]

- `cv2.rectangle`: esta función es una de las funciones de dibujo de la librería *OpenCV* y dibuja un rectángulo en la imagen, dadas unas coordenadas específicas, con color y ancho de línea específico. La función devuelve la imagen con dicho polígono dibujado. [17]

2.7. Scikit-Learn

Scikit-Learn por su parte, es una biblioteca con basto desarrollo en el aprendizaje automático, de software libre para el lenguaje de programación Python. Cuenta con varios algoritmos de regresión, clasificación y análisis de grupos como SVMs, Clustering, K-means, entre otros. Esta biblioteca inició como un proyecto de Google Summer of code por David Cournapeau en el 2007. Para mayor información y detalles acerca de la biblioteca dirigirse a http://scikit-learn.org/stable/.

CAPÍTULO III

MARCO METODOLÓGICO

3.1. Tipo de investigación

Considerando el problema, referido al diseño de un sistema de reconocimiento de matrículas vehiculares, la modalidad de investigación del presente proyecto es *Proyecto factible* de acuerdo el Manual de la UPEL. Adicionalmente, el diseño podrá ser integrado a prototipos de sistemas de vigilancia vehícular. “El Proyecto Factible consiste en la investigación, elaboración y desarrollo de una propuesta de un modelo operativo viable para solucionar problemas, requerimientos o necesidades de organizaciones o grupos sociales; puede referirse a la formulación de políticas, programas, tecnologías, métodos o procesos. El Proyecto debe tener apoyo en una investigación de tipo documental, de campo o un diseño que incluya ambas modalidades."[26]

3.2. Fases metodológicas

A continuación, se presentan las fases que se llevaron a cabo para el desarrollo del trabajo de grado:

3.2.1. Recopilación de información

Se llevó a cabo una investigación documental acerca de:

- Procesamiento de imágenes.
  - Imagen digital.
  - Espacios de color.
  - Mascaras derivativas discretas.
  - Binarización.
- Lenguaje de programación Python.
- OpenCV para procesamiento de imágen.
- Máquina de Soporte Vectorial.

Tomando en cuenta el estado del arte en procesamiento de imágenes con referencias técnicas apropiadas de la información disponible en la red.

3.2.2. Selección

Con base a la información obtenida, se seleccionaron las técnicas de procesamiento de imagen más adecuadas a las necesidades de la detección de la placa y segmentación de los caracteres. La selección se caracterizó por obtener técnicas que hayan sido probadas y optimizadas en diferentes experimentos, de código abierto y con la información necesaria para la compresión de sus conceptos. Se seleccionó la librería de código abierto OpenCV, debido a su amplia utilización en la visión por computador y procesamientos de imágenes. OpenCV es una librería con gran variedad de técnicas y aplicaciones, siendo una herramienta empleada en

diversos entornos e incluso en investigaciones recientes en el campo de la visión por computador. Tomando en cuenta lo anterior, se profundizó en las técnicas que permitirían el filtrado de la información de interés en una imagen de condiciones controladas. Por otro lado, se seleccionó la Máquina de Soporte Vectorial, siendo uno de los algoritmos más eficientes e importantes en el mundo del machine learning. Dada su capacidad y efectividad, es un algoritmo que ha participado en diferentes campos de desarrollo, entre los cuales también se encuentra la visión por computador. Considerando lo anterior, se decidió explorar su potencialidad en la clasificación de caracteres.

3.2.3. Evaluación de aplicabilidad

Con el fin de evaluar la factibilidad se diseñó un diagrama de flujo a fin de definir las instrucciones generales necesarias para lograr el procesamiento de imagen con los resultados deseados. Se realizaron pruebas preliminares para evaluar los resultados experimentales obtenidos con los esperados y así determinar su factibilidad.

3.2.4. Programación y diseño

Una vez seleccionadas las técnicas para el procesamiento de imágenes y el algoritmo para la clasificación y reconocimiento de patrones, se procedió a diseñar de manera modular la estructura del programa. En función de lo anterior, a partir del diagrama de flujo obtenido, se elaboró el pseudocódigo y, de esta manera, se describieron las funciones de cada una de las etapas que componen el proyecto.

3.2.5. Implementación y pruebas

Por último, se procedió a implementar de manera modular toda la lógica necesaria para cada una de las etapas descritas en el pseudocódigo en el lenguaje Python. Se realizaron experiencias computacionales orientadas a la verificación del sistema de reconocimiento de matrículas. En primera instancia, se diseñaron pruebas dirigidas a monitorear el desempeño y la funcionalidad del programa, donde se tomaron acciones a fin de ajustar el algoritmo para obtener los resultados esperados.

CAPÍTULO IV

DEFINICIÓN Y DESCRIPCIÓN DEL SOFTWARE

En el presente capítulo se describe el proceso de desarrollo del proyecto, donde se detallan los procedimientos llevados a cabo a fin de cumplir los objetivos planteados.

4.1. Arquitectura de software

El software se basó en una arquitectura estructural. Para cumplir la condición anterior, dado el problema que se desea resolver a través de un algoritmo, se divide dicho algoritmo en módulos siguiendo los principios de diseño de descomposición por refinamiento sucesivo, creación de jerarquía modular y elaboración de módulos independientes.

Los procesos descritos fueron diseñados para trabajar de forma modular e independientes, cada uno con una funcionalidad específica. Este proyecto se dividió en dos grandes módulos: procesamiento de imágenes y reconocimiento de caracteres.

Se seleccionó esta arquitectura ya que la solución propuesta para la problemática planteada se puede describir en dos etapas diferentes, cuyas funcionalidades son independientes entre sí, pudiéndose definir las interacciones entre las entradas y salidas. En la Figura 4.1 se puede observar un esquema general del proyecto

expresada en módulos.

Figura 4.1: Módulos del sistema

4.2. Componentes del sistema

Se busca recrear de manera computarizada el proceso de observar una placa y leer su información. Por ello, se cuenta con una serie de herramientas y técnicas desarrolladas en Machine Learning y Visión Computarizada. El procesamiento de imágenes con OpenCV, el filtrado de la información y por último la SVM imitará el reconocimiento de caracteres, cumpliendo así con los objetivos planteados.

Dado lo anterior, se necesitará: un computador dónde se ejecutará el algoritmo del proyecto y una base de datos compuesta por un conjunto de imágenes de caracteres que conformarán el conjunto de entrenamiento. Dado que el presente proyecto consiste en el diseño de software, no contempla la captura de escenas. Las pruebas se realizarán con tomas previamente capturadas en condiciones controladas tanto en iluminación como en la posición del objeto. Con lo anterior expuesto, el único componente físico que se requiere es un computador.

4.3. Computador.

Este proyecto, al tratarse de un diseño de software, requiere únicamente de un computador con las capacidades de computo suficientes para manejar los

procedimientos lógicos y matemáticos que componen al algoritmo. Por lo tanto, no es necesario el uso de equipos especializados para su desarrollo.

El computador es una máquina Toshiba Satellite C55 Series. El sistema operativo es Microsoft Windows 10. Las principales características del computador se describen en la Tabla 4.1:

Tabla 4.1: Especificaciones del computador Toshiba Satellite C55 Series

Alimentación	19V
Procesador	AMD QUAD-CORE A6
Memoria RAM	16GB
Tipo de sistema operativo	64 bits

4.4. Base de datos

En general, existen números osas bases de datos de matrículas vehículares de diferentes países del mundo y estas incluyen fotos de la placa completa. Es por ello que no es posible emplearlas para el entrenamiento de la SVM tomando en cuenta los objetivos del presente proyecto.

Para cumplir con los requerimientos planteados, es fundamental contar con una base de datos compuesta por imágenes de todas las letras y números del abecedario español individualmente que participen en la identificación de la matrícula. Es decir, cada imagen debe contener solo un tipo de carácter. Por lo tanto, para entrenar la SVM fue necesario construir dicha base de datos.

Se construyó una base de datos que cuenta con 1496 imágenes de caracteres de letras y números. Para su construcción se recopiló al menos 250 fotos de placas vehículares de origen Venezolano y se procesaron utilizando herramientas provistas por la librería OpenCV, a fin de obtener la información de interés.

Las técnicas empleadas para su construcción se basan en la detección de contornos a partir de la umbralización de la imagen. Empleando diferentes funciones del procesamiento de imágenes y lógica condicional en la programación, se detectaron los caracteres de interés y se extrajeron recortando la foto. Las medidas adecuadas para su extracción se obtuvieron a partir de la medición del aspect ratio (Relación entre la altura y ancho del contorno rectangular) y el área ocupada por el carácter en la foto, conservando una distancia prudencial entre los bordes y el carácter . El resultado de la construcción resultó en imágenes de 25 x 45 px, en blanco y negro.

El almacenamiento de cada uno de los ejemplos de entrenamiento se realizó en un archivo .csv, el cual permitió registrar en forma de matriz todos los ejemplos de entrenamiento. Para generar dicha matriz se redujo aún más el tamaño de la imagen que contiene cada muestra, resultado en una imagen de 27 x 15 px, lo cual se traduce en 405 pixeles por muestra. De esta forma, cada fila de la matriz contenida en el archivo .csv corresponde a un ejemplo de entrenamiento individual. La última columna de una fila corresponde a la etiqueta que identifica el carácter de esa imagen. Para la escritura de este archivo se empleó la librería csv provista por Python.

En el diseño, una de las mayores dificultades fue la construcción de la base de datos, ya que fue necesario ajustar diferentes parámetros en el proceso de extracción de las muestras. Por otro lado, la adquisición de tantas placas Venezolanas representó una gran dificultad, dado que fue necesario tomar todas las fotos y no se disponía el acceso a tantas placas de manera inmediata o en la red. Es por esto, que se cuenta con un número limitado de ejemplos de entrenamiento en comparación a los números comunes, que por lo general superan el millón de muestras.

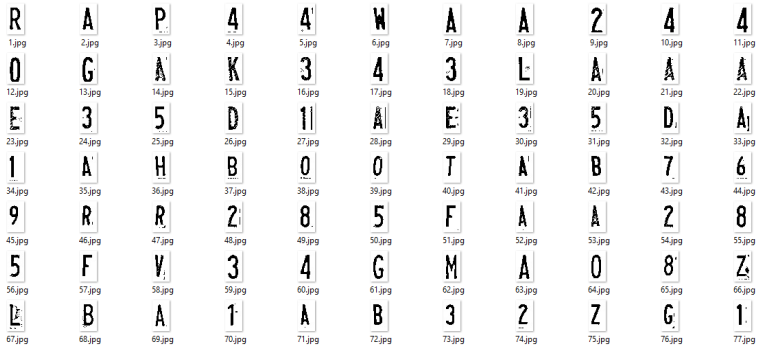

Figura 4.2: Muestras de imágenes del conjunto de entrenamiento creado.

La cantidad de muestras extraídas por cada clasificación se muestra en la Tabla 4.2.

Tabla 4.2: Clasificación y etiquetas de cada carácter.

Clasificación	Etiqueta	Cantidad extraída
0	0	46
1	1	43
2	2	71
3	3	54
4	4	66
5	5	50
6	6	72
7	7	65
8	8	58
9	9	70
A	10	297
B	11	65
C	12	38
D	13	51
E	14	37
F	15	43
G	16	52
H	17	26
I	18	18
J	19	10
K	20	28
L	21	13
M	22	46
N	23	16
Ñ	-	-
O	24	28
P	25	12
Q	26	0
R	27	18
S	28	8
T	29	5
U	30	13
V	31	23
W	32	13
X	33	18
Y	34	12
Z	35	7

4.5. Software

El proyecto tiene como objeto diseñar un sistema que permita el reconocimiento de la matrícula de un vehículo en una imagen estática, por lo tanto los videos no están contemplados en este diseño.

Dado que el objetivo es reconocer la información contenida en la matrícula de un vehículo, el software debe tener la capacidad de extraer los caracteres de interés contenidos en la placa mediante el procesamiento de imágenes y luego clasificar cada uno de ellos.

Figura 4.3: Proceso de segmentación de caracteres.

Para lograr lo anterior, se empleó Python y la librería OpenCV de Intel.

Así, el software se divide en los siguientes tres módulos:

- Detección de la placa través de técnicas de procesamiento de imágenes.
- Segmentación de los caracteres mediante el procesamiento de imágenes.
- Clasificación de los caracteres empleando Machine Learning.

4.5.1. Detección de la placa

El módulo de detección (ver Apéndice III) de la placa se define como un sistema dónde se ingresa una imagen frontal o posterior de un vehículo y se obtiene la placa contenida. Se puede describir este proceso en las siguientes etapas:

- Localización.
- Extracción.

De esta forma, es necesario emplear técnicas y herramientas que permitan detectar la localización de la placa mediante el procesamiento imagen. Se diseñó un procedimiento que sigue una serie de pasos con el fin de filtrar la información y detectar la placa en una captura de imagen. Para ello se emplearon funciones de la biblioteca OpenCV. Adicionalmente, se establecieron parámetros para la captura de imagen que dará entrada al sistema, las cuales se contemplan: distancia de 2 metros entre el vehículo y la cámara fotográfica, distancia de 1 metro de altura respecto al suelo y la cámara fotográfica, con el vehículo centrado en el medio del marco de captura y tomar la fotografía con el flash activo. Se describe dicho proceso en el siguiente diagrama de flujo:

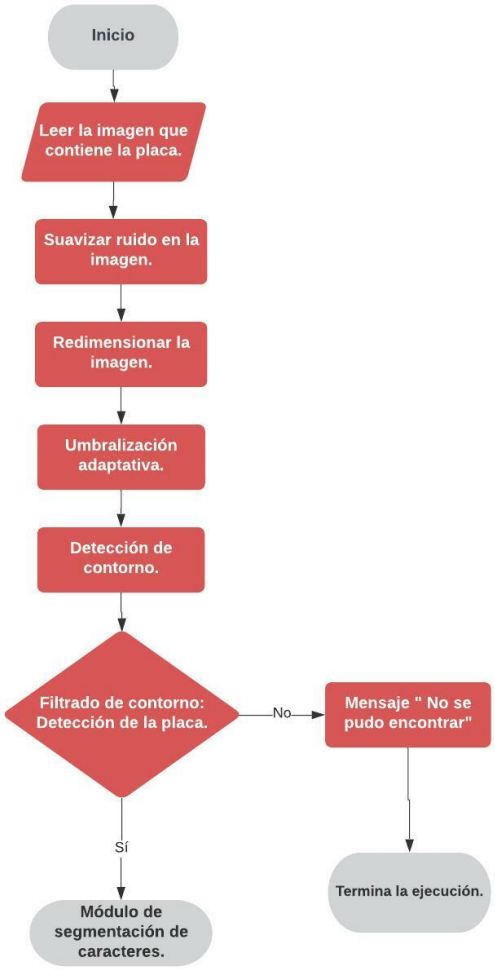

Figura 4.4: Proceso de detección de la placa.

Así, se describe el procedimiento para la detección y extracción de la siguiente forma:

1. **Suavizado de ruido**: En primer lugar, es necesario suavizar el ruido de la imagen que contiene la placa, limpiándola de características no deseadas. En la placa, particularmente, se halla rodeada de mucho ruido debido que se encuentran muchos objetos alrededor del carro, por lo que es muy importante realizar este procedimiento. Para realizar esto se empleó con un núcleo

gaussiano mediante la función `cv2.GaussianBlur`.

2. **Redimensionamiento de imagen**: La imagen que contiene la placa es redimensionada para reducir el mapa de pixeles dónde se estima que estará la placa. Esto con el fin de facilitar el trabajo de la posterior detección de contornos. En las condiciones controladas presentadas se establece que el vehículo se debe encontrar en el medio del marco de captura, por lo que aproximadamente el 20 % de los laterales de la fotografía se considera ruido. Así, se elimina el 20 % de los laterales y este redimensionamiento se logra empleando fundamentalmente la función `cv2.rezise`.

3. **Umbralización**: La umbralización se realiza con el fin de binarizar la imagen. Para la detección de contornos es imprescindible que la binarización sea inversa. De esta forma, los contornos detectados se resaltan en blanco y el fondo en negro. En este proyecto se empleó la umbralización adaptativa `cv2.adaptativetresholding`. La *umbralización adaptativa* es una función que a partir de los parámetros ingresados binariza considerando pequeñas regiones de la imagen, a diferencia de una umbralización estandar. Por lo tanto, la umbralización adaptativa cálcula el valor umbral adecuado para cada región, mediante un núcleo gaussiano. La importancia de su capacidad adaptativa radica en que factores externos como la variación de la iluminación pueden afectar significativamente el proceso de binarización.

4. **Detección de contorno**: La detección de contornos se aplica sobre la imagen umbralizada, donde ya la mayoría del ruido ha sido filtrado. En este paso, se utilizó la función `cv2.findcontours` la cuál retorna una variable que contiene los contornos detectados. La detección de los contornos en la imagen depende de los parámetros ingresados en la función. En este caso, se detectan los contornos cerrados en la imagen.

5. **Filtrado de contorno**: La filtración de contorno consiste en seleccionar

los contornos de interés. Se empleó lógica condicional a fin de verificar si el contorno califica o no. Para ello, se estimó el área y el aspect ratio que ocupa un caracter dentro de la placa. Se ponderaron los valores de ancho y alto de varias muestras para obtener un valor umbral. Para obtener las coordenadas de los caracteres se utilizó la función `cv2.boundingrectangle`, la cual permite encerrar en rectángulos los contornos detectados y devuelve los valores que permiten definir su posición en la imagen.

6. **Extracción de la placa**: Así, se extrajo recortando la imagen con funciones básicas de Python, que consiste en extraer un segmento de la imagen a partir de las coordenadas indicadas.

4.5.2. Segmentación de los caracteres.

El módulo de segmentación (ver Apéndice IV) se encarga de detectar, segmentar y extraer los caracteres de interés dentro de la placa. La secuencia de pasos a seguir en este módulo es muy similar al de módudulo anterior *Detección de la placa*, pues se requiere un tratamiento para la extracción de cada caracter. Considerando lo anterior, se detallarán las diferencias de importancia.

Esto se realizó mediante técnicas de procesamiento de imágenes mediante la librería la librería OpenCV. Se describe dicho proceso en el siguiente diagrama de flujo:

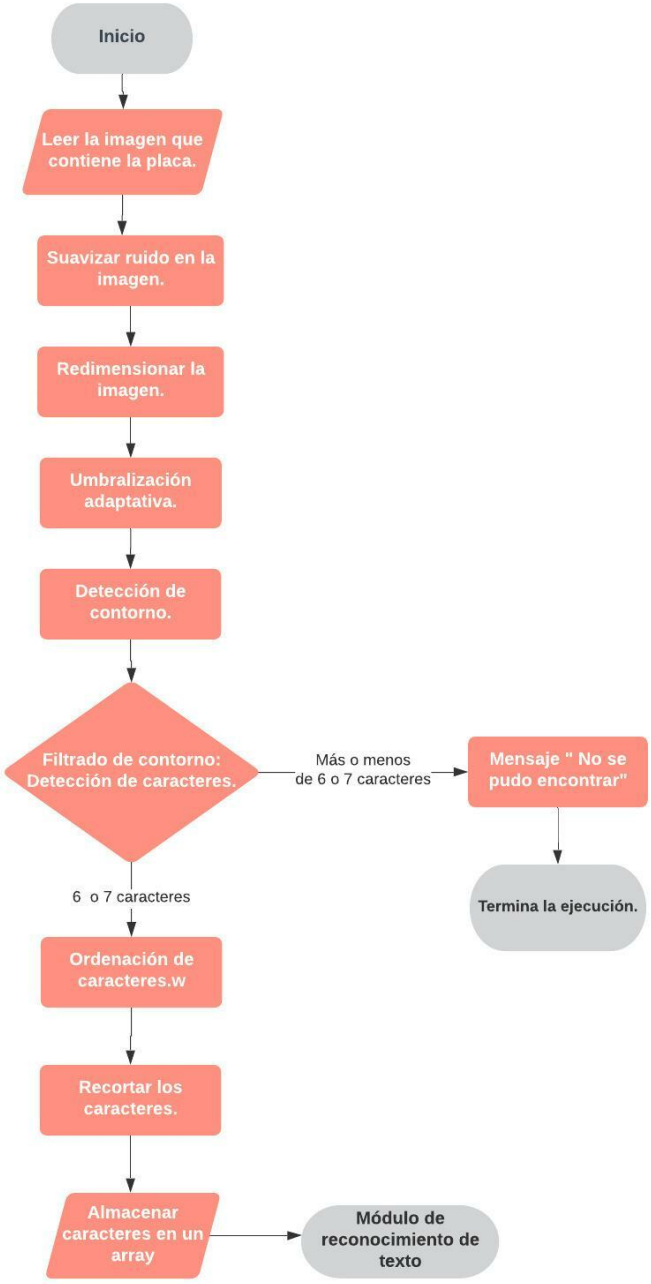

Figura 4.5: Procedimiento de segmentación de los caracteres de la matrícula.

1. **Suavizado de ruido:** Con el fin de eliminar las características no deseadas en la placa se emplea el suavizado de ruido con un kernel gaussiano mediante la función `cv2.GaussianBlur`.

2. **Redimensionamiento de la imagen:** La imagen que contiene la placa es redimensionada hasta obtener 150px de altura, posteriormente permitirá que el algoritmo pueda detectar y extraer los caracteres en segmentos de 25 x 45 px. En general, los ejemplos de entrenamiento son contenidos en imágenes de dimensiones pequeñas considerando que cada pixel que la compone es una característica de entrada para el algoritmo, lo cual significa un importante costo computacional. Para lograr el redimensionamiento se empleó fundamentalmente la función `cv2.rezise`.

3. **Umbralización de la imagen:** Se binariza la imagen inversamente mediante la Umbralización adaptativa empleando un núcleo Gaussiano y se implementa utilizando la función `cv2.adaptativetresholding`.

4. **Detección de contorno:** Se detectan los contornos cerrados de la imagen binarizada mediante la función `cv2.findcontours` la cuál retorna una variable que contiene los contornos detectados.

5. **Filtración de contornos:** Empleando la lógica condicional se filtran los contornos de interés en la imagen binarizada. Para ello, se estimó el área y el aspect ratio que ocupa un caracter dentro de la placa. Se ponderaron los valores de ancho y alto de varias muestras para obtener un valor umbral. Así, se definió una relación porcentual del área del carácter respecto al área de la placa y un bias que permita cubrir la mayoría de las variaciones. Para obtener las coordenadas de los caracteres se utilizó la función `cv2.boundingrectangle`, la cual permite encerrar en rectángulos los contornos detectados y devuelve los valores que permiten definir su posición en la imagen.

6. **Recorte de los caracteres:** Con las coordenadas de los caracteres, se recortaron de la imagen con un margen de provisión adecuado. Así, se aseguró contener completamente cada caracter en un segmento. Para la

recortar la imagen se usaron las funciones básicas de Python, que consiste en extraer un segmento de la imagen a partir de unas coordenadas dadas. Así mismo, los caracteres son ordenados mediante la función sorted de Python, empleando una Key a través del cual se logran ordenar de izquierda a derecha.

7. **Escritura en computador de los caracteres:** Una vez obtenido todos los segmentos se escriben en el computador utilizando `cv2.imwrite`, con formato .jpg.

4.5.3. Clasificación de caracteres.

La etapa de clasificación de caracteres se lleva a cabo mediante el modelo entrenado. En el Capitulo 2 en la Sección 2.4 se describió como se puede obtener una etiqueta a partir de los parámetros obtenidos y la nueva observación. Esta etiqueta es la predicción de clase para la imagen de entrada que se está clasificando. En conclusión, este módulo se encarga de predecir la clasificación de cada nuevo carácter en base al modelo obtenido del entrenamiento (ver apéndice V en el Anexo).

El clasificador es un método para determinar la clase más probable de una entrada desconocida con respecto a un número de ejemplos de todas las clases. Este conjunto de ejemplos se denomina conjunto de entrenamiento.

El primer paso en el proceso de clasificación es la extracción de características de una entrada, esto se traduce en expresar cada instancia u objeto como un vector de medidas. Cuando las imágenes están siendo clasificadas, usualmente el vector es extraído a partir de las intensidades de los píxeles. Generalmente se lleva a cabo un paso de reducción de características, ya que disminuye los costos computacionales. Es importante considerar que la reducción no debe provocar una

pérdida significativa de la información. Las medidas obtenidas son denominadas características y pueden ser reales, enteras o categóricas.

Una vez extraídas las características se introducen al algoritmo clasificador el cuál trabaja en dos naturalezas distintas: supervisado o no supervisado. En este proyecto se utilizará la clasificación supervisada, ya que se conoce la relación entre la entrada y la salida. En la clasificación supervisada, durante el entrenamiento conocemos la entrada y cuál debería ser su clasificación. En la clasificación no supervisada no se conoce la relación entre la entrada y la salida.

Para la extracción de características se emplearon las diferentes herramientas de manejo de matrices que ofrece Python para convertir esa entrada en un vector plano que pudiera leer el algoritmo de clasificación. En el Figura 4.6 se puede observar el diagrama del proceso de clasificación.

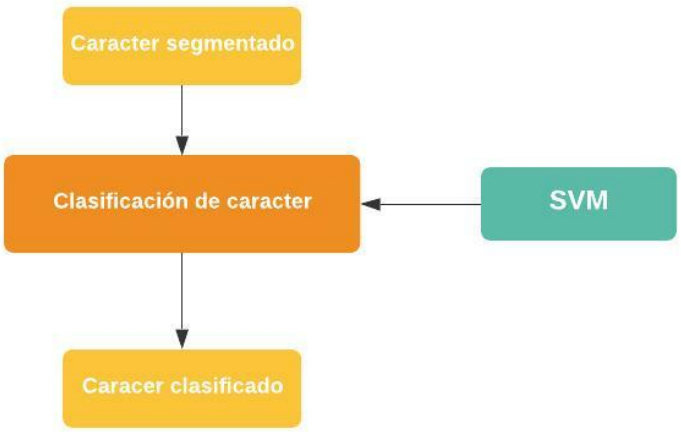

Figura 4.6: Diagrama de proceso de clasificación

4.6. Módulo de entrenamiento

Dado un conjunto de muestras, se etiquetan las clases a manera de construir un modelo con la capacidad de predecir una nueva entrada desconocida. En este caso, el modelo debe tener la capacidad de distinguir entre 36 clases diferentes, entre las cuales se encuentran las letras del abecedario español (excluyendo la Ñ) y los números del 0 al 1. En la Tabla 4.2 se puede observar cada etiqueta o clase asignada para cada carácter.

De esta forma, se lleva a cabo el entrenamiento de la SVM utilizando un núcleo Lineal empleando las herramientas proporcionadas por la librería SKlearn SVM. El entrenamiento de una SVM consiste en encontrar los parámetros que generan el híperplano que cumplen con el máximo margen de separación entre las clases. Para la multiclasifidora se empleó la función de decisión `ovr` el cuál consiste en que cada clase se enfrenta hacia el resto de las clases para encontrar el híperplano que mejor las separa.

El modelo obtenido es el resultado de ajustar los parámetros para todos los ejemplos de entrenamiento que conforman la base de datos. Cada ejemplo de entrenamiento representa una fila de la matriz que conforma la base de datos. Así, cada fila está compuesta por las características de la imagen y su etiqueta. Este modelo es el utilizado para la clasificación de caracteres, el módulo final del proyecto.

Así, el módulo de entrenamiento se encarga de encontrar el modelo que mejor se ajusta a la base de datos proporcionada. El desempeño de este modelo es evaluado en la etapa de validación.

CAPÍTULO V

PRUEBAS Y RESULTADOS

Una vez diseñados e implementados los módulos de detección de placa, detección de caracteres y reconocimiento de caracteres, se realizaron diferentes pruebas que permiten evaluar el desempeño del sistema y validar su funcionamiento, así como definir las condiciones bajo los cuales se obtiene los mejores resultados en cuánto a detección y clasificación. Todas las pruebas presentadas a continuación se realizaron con placas Venezolanas.

5.1. Desempeño del sistema

Para medir el desempeño del sistema en cada uno de sus módulos así como en su funcionamiento global, se probaron 2 escenarios: reconocimiento de caracteres que participaron en la base de datos y reconocimiento de caracteres que no participaron en la base de datos.

En un sistema real, la entrada al sistema es capturada por un dispositivo instalado en posición y espacio fijo, por lo que todas las entradas se encuentran sometidas a las mismas condiciones controlables. Por lo que una vez ajustada la resolución de la cámara se cuenta con entradas que tienen características muy similares.

Para la realización de estas pruebas, las entrada al sistema fueron imágenes de

diferentes resoluciones, ángulos de captura y distancias, dado que no se disponía de un sistema físico real que capture entradas uniforme. Esto sucede especialmente en los ejemplos de entrenamiento que componen la base de datos. Es importante recalcar que el cambio de resolución en una captura podría significar una pérdida de información importante que incide directamente en la correcta detección y reconocimiento del carácter, ya que su vector de característica es directamente afectado. Por otro lado, la variación del ángulo de captura genera distorsión en las imágenes por lo que también incide directamente en la clasificación de caracteres, producto de la variación en el vector de medida.

5.1.1. Reconocimiento de caracteres que forman parte de la base de datos

En esta prueba no fue posible evaluar el módulo de detección de placa, ya que las muestras disponibles son directamente una foto de la placa sin considerar el resto del automóvil. Por lo que se evaluará el desempeño de la detección de carácter y clasificación de los mismos.

Se tomaron aleatoriamente 50 de las 250 placas que se obtuvieron para la construcción de la base de datos, a fin de evaluar el rendimiento de detección y reconocimiento de caracteres ya conocidos por el algoritmo. En la Figura 5.1 se muestran algunas de las placas que en la construcción de la base de datos.

Figura 5.1: Algunas de las placas que participaron en la construcción de la base de datos.

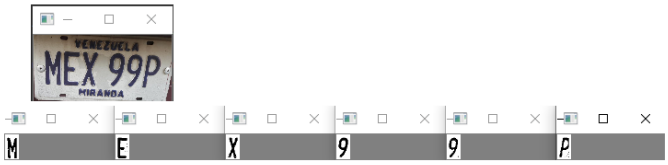

Figura 5.2: Procesamiento de la placa 82 de la base de datos.

Figura 5.3: Procesamiento de la placa 161 de la base de datos.

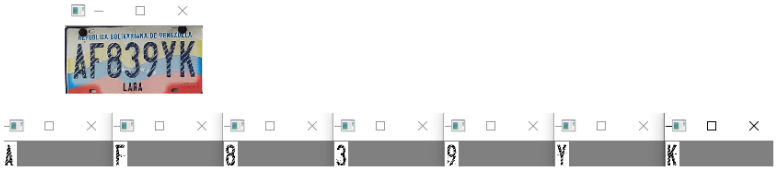

Figura 5.4: Procesamiento de la placa 229 de la base de datos.

El módulo reconoció en su mayoría los caracteres de las placas realizando correctamente las extracciones como se puede observar en las Figuras 5.2, 5.3 y 5.4. Se observa en la parte superior de las figuras la placa a la cual corresponde la extracción.

Figura 5.5: Procesamiento de la placa 84 de la base de datos

Figura 5.6: Procesamiento de la placa 179 de la base de datos

Figura 5.7: Procesamiento de la placa 243 de la base de datos

Por otro lado, en las Figuras 5.5, 5.6 y 5.7 se muestran algunos de los casos dónde el Módulo de detección de caracteres no funciona correctamente y clasifica como fallo. En las placas Venezolanas se pueden hallar de 6 a 7 caracteres dependiendo del año en que fueron fabricadas. En consecuencia, si el módulo de detección de caracteres no detecta al menos 6 contornos que cumplan con las características arroja fallo en la detección general. Experimentalmente se encontró que este tipo de errores se originan en la detección de contornos del sistema y usualmente se debe a una combinación de factores externos e internos.

Los factores externos que indicen significativamente en la detección de contornos frecuentemente es la iluminación, ángulo de captura y condiciones de preservación del cuerpo físico matrícula. Los factores internos hacen referencia a los parámetros o procedimientos del procesamiento de imagenes. Una solución para reducir al mínimo los factores externos es implementar un sistema físico que permita tomar capturas a una misma resolución, con un ángulo de captura fijo y condiciones de iluminación similares.

En la Figura 5.6, se muestra un caso que a simple vista se puede descartar a los factores externos como origen al problema. Aún así, se obtuvo un fallo en la detección general. El sistema realmente no es capaz de ver propiamente la matrícula, este compara matemáticamente dos valores y toma una decisión en base a eso. Entonces, esta detección se origina debido a un factor interno. Para realizar la detección de contornos el sistema recorre la imagen mediante un núcleo en el proceso de umbralización. A pesar que la Umbralización es adaptable por zonas, el tamaño del núlcoe con el que recorre la imágen no lo es. En este proyecto el tamaño del núcleo se estableció en 15 pixeles. Una posible solución podría ser considerar diferentes tamaños para obtener mejores resultados. Adicionalmente, en el reconocimiento de caracteres el algoritmo clasificó la K como una A. Este error puede deberse a que en la base de datos exista un ejemplo que los valores introducidos al modelo por medio de la extracción de características coincidan matemáticamente con los de la letra K de esta placa. Este último se soluciona depurando la Base de Datos de los ejemplos que puedan ocasionar este tipo de errores.

En la Figura 5.7, los inconvenientes se presentan en forma similar al anterior, pero es importante destacar que en la extracción se puede observar que los contornos superan las medidas límites de los cuadros como es el caso particular de la letra D y E. A pesar de lo anterior, el sistema de reconocimiento fue capaz

de reconocer la letra D, sin embargo, la letra E la clasificó con una F. Este problema de reconocimiento no se origina en el modelo del algoritmo SVM sino en la extracción de caracteres. Su solución recae en mejorar el rendimiento de extracción de caracteres.

Se presenta la Tabla 5.1, en la cual se reporta la placa, caracteres detectados y reconocidos a fin de evaluar su desempeño.

En la Tabla 5.2 se observan los resultados obtenidos de 50 muestras que participaron en el entrenamiento de la base de datos. En este experimento se obtuvo una tasa de éxito de 84.4 % en la detección de caracteres para las muestras dónde el algoritmo fue capaz de detectar en su totalidad la placa.

El sistema falló en la detección de caracteres de 7 de 50 placas, dado que el *Módulo de segmentación de caracteres* no logró encontrar al menos 6 caracteres dentro de la matrícula o, por el contrario, reconoció más de 7 contornos. Cuando esto sucede el sistema clasifica esto como un error. Es posible que el algoritmo reconozca un contorno que matemáticamente se parezca al área y aspect ratio establecidos para los caracteres de una placa, pero no lo sea. Debido a lo anterior, 7 placas fueron clasificadas como error. Las mismas no están incluidas en la Tabla 5.1.

Sin embargo, aunque estas 7 placas fueron clasificadas como error el *Módulo de segmentación de caracteres* si fue capaz de detectar algunos de los caracteres contenidas en estas. Tomando en cuenta el número total de caracteres detectado, incluso en las placas clasificadas como error el rendimiento del *Módulo de segmentación de caracteres* es de 98,01 %. Mientras que en el rendimiento de reconocimiento de caracteres obtuvo una tasa de éxito 98,90 %.

Tabla 5.1: Lista de placas que participaron en la prueba.

Placa ID	Número de Placa	Detectados	Reconocidos
82	MEX99P	MEX99P	MEX99P
45	AF689NV	AF689NV	AF689NV
244	AJ294OA	AJ294OA	AJ294OA
162	DCN07P	DCN07P	DCN07P
121	A86DC6G	A86DC6G	A86DC6G
248	AB275UN	AB275UN	AB275UN
161	AA192SU	AA192SU	AA192SU
192	A05DJ4G	A05DJ4G	A05DJ4G
175	DCE96B	DCE96B	DCE96B
88	AFD06Y	AFD06Y	AFD06Y
62	AGH56W	AGH56W	AGH56W
204	**AA251MD**	**AA25MD**	**AA25HD**
241	AB076DF	AB076DF	AB076DF
133	AG366GG	AG366GG	AG366GG
197	DBM43M	DBM43M	DBM43M
90	AH534HM	AH534HM	AH534HM
79	BBM29K	BBM29K	BBM29K
84	AD591RM	AD59RM	AD59RM
238	A00CG8V	A00CG8V	A00CG8V
160	MBF06L	MBF06L	MBF06L
104	AB747BM	AB747BM	AB747BM
217	AF283VG	AF283VG	AF283VG
74	MFJ88M	MFJ88M	MFJ88M
108	AI397IG	AI397IG	AI397IG
249	AK179GA	AK179GA	AK179GA
65	AH118WG	AH118WG	AH118WG
94	AB302DE	AB302DE	AB302DE
219	ABI37X	ABI37X	ABI37X
242	MAC24A	MAC24A	MAC24A
87	AB789PA	AB789PA	AB789PA
92	A66CA7K	A66CA7K	A66CA7K
190	AA473BU	AA473BU	AA473BU
56	AF834EG	AF834EG	AF834EG
243	**AD476EM**	**D476EM**	**D476FM**
54	AFH23X	AFH23X	AFH23X
137	AB242AA	B242AA	B242AA
181	AA096TF	AA096TF	AA096TF
174	ABI50J	ABI50J	ABI50J
228	AL092BA	AL092BA	AL092BA
114	MEX15L	MEX15L	MEX15L
251	AH850DM	AH850DM	AH850DM
126	AE192BA	AE192BA	AE192BA
179	**AB218KN**	**AB218K**	**AB218A**
229	AF839YK	AF839YK	AF839YK
234	AA453UD	AA45UD	AA45UD

El rendimiento por caracter se presenta en la Tabla 5.2. En las Figuras 5.27, 5.9 y 5.10, se muestran gráficos circulares de la tasa de éxito y error del reconocimiento de los caracteres E, M y K.

Tabla 5.2: Desempeño con muestras de la base de datos.

Carácter	Tasa de éxito	Detectados	Reconocidos
0	1.0	12	12
1	1.0	7	7
2	1.0	14	14
3	1.0	11	11
4	1.0	11	11
5	1.0	10	10
6	1.0	14	14
7	1.0	12	12
8	1.0	12	12
9	1.0	15	15
A	1.0	49	49
B	1.0	18	18
C	1.0	6	6
D	1.0	13	13
E	0.857	7	6
F	1.0	10	10
G	1.0	12	12
H	1.0	6	6
I	1.0	4	4
J	1.0	4	4
K	0.8	5	4
L	1.0	2	2
M	0.933	15	14
N	1.0	3	3
O	1.0	1	1
P	1.0	3	3
Q	0	0	0
R	1.0	1	1
S	1.0	1	1
T	1.0	1	1
U	1.0	4	4
V	1.0	3	3
W	1.0	2	2
X	1.0	4	4
Y	1.0	2	2
Z	0	0	0

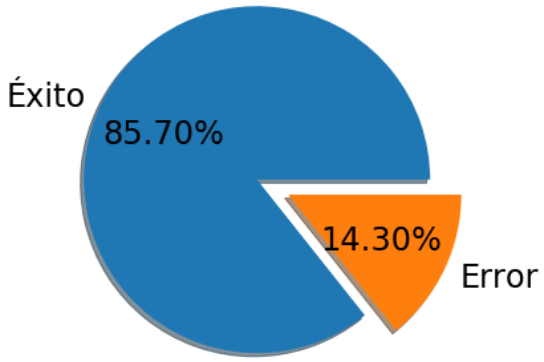

Figura 5.8: Tasa de éxito de reconocimiento de la letra E

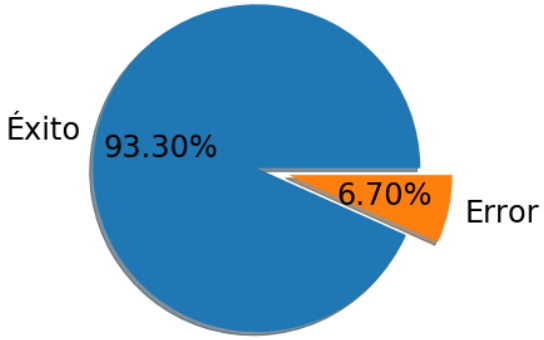

Figura 5.9: Tasa de éxito de reconocimiento de la letra M

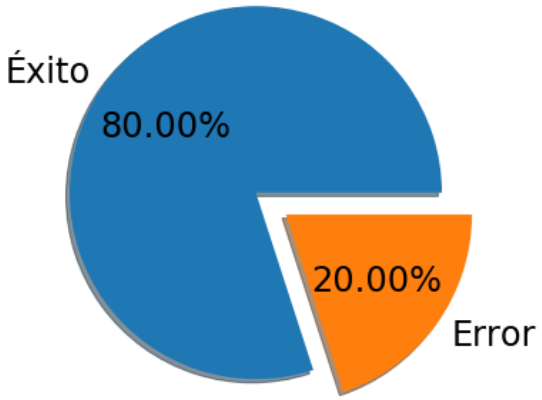

Figura 5.10: Tasa de éxito de reconocimiento de la letra K

A continuación se presentan las placas que fueron clasificadas como error:

Tabla 5.3: Placas que fueron clasificadas como error.

Placa ID	Número de Placa	Detectados
97	FAG71U	FAG71
96	RAG59S	RAG59S
154	AG722PA	AG7222PA
168	AE830GD	A830G
170	BCA770	CA770
61	MFC40N	FC40N
144	BBF29K	BBF29K
204	AA251MD	AA25D

Figura 5.11: Error en la placa 97

Figura 5.12: Error en la placa 96

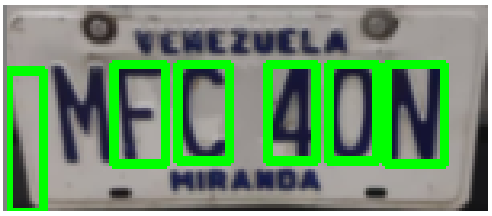

Figura 5.13: Error en la placa 61

Figura 5.14: Error en la placa 144

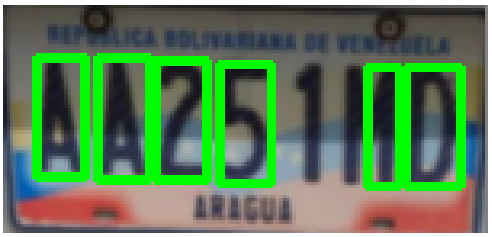

Figura 5.15: Error en la placa 204

En las Figuras 5.12 y 5.14 se detectaron todos los caracteres de la placa y además otro contorno con características matemáticas similares a las de un caracter fue detectado, siendo un total de 8 contornos en estas placas. Dado que el sistema es diseñado para detectar un máximo de 7 caracteres, la detección y extracción del sistema ve este caso como un error. Por otro lado, tener un contorno que no contiene un caracter genera una incongruencia matemática en el procesamiento de reconocimiento lo cual concluye en un error. Es por eso que la placa 61 de la figura 5.13 es clasificada como error a pesar de tener solo 7 contornos. Este tipo de errores se puede corregir estableciendo un margen, esto es, si las coordenadas del contornos se encuentran por fuera de este margen entonces no se tomaron en cuenta para la detección y extracción.

5.1.2. Reconocimiento de caracteres que no forman parte de la base de datos

Para esta prueba se consideraron las siguientes condiciones: 2 metros de distancia respecto a la placa del automóvil y un metro de altura respecto al suelo. Así mismo, las capturas se realizaron con iluminación flash del celular para mejorar la detección de la placa, considerando que de iluminación eran tenues ya que los vehículos se encontraban en un estacionamiento. Por lo tanto, teniendo en cuenta estas variables controladas se obtuvo la captura de 10 automóviles diferentes para evaluar el desempeño de detección de placa, detección de caracteres y reconocimiento de caracteres.

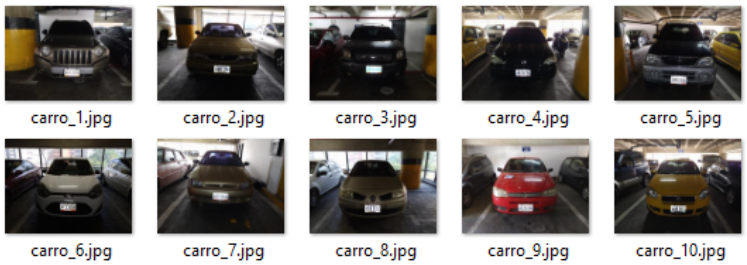

Figura 5.16: Placas que no participaron en la base de datos

El primer módulo por evaluar es el *Módulo de detección de placa*. Como se explicó en capítulos anteriores, el sistema buscará dentro de la imagen un contorno que cumpla con la relación de ancho y altura de una placa a la distancia de 2 metros. De los 10 vehículos considerados en la Figura 5.16 el módulo logró detectar 9. Las placas detectadas se observan en la Figura 5.17.

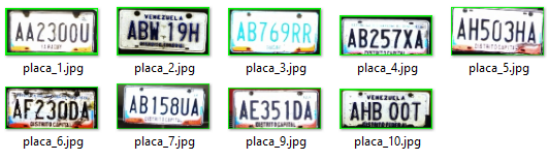

Figura 5.17: Resultado del *Módulo de detección de placa* procesando entradas que no participaron en la base de datos.

Tabla 5.4: Desempeño en la detección de placas.

Placas detectadas	Placas no detectadas	Número total de placas	Tasa de éxito
9	1	10	0.9

De esta forma, el sistema continúa hasta la detección de caracteres. Dado que las 9 placas ingresadas al siguiente módulo se detectaron todos los caracteres. Los resultados de detección se presentan a continuación:

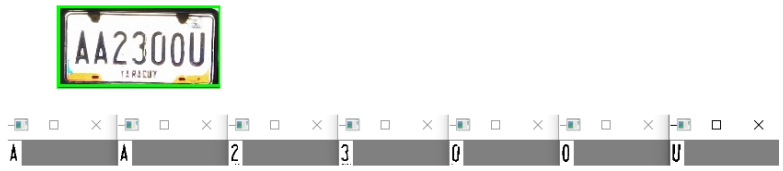

Figura 5.18: Extracción de caracteres de la placa 1

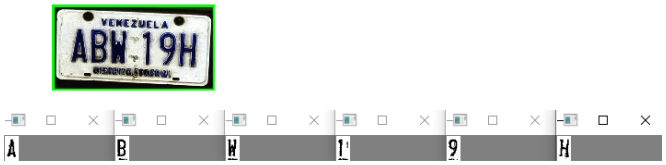

Figura 5.19: Extracción de caracteres de la placa 2

Figura 5.20: Extracción de caracteres de la placa 3

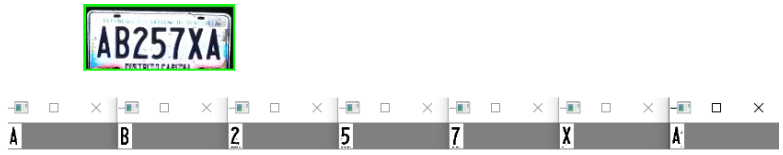

Figura 5.21: Extracción de caracteres de la placa 4

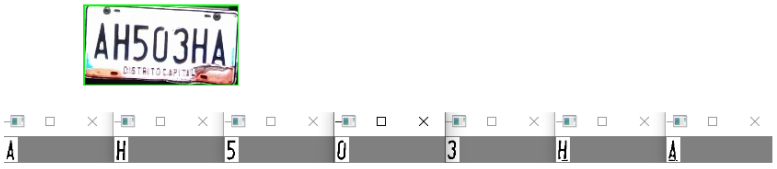

Figura 5.22: Extracción de caracteres de la placa 5

Figura 5.23: Extracción de caracteres de la placa 6

Figura 5.24: Extracción de caracteres de la placa 7

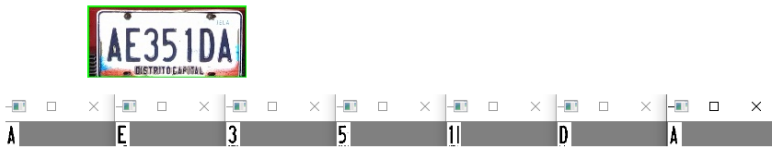

Figura 5.25: Extracción de caracteres de la placa 9

Figura 5.26: Extracción de caracteres de la placa 10

A continuación se presenta la Tabla 5.5 que reporta la placa, caracteres detectados y reconocidos a fin de evaluar su desempeño:

Tabla 5.5: Desempeño con nuevas entradas

Placa ID	Número de Placa	Detectados	Reconocidos
1	AA230OU	AA230OU	AA230OU
2	ABW19H	ABW19H	ABW19H
3	AB769RR	AB769RR	AB769RR
4	AB257XA	AB257XA	AB257XA
5	AH503HA	AH503HA	AH503HA
6	**AF230DA**	**AF230DA**	**XF23ODA**
7	AB158UA	AB158UA	AB158UA
9	AE351DA	AE351DA	AE351DA
10	**AHB00T**	**AHB00T**	**AH8007**

Se observa que la placa de la Figura 5.23 el algoritmo no reconoció correctamente el número 0 clasificándolo como una O. Este error ocurrió debido a factores externos que afectan las características físicas de la placa. Como se puede observar, esta placa se encuentra bastante deteriorada físicamente por lo que las manchas agregan bastante ruido para el algoritmo de reconocimiento. Las irregularidades físicas que se aprecian en la matrícula indicen significativamente en el procesamiento de imagen para la detección de contornos en el área de la placa. En consecuencia, la detección del contorno, como en el primer carácter, se ve influenciada por las manchas alrededor y desemboca en una extracción inexacta.

Tabla 5.6: Desempeño con nuevas entradas

Carácter	Tasa de éxito	Detectados	Reconocidos
0	0.8	5	4
1	1.0	3	3
2	1.0	3	3
3	1.0	4	4
4	0	0	0
5	1.0	4	4
6	1.0	1	1
7	1.0	2	2
8	1.0	1	1
9	1.0	2	2
A	0.933	15	14
B	0.8	5	4
C	0	0	0
D	1.0	2	2
E	1.0	1	1
F	1.0	1	1
G	0	0	0
H	1.0	4	4
I	0	0	0
J	0	0	0
K	0	0	0
L	0	0	0
M	0	0	0
N	0	0	0
O	1.0	1	1
P	0	0	0
Q	0	0	0
R	1.0	2	2
S	0	0	0
T	0.0	1	0
U	1.0	2	2
V	0	0	0
W	1.0	1	1
X	1.0	1	1
Y	0	0	0
Z	0	0	0

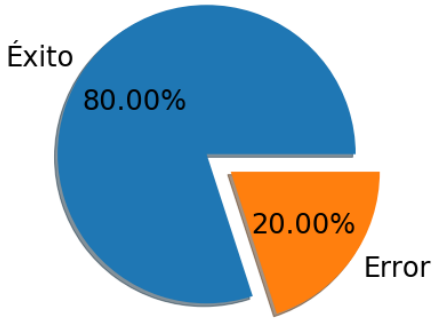

Figura 5.27: Tasa de éxito de reconocimiento de la letra 0.

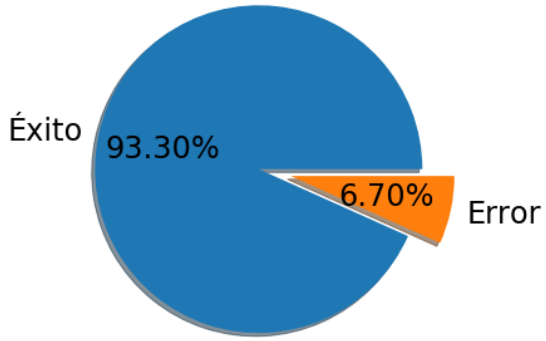

Figura 5.28: Tasa de éxito de reconocimiento de la letra A.

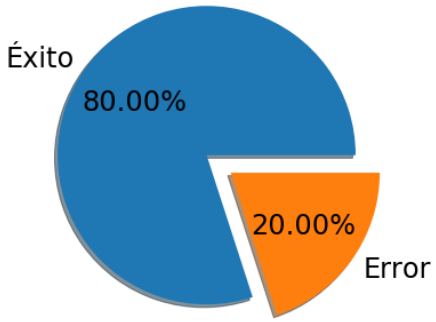

Figura 5.29: Tasa de éxito de reconocimiento de la letra B.

En la Tabla 5.6 se observan los resultados obtenidos de las 10 capturas. La

tasa de éxito obtenida en la detección de carácter fue de 100 % Por otro lado, la tasa de éxito en el reconocimiento de caracteres es de 93,40 % en reconocimiento de caracteres. En las Figuras 5.27, 5.28 y 5.29, se muestran gráficos circulares de la tasa de éxito y error del reconocimiento de los caracteres 0, A Y B.

5.1.3. Tiempos de ejecución

Otro aspecto crucial en la investigación fue el tiempo de entrenamiento de la SVM el cual fue de 1.29s, para la base de datos construída y las características descritas del computador.

En la Tabla 5.7 se observan los tiempos (Tiempo de CPU en segundos) de procesamiento de imágenes y reconocimiento de caracteres obtenidos en la prueba de *Reconocimiento de caracteres que no forman parte de la base de datos.*

Tabla 5.7: Tiempos de procesamiento y reconocimiento

Placa ID	Número de Placa	Procesamiento de imagen [s]	Reconocimiento de caracteres [s]
1	AA230OU	0.839	0.338
2	ABW19H	0.55	0.268
3	AB769RR	0.577	0.252
4	AB257XA	0.7	0.282
5	AH503HA	0.954	0.365
6	AF230DA	0.644	0.349
7	AB158UA	0.665	0.396
9	AE351DA	0.655	0.214
10	AHB00T	0.656	0.195

En la Figura 5.30, se muestra la gráfica de línea del tiempo de CPU del procesamiento de imagen y del reconocimiento de caracteres.

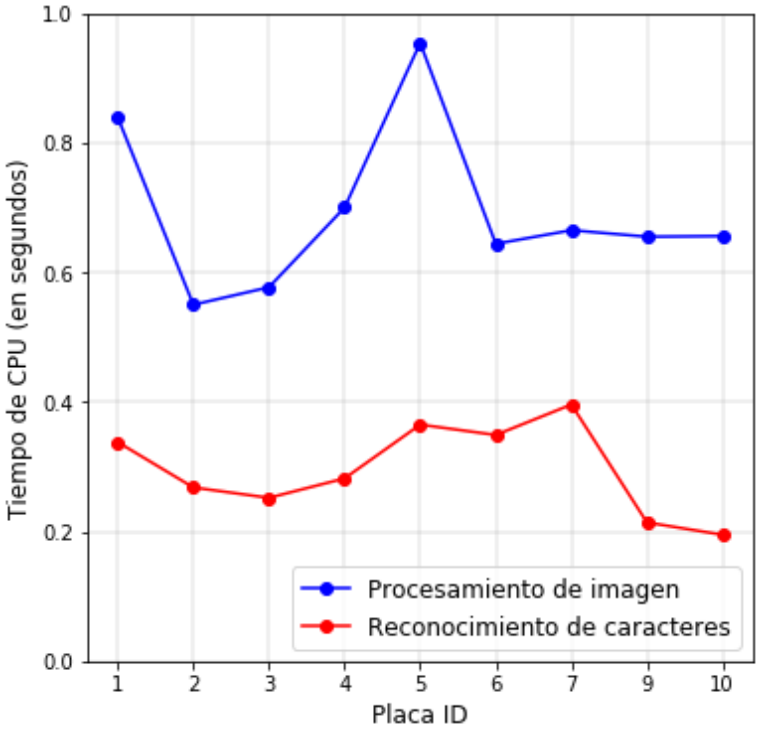

Figura 5.30: Gráfica tiempo del CPU del sistema para el procesamiento de imagen y reconocimiento de caracteres.

En promedio se obtiene que el tiempo de procesamiento de imagen es 0.624s y el tiempo de reconocimiento de caracteres 0.266s.

CAPÍTULO VI

CONCLUSIONES

El procesamiento de imágenes es una tecnología que permitirá la creación de sistemas capaces de automatizar tareas que darán paso a una nueva sociedad tecnológica, como lo son los automóviles no tripulados, reconocimiento de rostros, detección de movimiento, entre otros. Lo anterior se logra obteniendo información de interés dentro de una captura de imagen mediante herramientas y procedimientos matemáticos.

Desarrollar el sistema en módulos independientes permitió probar diferentes técnicas y herramientas sin afectar las entradas y salidas de cada uno con el fin de evaluar y elegir la solución más adecuada. A su vez, permitió realizar el entrenamiento del sistema de reconocimiento SVM sin depender del estado del módulo del procesamiento de imágenes. Por otro lado, una de las ventajas más importante fue la capacidad corregir los errores de un módulo sin perjudicar el funcionamiento del otro, pudiendo así desarrollarlos simultáneamente.

La detección conforma parte del procesamiento de imágenes y es el pilar fundamental que sostiene el sistema ANPR. Una detección exacta concluye en el correcto reconocimiento de la matrícula considerando las analíticas obtenidas en las diferentes pruebas. La detección es muy sensible a los factores externos como la exposición a la luz, la resolución y el ángulo de captura. Estos factores, que al ser controlables, pueden mejorar considerablemente el rendimiento de la detección y la efectividad del sistema completo. Existen otros factores que no se

puede controlar, por ejemplo el deterioro físico de la matrícula, que indicen de manera drástica en el rendimiento general; por lo que se hace necesario considerar otras metodologías.

En el procesamiento de imágenes se encuentran diferentes técnicas, que a través de procedimientos matemáticos, permiten extraer la información de interés en una captura de imagen como lo es el Filtro de Sobel. Empresas de tecnología como OpenCV crean herramientas que combinan diferentes técnicas de procesamiento de imágenes para proveer buenos resultados en menor tiempo. La umbralización adaptativa de OpenCV fue una herramienta fundamental en el desarrollo de los módulos de *Detección de placa* y *Detección de carácter*, puesto que las características obtenidas en el tratamiento de la imagen tenían mejor definición de los contornos de interés y menor detección en los contornos que, para efectos del proyecto, se interpretaban como ruido.

En el módulo de procesamiento de imagen se presenta la mayor dificultad, dado que los métodos de restauración y mejoramiento de la imagen para extraer la información de interés requieren de mayor investigación. A pesar de que las bibliotecas de código abierto proveen a los usuarios de extensa documentación, es necesario instruirse en los conceptos que envuelven la imagen digital y sus tratamientos para de esta forma aplicar las técnicas adecuadas según lo que se necesita extraer.

Uno de los principales retos de este trabajo fue contar con una base de datos estandarizada que cumpliera con los requerimientos, además, fue la etapa cuyo desarrollo tomó más tiempo. En esta etapa nuevamente el ángulo de captura de las placas representó un factor de gran importancia. Para distintas muestras, letras diferentes poseían características matemáticas similares debido al ángulo de captura lo cual resultó en el erróneo reconocimiento de la matrícula. Por ello, es necesario depurar la base de datos con un criterio definido para evitar

ambigüedades en el entrenamiento de la SVM.

Con una base de datos de apenas 1496 muestras y un tiempo de entrenamiento de 1.2 segundos, se obtuvo una tasa de éxito mayor al 90 % en el reconocimiento de caracteres. En promedio, el tiempo de reconocimiento del sistema fue de 0.26 segundos. Con base en los resultados obtenidos, la SVM demostró un alto rendimiento en el reconocimiento de matrículas. Esto demuestra que este algoritmo tiene la capacidad de destacar en el mundo del reconocimiento de texto con altas expectativas en su desempeño.

De las pruebas realizadas se destaca lo consiguiente:

- La detección de contornos es un proceso delicado que se ve influenciado por factores externos como el ángulo de captura y la exposición a la luz. Se puede disminuir su impacto estandarizando las condiciones de captura y ambientales en medida de lo posible.
- La detección de la placa es un proceso en el que algoritmo se enfrenta a mucho ruido. Se puede mejorar el rendimiento general del sistema si se emplean técnicas de detección con capacidades dinámicas.
- El modelo generado desde la base de datos se ajusta a los requerimientos del trabajo. La tasa de éxito del reconocimiento de caracteres fue el esperado y se puede mejorar, mejorando el módulo de procesamiento de imágenes.

CAPÍTULO VII

RECOMENDACIONES

Debido al aprendizaje adquirido en el desarrollo del trabajo y en el análisis de los resultados obtenidos, las recomendaciones con miras a mejorar el rendimiento del sistema propuesto son las siguientes:

- Emplear una técnica de procesamiento de imagen que brinde al sistema mayor capacidad para detectar la localización de la placa.
- Incluir en la base de datos muestras de placas de otros países que le permita al modelo tener mayor capacidad de reconocimiento.
- Depurar cualquier conjunto de muestras que se adicione a la base de datos antes de realizar el entrenamiento.
- Evaluar la integración de herramientas más sofisticadas en el área de procesamiento de imágenes para aumentar la efectividad del sistema en la localización de la placa y caracteres.
- Adaptar las condiciones y parámetros del sistema a la región en la cual se está empleando.

Apéndice I

Pseudocódigo

Programa 1. Pseudocódigo

```
Sistema de Reconocimiento Autómático de matrícula vehícular:

    Procesamiento de imagen:

        detección_placa (entrada == imagen ; salida == Placa vehícular)
            Redimensionamiento de la imagen
            Convertir en escala de grises
            Umbralización adaptativa
            detección de contorno
            for c en contornos
                if Valor máximo de área de la placa < c < Valor mínimo
                        promedio de área de la placa:
                    Recortar area de contorno de c
                    Placa Vehícular == c
            if placa vehicular == None
                'Por favor tome la captura nuevamente con ' \
                'las especificaciones de posición requeridas'
                end

        deteccion_caracteres: (entrada == Placa_vehícula;
             salida == caracteres)
            Umbralización adaptativa
            detección de contorno
            for c en contornos
                if Valor máximo de área de caracter < c < Valor mínimo
                    promedio de área de caracter:
                    caracter == deteccion de contorno
                    Acumular caracteristicas de c en  array caracteres

                if 5 > caracteres >= 8:
                    'Por favor tome la captura nuevamente con ' \
                    'las especificaciones de posición requeridas'
                end

            Extraer cada contorno en el array caracteres
                mediante sus coordenadas
            Organizar el array caracteres considerando desde
             la posicion más a la izquierda hacia derecha de
             cada caracter y volver almacenar en caracteres

        Reconocimiento de caracteres : (entrada == caracteres ;
         salida == Numero de placa)
```

```
            Aplanar las matrices que describen cada caracter
            en el array caracteres
            for caracter en caractares
                Alimentar la Máquina de Soporte
                 Vectorial con el vector caracter
                Acumular el resultado obtenido
                 en el string Numero de placa

    Mostrar en consola el resultado obtenido
    de reconocimiento de texto
```

Apéndice II

Código del programa ANPR

Programa 2. Módulo principal: ANPPR

```
import detecting_plate as dp
import Extraction as et
import PPIF as pf
import joblib
import cv2
from time import time


def character2str(character, plate_len, index):
    """This function transform the ID character to string text"""

    int_character = character.astype(int)[0]

    if plate_len == 7:                  # NEW PLATES

        if int_character < 10:          # Numbers
            str_character = str(int_character)

        else:                           # Letters
            str_character = chr(int_character + 55)
        print('caracter', character, str_character)

    if plate_len == 6:                  # OLD PLATES

        if int_character < 10:          # Numbers
            str_character = str(int_character)

            if int_character == 0 and not (index == 4 or index == 5):
                str_character = 'O'

        else:                           # Letters
            if int_character == 24 and (index == 4 or index == 5):
                str_character = '0'
            else:
                str_character = chr(int_character + 55)
        print('caracter', character, str_character)

    return str_character


def call_image():
    """ This function call the image that will be used for recognition"""
```

```
    file = './fuente/matricula_144.jpg'
    src = cv2.imread(file)
    name_number = pf.calculting_name()
    return src, name_number


def run():

    pis_time = time()
    image, name_number = call_image()
    plate = dp.detecting_plate(image, name_number)
    try:

        characters = et.extraction(plate)
        pif_time = time()
        svm_recon = joblib.load('modelo_entrenado1.pkl')
        plate_str = ''
        index = 1
        plate_len = len(characters)

        rs_time = time()
        for segment in characters:

            data = segment.reshape(-1)
            character = svm_recon.predict([data])
            str_character = character2str(character, plate_len, index)
            plate_str += str_character
            rf_time = time()

            for i in range(0, len(characters)):
             #Showing the character extraction results
                graf, _ = pf.resizing(image, image, 150)
                cv2.imshow('Plate', graf)
                cv2.imshow('Character' + str(i), characters[i])
                cv2.moveWindow('Character' + str(i), 20 + i * 120, 250)
            index += 1

        pif_time = round(pif_time - pis_time, 3)
        r_time = round(rf_time - rs_time, 3)
        print('Tiempo de procesamiento de imagen: '+ str(pif_time))
        print('Tiempo de reconocimiento de caracteres: ' +str(r_time))

        print('La placa es:', plate_str)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    except UnboundLocalError:
        print("Oops! May be your plate is physically deteriorated,"
              " Try the picture again with better light conditions...")

    except:
        print("Oops! I can't see all characters correctly. Try taking "
              "the picture again  and make sure about position
              and light conditions")

if __name__ == '__main__':
    run()
```

Apéndice III

Código del programa Detección de placa

Programa 3. Módulo detección de placa

```
import cv2
import PIF as pif


def detecting_plate(imagen, number_car):

    plate = []
    num_file = number_car

    plate_detected = False
    cv2.destroyAllWindows()

    src = imagen

    cut_src = pif.resizing_image(src)

    s_noise = pif.softing_noise(cut_src, 5)

    thresh_image, _ = pif.threshold_image(s_noise)

    contour = pif.finding_contours(thresh_image)

    for c in contour:  # FIRST CASE SEARCHING THE PLATE RECTANGLE

        plate_detected, plate =
         pif.searching_plate(contour, cut_src, plate_detected, num_file)
        break

    for kn in [3, 5, 7]:
     # SECOND CASE: THE IMAGE CONTOURS ARE EXPANDED TO FIND THE PLATE

        dilation = pif.dilating_image(thresh_image, kn, 1)

        contour_dilated = pif.finding_contours(dilation)

        for c in contour_dilated:

            x, y, w, h = cv2.boundingRect(c)

            plate_detected, plate =
            pif.searching_plate(contour_dilated, cut_src,
             plate_detected, num_file)
```

```
            break

        if plate_detected:
            break

    if not plate_detected:
     # THIRD CASE: REMOVE THE NOISE SMOOTHING AND
      DILATING TO SEARCH THE PLATE

        thresh_image, _ = pif.threshold_image(cut_src)
        contour = pif.finding_contours(thresh_image)

        for c in contour:
            x, y, w, h = cv2.boundingRect(c)

            plate_detected, plate = pif.searching_plate(contour, cut_src,
            plate_detected, num_file)
            break

    return plate
```

Programa 4. Librería pre procesamiento de imágenes para la detección de placa

```
import cv2
import numpy as np

""" This library is used for the plate detection and extraction"""


def resizing_image(image):
    """It Reading image's dimensions which going to be resizing and
    proper cutting  it. Whether you want change the size of the picture
    you have to change te width_new parameter. At the bottom of this
    function are a little code where you might check all values"""

    height, width = image.shape[0:2]

    aspect_ratio = (width / height)

    width_new = 1350

    height_new = int(round(width_new / aspect_ratio))

    standard_src = cv2.resize(image, (width_new, height_new))

    start_x = int(height_new * .45)
    final_x = int(height_new * .85)

    start_y = int(width_new * 0.20)
    final_y = int(width_new * 0.85)

    cut_src = standard_src[start_y:final_y, start_x:final_x]


    return cut_src


def softing_noise(image, kn):
    """ It Softing the noise in the original image. kn  is
    the dimension  of the Kernel.I recommend you to use kn=5 for
    majority of pictures """

    s_noise = cv2.GaussianBlur(image, (kn, kn), 0)

    return s_noise


def threshold_image(image):
    """Converting to gray scale the image to make an
     adaptative thresholding for find the outlines"""

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    thresh_image = cv2.adaptiveThreshold(gray, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 13)

    return thresh_image, gray
```

```
def dilating_image(image, kn, i):
    """Dilating image's contours. kn is the dimensions of kernel"""

    kernel_dlt = np.ones((kn, kn), np.uint8)

    dilation = cv2.dilate(image, kernel_dlt, i)

    return dilation


def finding_contours(image):
    """Searching in the image's contour"""

    contour, _ = cv2.findContours(image,
     cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    return contour


def searching_plate(contour, image_print, plate_detected, number_file):
    """This function compare all the contours had found
     width and height with average values in order to filter them"""

    global plate

    width_max_plate = 190
    width_min_plate = 160
    height_max_plate = 95
    height_min_plate = 70

    for c in contour:
        x, y, w, h = cv2.boundingRect(c)

        if (w <= width_max_plate) and (w >= width_min_plate)
         and (h <= height_max_plate) and (h >= height_min_plate):
          # FILTERING THE RECTANGLE'S HEIGHT

                image_plate = cv2.rectangle(image_print, (x, y),
                 (x + w, y + h), (0, 255, 0), 2)
                  #DRAWING THE PLATE'S RECTANGLE
                plate_detected = True

                plate = image_print[y:y+h, x:x+w]

                # cv2.imshow('Plate ' + number_file, plate)
                # cv2.moveWindow('Plate ' + number_file, 30, 20)

                # cv2.imshow('Plate detected number
' + number_file, image_plate)
                # cv2.moveWindow('Plate detected number
' + number_file, 950, 20)

        if not plate_detected:
            plate = None

    return plate_detected, plate
```

Apéndice IV

Código del programa Segmentación de caracteres

Programa 5. Módulo extracción de caracteres

```
import PPIF as pf
import cv2


def extraction(source):

    for n_char in [7, 6]:
        for kn_blr in [11, 15, 9, 1]:

            no_noise = pf.softing_noise(source, kn_blr)
            resize, image_tocut = pf.resizing(no_noise, source, 150)
            to_detect, to_cut = pf.threshold_image(resize)
            contour, _ = cv2.findContours(to_detect,
            cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            detecting, character = pf.detecting_characters(contour,
             resize, _)

            if len(character) == n_char:
                break
        if not len(character) == n_char:
            continue

        break

    character = pf.org_character(character)
    to_cut = pf.preparing_tocut(image_tocut)
    segmented = pf.cutting_characters(character, to_cut)

    return segmented

if __name__ == '__main__':
    extraction()
```

Programa 6. Librería pre procesamiento de imágenes para la segmentación de caracteres

```
import cv2
import numpy as np
import os
import glob

""" This library is used for the character detection and extraction"""

def calculting_name():
    """ This function search ID number of the image
     that contains a plate number"""

    list_of_files = glob.glob('./muestras/*')
     # * means all if need specific format then *.csv
    latest_file = max(list_of_files, key=os.path.getctime)
    _, name_file = os.path.split(latest_file)
    name, _ = os.path.splitext(name_file)
    name_number = str(name)

    return name_number


def resizing(image, image_2, desire_width):
    """This function Resize the image in width and
    height using original aspect ratio.
    It use desire width for calculate the new height

    Besides, resizing returns the same image twice because
    the extraction module needs it for later image
    processing purposes."""

    height, width = image.shape[0:2]

    aspect_ratio = (width / height)

    new_width = desire_width

    new_height = int(round(new_width / aspect_ratio))

    standard_src = cv2.resize(image, (new_width, new_height))

    image_tocut = cv2.resize(image_2,(new_width, new_height))

    return standard_src, image_tocut

def softing_noise(image, kn):
    """ It Softing the noise in the original image. kn
is the dimension  of the Kernel.I recommend you to use kn=5 for
    majority of pictures """

    s_noise = cv2.GaussianBlur(image, (kn, kn), 0)

    return s_noise


def threshold_image(image):
```

```
    """Converting to gray scale the image to make
    an adaptative thresholding for find the outlines"""

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    thresh_image_inv = cv2.adaptiveThreshold(gray, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 15, 7)

    thresh_image = cv2.adaptiveThreshold(gray, 255,
     cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 15, 7)
    return thresh_image_inv, thresh_image


def dilating_image(image, kn, i):
    """Dilating image's contours. kn is the dimensions of kernel"""

    kernel_dlt = np.ones((kn, kn), np.uint8)

    dilation = cv2.dilate(image, kernel_dlt, i)

    return dilation




def estimation_area(image, width, height):
    """ This functions develops differents  area
    estimation that will be uses in the caracter detection

    There is particular characters that are out of common average:

    Common average: w 17 - h 35.
    One: w 7 - h 35.
    I: w 10 - h 35.

    This occurs by the way that Boundingrectangle works.

    """

    area = width * height
    height_w, width_w = image.shape[0:2]
    whole_area = height_w * width_w
    relation_area = area / whole_area

    area_character = relation_area
    bias = area_character * 0.50
    low_limit = area_character - bias
    high_limit = area_character + bias

    aspect_ratio = width / height
    aspect_bias = aspect_ratio * 0.25
    max_aspect = aspect_ratio + aspect_bias
    min_aspect = aspect_ratio - aspect_bias

    return low_limit, high_limit, max_aspect, min_aspect, whole_area


def detecting_characters(contour, image_print, number_file):
```

```
    """This function compare all the contours  values had found
with area_estimation values in order to filter them """
    character = []
    image = image_print.copy()
    widths = [17, 10]

    for w in widths:
        low_limit, high_limit, max_aspect, min_aspect, whole_area =
         estimation_area(image_print, w, 35)
        for c in contour:
            x, y, w, h = cv2.boundingRect(c)
            area_contour = w * h
            aspect_ratio = w / h

            if (area_contour/whole_area >= low_limit) and
            (area_contour/whole_area <= high_limit) and
            (aspect_ratio < max_aspect) and (aspect_ratio > min_aspect):
                cv2.rectangle(image, (x, y), (x + w, y + h),
                (0, 255, 0), 2)  # DRAWING THE PLATE'S RECTANGLE

                # print(w, h)
                # cv2.imshow('Drawing', image)
                # cv2.waitKey(0)

                rectangle_char = (x, y, w, h)
# x = UPPER - LEFT CORNER OF THESE RECTANGLE
                character.append(rectangle_char)
# FILLING THE CHARACTER VARIABLE.


    image_plate_char = image

    return image_plate_char, character


def filling_white(image, smaller_image):
    """ This functions set white the pixels according
     to percents that it has asigned """

    rec_char_outer = image.copy()

    fil_out, col_out = rec_char_outer.shape[0:2]

    left_limit = 4
    right_limit = col_out * 0.80
    up_limit = 5
    down_limit = fil_out * 0.95

    for n in range(0, fil_out):
        for m in range(0, col_out):
            if ((m < left_limit) or (m > right_limit))
             or ((n < up_limit) or (n > down_limit)):
                rec_char_outer[n, m] = 255

    return rec_char_outer


def key_ordenation(tupla):
```

```
    """ This key indicates that it will be sort by
     the fisrt tupla's element. This element is the upper left
      x coordinate of the rectangle."""

    return tupla[0]


def org_character(characters):
    """ This functions will sort the characters have
     found left to right using the key ordenation"""

    ord_characters = sorted(characters, key=key_ordenation)
    return ord_characters


def cutting_characters(character, image_2cut):
    """ In this functions develops the rectangles cut
     that contains every single character detected by using
    the coordinates given by  BoundingRectangle function"""

    preparing = []
    m = len(character)
    image_2cut = image_2cut.copy()

    for n in character:

        # The information is extracted from  the
         tupla n in character list.
        # For more information about this coordinates check
         the Bounding Rectangle function resources
        ulc_X = n[0]
        ulc_Y = n[1]

        width = n[2]
        height = n[3]

        #There is asigned new name to the above
         information and is constructed the rectangle.
        start_x = int(ulc_X)
        start_y = int(ulc_Y)

        width_new = int(width)
        height_new = int(height)


        final_x = start_x + width_new
        final_y = start_y + height_new

        # A width and height outter value is placed
         that allow a prudential margin of the principal content.
        width_outer = 25
        height_outer = 45


        #Then the rectangle is constructed with these
         outter width and heigt and the x and y coordinate
          are displaced too.
        x_outer = int(ulc_X) - 4
```

```
        y_outer = int(ulc_Y) - 6

        outer_xf = x_outer + width_outer
        outer_yf = y_outer + height_outer

        # Both rectangles are cutted by image_2cut

        rec_char_outer = image_2cut[y_outer:outer_yf, x_outer:outer_xf]

        rec_char_inter = image_2cut[start_y:final_y, start_x: final_x]

        # Imperfections are corrected and filling with
         white color by filling_white

        prep = filling_white(rec_char_outer, rec_char_inter)

        prep, _= resizing(prep, prep, 15)

        preparing.append(prep)

    return preparing


def preparing_tocut(image):
    """ This function  makes the treshhold in the entry
     image but use the non inverted treshold considering
      subsequent recognition processes"""

    _, image = threshold_image(image)

    return image
```

Apéndice V

Código del entrenamiento del modelo

Programa 7. Módulo entrenamiento del modelo utilizando SK-learn

```
import numpy as np
from sklearn import svm
import joblib
from time import time

def run():

    with open("./data_base1.csv") as file:
        n_cols = len(file.readline().split(";"))
        print(n_cols)

    X = np.loadtxt("./data_base1.csv", delimiter=";",
     usecols= np.arange(0, n_cols - 1))
    Y = np.loadtxt("./data_base1.csv", delimiter=";",
     usecols= n_cols-1)

    recon_char = svm.SVC(kernel='linear', decision_function_shape='ovr')
    print('Entrenando')
    start_time = time()
    recon_char.fit(X, Y)
    time_lapse = time() - start_time
    print(time_lapse)

    w = recon_char.coef_[0]
    b = recon_char.intercept_[0]

    num_suportv = recon_char.n_support_
    ind_suportv = recon_char.support_
    suport_vec = recon_char.support_vectors_


    joblib.dump(recon_char, 'modelo_entrenado1.pkl')
    modelo_cargado = joblib.load('modelo_entrenado1.pkl')

    print('Probando la prediccion sobre la base de datos',
    recon_char.score(X, Y))



if __name__ == '__main__':
    run()
```

REFERENCIAS

[1] Alegsa, L. Definición de gráfico rasterizado. Consultado en: https://marketing4ecommerce.net/que-es-una-imagen-vectorial-y-como-reconocerla/ http://www.alegsa.com.ar/Dic/grafico_rasterizado.php, 2010. Fecha de consulta: 20-01-2020.

[2] Alonso, L. Qué es una imagen vectorial. Consultado en: https://marketing4ecommerce.net/que-es-una-imagen-vectorial-y-como-reconocerla/, 2018. Fecha de consulta: 20-01-2020.

[3] Bracho, J. Diseño e implementación de un sistema de reconocimiento de matrícula vehiculares. Trabajo de Grado, Universidad Central de Venezuela, 2016.

[4] Brend, J. y Stefan, K. *Handbook of ComputerVision and Applicationsl*, 1ra ed. 1999.

[5] Cortes, C., and Vapnik, V. Suport vector networks. *Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.* (1995), 273–297.

[6] Cover, T. Geometrical and statical properties of systems of linear inequalities with applications in patters recognition. *IEEE Transactions of Electronic Computer 14* (1965), 326–334.

[7] Cristianini, N., and Shawer-Taylor, J. *An introduction to Support Vector machines and other Kernels-based learning methods*, 16th ed. Cambridge University Press, 2014, ch. 1, 3, 6.

[8] Dhiraj, Y., and Pramod, B. A review paper on automatic number plate recognition (anpr) system. *International Journal of Innovative Research in Advanced Engineering (IJIRAE) 1* (2014), 88–92.

[9] Doxygen. Accessing image properties, image shape. Consultado en: https://docs.opencv.org/master/d3/df2/tutorial_py_basic_ops.html, 2019. Fecha de consulta: 30-06-2020.

[10] Doxygen. Image file reading and writing, imread. Consultado en: https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56, 2019. Fecha de consulta: 30-06-2020.

[11] Doxygen. Smoothing images, gaussian blur. Consultado en: https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html, 2019. Fecha de consulta: 30-06-2020.

[12] Doxygen. Structural analysis and shape descriptors, boundingrect. Consultado en: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga103fcbda2f540f3ef1c042d6a9b35ac7, 2019. Fecha de consulta: 30-06-2020.

[13] Fernández, A. Desarrollo de un modelo computacional para un sistema de reconocimiento de matrículas a través de una imagen proveniente de una cámara de tráfico. Trabajo de Grado, Universidad Central de Venezuela, 2011.

[14] Gonzales, R. y Woods, R. *Digital image processing*, 3ra ed. 2007.

[15] Magro, R. Binarizacion de imágenes digitales y su algoritmia como herramienta aplicada a la ilustriacion entomologica. *Boletín de la Sociedad Entomológica Aragonesa (S.E.A.) 53* (2013), 445.

[16] Morales, R. Diseño de un sistema de reconocimiento de embarcaciones en medio marítimo mediante el procesamiento de imágenes. Trabajo de Grado, Universidad Central de Venezuela, 2018.

[17] OpenCV, T. Drawing functions, rectangle. Consultado en: https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html#rectangle, 2019. Fecha de consulta: 30-06-2020.

[18] OpenCV, T. Geometric image transformations, resize. Consultado en: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html?highlight=resize#resize, 2019. Fecha de consulta: 30-06-2020.

[19] OpenCV, T. Miscellaneous image transformations, adaptative treshold. Consultado en: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html, 2019. Fecha de consulta: 30-06-2020.

[20] OpenCV, T. Miscellaneous image transformations, cvtcolor. Consultado en: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor, 2019. Fecha de consulta: 30-06-2020.

[21] OpenCV, T. Morphological operations, eroding and dilating. Consultado en: https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html, 2019. Fecha de consulta: 30-06-2020.

[22] OpenCV, T. Structural analysis and shape descriptors, findcontour. Consultado en: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours, 2019. Fecha de consulta: 30-06-2020.

[23] Petrou, M. y Petrou, C. *Image Processing: The Fundamentalsl*, 2da ed. 2010, ch. 1.

[24] Sarkar, D., Bali, R., and Sharma, T. *Practical Machine Learning with Python. A Problem-Solver's Guide to Building Real-World Intelligent Systems.* Apress, New York, 2018.

[25] scikit-learn developers. Support vector machines. Consultado en: https://scikit-learn.org/stable/modules/svm.html#svm-classification, 2020. Fecha de consulta: 12-12-2020.

[26] U.P.E.L. *Manual de trabajos de grado, Especialización y Maestía y Tesis Doctorales.*, 3ra ed. FEDUPEL, 2006, ch. 2.

[27] Vapnik, V. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.