TRABAJO ESPECIAL DE GRADO

DISEÑO DE UN SISTEMA DE RECONOCIMIENTO DE MATRICULAS VEHICULARES A TRAVÉS DEL PROCESAMIENTO DE IMAGENES 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, julio de 2020

TRABAJO ESPECIAL DE GRADO

DISEÑO DE UN SISTEMA DE RECONOCIMIENTO DE MATRICULAS VEHICULARES A TRAVÉS DEL PROCESAMIENTO DE IMAGENES 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, julio de 2020

A quien desees dedicar este trabajo

RECONOCIMIENTOS Y AGRADECIMIENTOS

Autor del Trabajo de Grado

Título del Trabajo de Grado

Tutor Académico: nombre del profesor. Tesis. Caracas, Universidad Central de Venezuela. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Mención Electrónica. Año 2020, xvii, 144 pp.

Palabras Claves: Palabras clave.

Resumen.- Escribe acá tu resumen

ÍNDICE GENERAL

RECONOCIMIENTOS Y AGRADECIMIENTOS III

ÍNDICE GENERAL VIII

LISTA DE FIGURAS XI

LISTA DE TABLAS XIII

LISTA DE ACRÓNIMOS XIV

INTRODUCCIÓN 1

CONCEPTUALIZACIÓN DEL PROYECTO 3

1.1. PLANTEAMIENTO DEL PROBLEMA 3

1.2. OBJETIVOS 4

1.2.1. OBJETIVO GENERAL 4

1.2.2. OBJETIVOS ESPECÍFICOS 4

1.3. ANTECEDENTES 5

MARCO REFERENCIAL 7

2.1. Imagen digital 7

2.2. Espacios de color 8

2.3. Procesamiento de imágenes digitales 8

2.4. Mascaras derivativas discretas 9

2.5. Binarización 11

2.6. Lenguaje de programación Python 3 12

2.7. OpenCV 3.4 12

2.8. Scikit-Learn 14

2.9. Máquina de Soporte Vectorial 14

MARCO METODOLÓGICO **19**

3.1. Tipo de investigación 19

3.2. Fases metodológicas 19

3.2.1. Recopilación de información 19

3.2.2. Selección 20

3.2.3. Evaluación de aplicabilidad 20

3.2.4. Programación y diseño 21

3.2.5. Implementación y pruebas 21

DEFINICIÓN Y DESCRIPCIÓN DEL SOFTWARE **22**

4.1. Arquitectura de software 22

4.2. Componentes del sistema 23

4.3. Computador. 23

4.4. Base de datos 24

4.5. Software 26

4.5.1. Detección de la placa 27

4.5.2. Segmentación de los caracteres. 28

4.5.3. Clasificación de caracteres. 30

4.6. Módulo de entrenamiento 31

PRUEBAS Y RESULTADOS **33**

5.1. Desempeño del sistema 33

5.1.1. Reconocimiento de caracteres que forman parte de la base de datos . 34

5.1.2. Reconocimiento de caracteres que no forman parte de la base de datos 40

5.1.3. Tiempos de ejecución 45

CONCLUSIONES **46**

RECOMENDACIONES **47**

Código del programa **48**

TÍTULO DEL ANEXO 58

TÍTULO DEL ANEXO 59

REFERENCIAS 60

LISTA DE FIGURAS

2.1. Operación de convolución sobre un pixel 9

2.2. Operador Sobel 10

2.3. Umbralización adaptativa de librería OpenCV 10

2.4. Filtro de sobel vs Umbralización adaptativa 10

2.5. Función Canny vs Umbralización adaptativa 11

2.6. Imagen a color binarizada. 11

2.7. Un hiperplano que separa los datos en dos grupos. 15

2.8. Otros ejemplos de hiperplano que separan los datos en dos grupos. 16

2.9. Hiperplano óptimo: máximo margen de separación entre dos clases. 16

4.1. Módulos del sistema 22

4.2. Muestras de imágenes del set e entrenamientos creada. 25

4.3. Proceso de segmentación de caracteres. 27

4.4. Proceso de segmentación de los caracteres de una matrícula. 28

4.5. Diagrama de proceso de clasificación 31

5.1. Placas que participaron en la construcción de la base de datos. 34

5.2. Procesamiento de la placa 82 de la base de datos. 34

5.3. Procesamiento de la placa 161 de la base de datos. 34

5.4. Procesamiento de la placa 229 de la base de datos. 35

5.5. Procesamiento de la placa 84 de la base de datos 35

5.6. Procesamiento de la placa 179 de la base de datos 35

5.7. Procesamiento de la placa 243 de la base de datos 35

5.8. Placas que no participaron en la base de datos 40

5.9. Resultado del *Módulo de detección de placa* procesando entradas que no participaron en la base de datos. 40

5.10. Extracción de caracteres de la placa 1 41
5.11. Extracción de caracteres de la placa 2 41
5.12. Extracción de caracteres de la placa 3 41
5.13. Extracción de caracteres de la placa 4 41
5.14. Extracción de caracteres de la placa 5 42
5.15. Extracción de caracteres de la placa 6 42
5.16. Extracción de caracteres de la placa 7 42
5.17. Extracción de caracteres de la placa 9 42
5.18. Extracción de caracteres de la placa 10 42

LISTA DE TABLAS

4.1. Especificaciones del computador Toshiba Satellite C55 Series 23

4.2. Clasificación y etiquetas de cada carácter. 26

5.1. Lista de placas que participaron en la prueba. 38

5.2. Desempeño con muestras de la base de datos. 39

5.3. Desempeño en la detección de placas. 41

5.4. Desempeño con nuevas entradas 43

5.5. Desempeño con nuevas entradas 44

5.6. Tiempos de procesamiento y reconocimiento 45

LISTA DE ACRÓNIMOS

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

SVM: Suport Vector Machine, Maquina de Soporte Vectorial.

INTRODUCCIÓN

Los primeros indicios del uso de la energía eléctrica sucedieron en el cuarto final del siglo XIX. La sustitución del gas y aceite por la electricidad además de ser un proceso técnico fue un verdadero cambio social que implicó modificaciones extraordinarias en la vida cotidiana de las personas, cambios que comenzaron por la sustitución del alumbrado público y posteriormente por varias clases de procesos industriales como motores, metalurgia, refrigeración y de último llegaron a las comunicaciones con la radio y la telefonía.

El siguiente cambio de paradigma en el que se vio involucrado la electricidad tuvo lugar a lo largo del siglo XX y surge desde la necesidad de facilitar las tareas realizadas a diario en casa. En ello los investigadores de la época vieron una solución adaptando equipos con energía eléctrica para su uso en el hogar. Las industrias replicaron el crecimiento tecnológico que tuvieron en sus productos, lo que trajo como consecuencia el desarrollo los electrodomésticos. La primera producción de aparatos en masa como refrigeradores, lavadoras, televisores y radios sucedieron en esta época y tuvieron una alta receptividad por parte de los compradores. La invención del transistor solo aceleró el reemplazo de aparatos dada su capacidad de minimizar los equipos.

La integración de la electrónica a la industria fomentó la creación de sistemas automatizados de adquisición de datos, supervisión y control también llamados sistemas *SCADA* por sus siglas en inglés. Estos sistemas manejan áreas críticas de las industrias y son parte de los procesos fundamentales de muchas de ellas por lo que necesitan ser diseñados con robustez, fiabilidad y seguridad. La aparición del Internet y las comunicaciones modernas en estos sistemas permite a los usuarios, de manera inálambrica incluso, monitorear y actuar sobre el sistema a distancia, sin presencia física en la planta.

Además de poder monitorear y realizar acciones sobre los sistemas, los instrumentos de

medida de última tecnología se fabrican de modo que puedan ser compatibles con medios de comunicación inalámbricas lo que posibilita la transmisión de datos adquiridos sin necesidad de cable a la central del sistema *SCADA*. El presente trabajo de grado pretende realizar el diseño de un sistema de adquisición y transmisión de datos integrando microcontroladores ESP32 a medidores de energía eléctrica para formar una red mallada inalámbrica capaz de transmitir los datos recolectados a un punto central.

En este archivo debe escribir su introducción.

De acuerdo a Brea la transformada de Laplace debe estudiarse como una función definida en el campo de los números complejos [**?**].

Otro mode de referencial es [**?**]

El resto del reporte consta de: en el Capítulo **??** se describe...

En el trabajo se emplea el enfoque de [**?**]

De acuerdo a la ecuación

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 estos representan grandes costos para su eficiencia.

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 sistemas de reconocimiento de matrícula vehicular, 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" [1], 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 realizar 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" [2], 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 [3] 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 antes 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" [4], 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 satisfactoriamente, 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 de procesamiento de imagen y aprendizaje de máquina.

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 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 [5] .

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 imagenes rasterizadas está definida por el total de pixeles que posee (Resolución) y la cantidad de información por pixel (Profunidad de color, bits por pixel). [6]

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

Se define así al conjunto de técnicas y métodos desarrollados para manipular la información contenida en una imagen digital. Estos consisten en aplicar diferentes operadores a la imagen con los siguientes objetivos [7]:

- 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.

2.4. 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 "kernel"del filtro. Dependiendo de los valores que componen al kernel y su distribución se obtienen diferentes resultados de filtrado en la imagen. Las mascaras derivativas discretas no son más que Kernels cuyos elementos representan una aproximación de la derivada [8]. En la figura 2.1 se puede apreciar el proceso de convolución sobre un pixel.

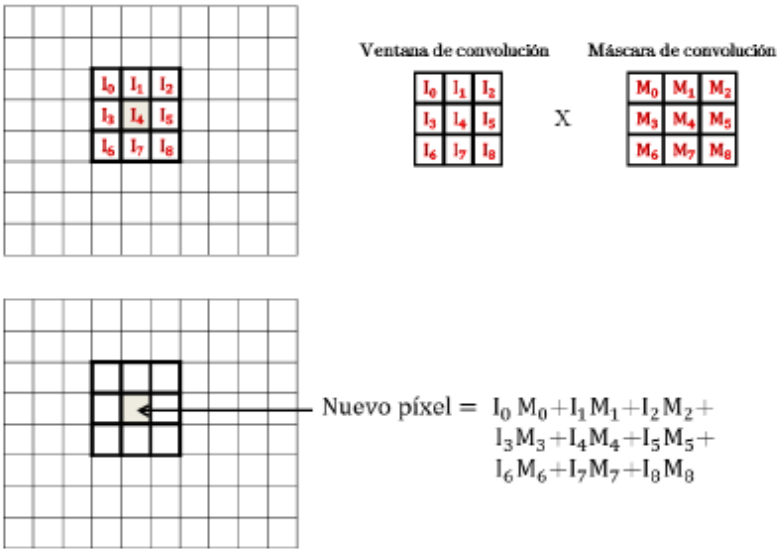

Figura 2.1. Operación de convolución sobre un pixel

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 [9]. 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:

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:

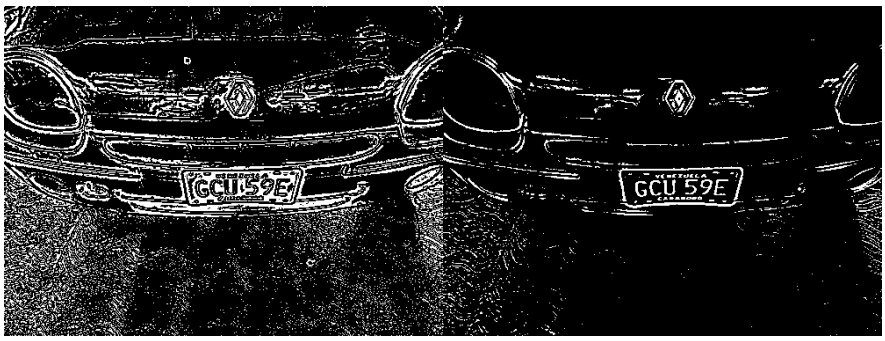

Figura 2.5. Función Canny vs Umbralización adaptativa

2.5. 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 [10]:

- 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

2.6. Lenguaje de programación Python 3

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 3 se caracteriza por su amplia aplicabilidad en diferentes campos del desarrollo de nuevas tecnologías como lo es el 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.

2.7. OpenCV 3.4

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/.

Entre las funciones que ofrece:

- `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. [11]
- `img.shape[]`: lee y devuelve las dimensiones de una imagen digital en función del ancho y alto. [12]
- `cv2.resize()`: redimensiona el ancho y alto de una imagen de acuerdo a los parámetros de entrada especificados. [13]
- `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. [14]
- `cv2.cvtColor()`: cambia el espacio de color de una imagen digital a otro. [15]
- `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. [16]
- `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 çrezcan". [17]

- `cv2.findcontours()`: definiendo contorno como una curva todos los puntos continuos que 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. [18]

- `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. [19]

- `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. [20]

2.8. Scikit-Learn

Scikit-Learn por su parte, es una biblioteca con basto desarrollo en los aprendizajes automático de máquina de software libre para el lenguaje de programación Python. Cuenta con varios algoritmos de regresión, clasificación y análisis de grupos como SVMs, bosques aleatorios, 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/.

2.9. Máquina de Soporte Vectorial

Se define la Máquina de Soporte Vectorial (SVM por sus siglas en inglés) como modelos de aprendizaje supervisado que implementa la idea de mapear el vector de entrada en un espacio de características Z de gran dimensión mediante un mapeo no lineal. En este espacio una superficie lineal es construída con propiedades que aseguran una alta capacidad

de generalización. Este tipo de algoritmo es usualmente utilizado en el reconocimiento de patrones, aunque también puede desempeñarse en regresión lineal.

Los problemas de clasificación en su forma más básica consiste en separar dos grupos o clases, estos son llamados problemas de clasificación binaria. Muchos problemas de clasificación pueden ser resueltos por las máquinas si los grupos son linealmente separables. Al ser linealmente separables es posible encontrar un hiperplano que divida las clases. De lo contrario, es necesario utilizar máquinas no lineales o llevar a cabo transformaciones que permitan separar los datos.

Construir el modelo o hipótesis del algoritmo SVM supone encontrar los parámetros del hiperplano que mejor separe $x_i \in R^m$ para $i = 1, 2, ..., n$ elementos en dos grupos o clases de datos. Existen infinitos hiperplanos que separan un conjunto de datos, sin embargo, esto no asegura una buena generalización.

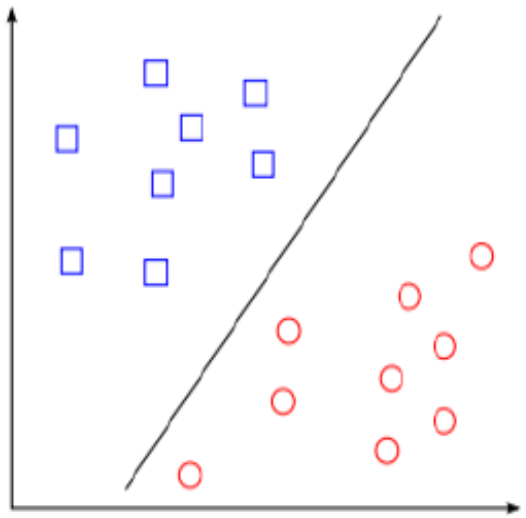

Figura 2.7. Un hiperplano que separa los datos en dos grupos.

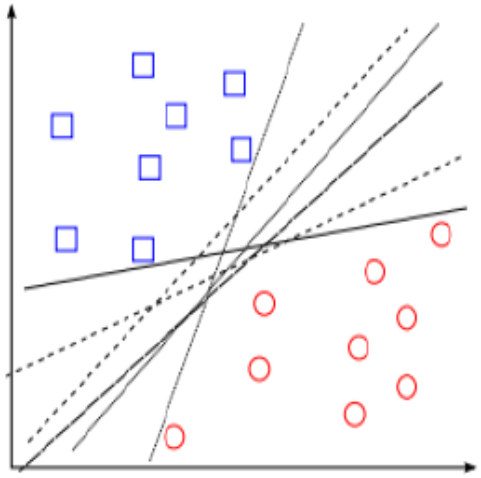

Figura 2.8. Otros ejemplos de hiperplano que separan los datos en dos grupos.

El problema anterior es resuelto por el hiperplano óptimo de clases separables. El hiperplano óptimo es definido por la función de decisión lineal con máximo margen entre los vectores de dos clases. La propiedad fundamental del híperplano de separación óptimo es que este equidista del ejemplo más cerca de cada clase [21]. Esta propiedad se refleja en la siguiente figura:

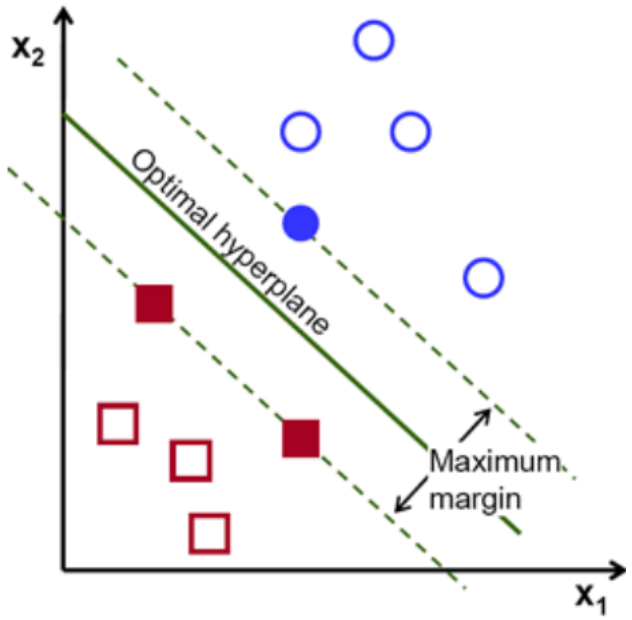

Figura 2.9. Hiperplano óptimo: máximo margen de separación entre dos clases.

Dado un set de entrenamiento clasificado:

$$(x_1, y_1), (x_2, y_2), ..., (x_m, y_m), \qquad y_m \ \in \ \{1, -1\}$$

Se dice que son linealmente separables si existe un vector w y un escalar w_0 que satisfaga la siguientes inecuaciones para todo el set de entrenamiento:

$$\begin{aligned} w^t \cdot x_i + w_0 1, \quad & Si \ \ y_i = 1 \\ w^t \cdot x_i + w_0 - 1, \quad & Si \ \ y_i = -1 \end{aligned} \tag{2.1}$$

Dónde w es el vector de pesos que ajusta el modelo, w_0 es un escalar también conocido como sesgo y x es el vector de datos de entrada a clasificar.

De esta forma, ajustar el modelo para el algoritmo se traduce en encontrar el hiperplano óptimo mediante el vector de pesos. El hiperplano óptimo se puede escribir entonces cómo:

$$w^t \cdot x_i + w_0 = 0 \tag{2.2}$$

Es el único que separa el conjunto de entrenamiento con el máximo margen, el cual determina la dirección $w/|w|$ dónde la distancia entre las proyecciones de los vectores de entrenamiento de dos clases es máxima. Esta distancia es definida cómo:

$$\gamma = \frac{1}{\|w^2\|} \tag{2.3}$$

Entonces el problema se traduce en máximar γ para encontrar dicho híperplano de separación óptimo tomando en cuenta las condiciones de las ecuaciones (2.1).

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.[22]

Algunas de las ventajas de las SVMs frente otros algoritmos son [23]:

- 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 ejemplos de entrenamiento en la función de decisión (Llamados vectores de soporte) por lo tanto es eficiente computacionalmente, en especial en el uso de la memoria.
- Es versátil: diferentes funciones de Kernel pueden ser especificadas para la función de decisión. Kernels comunes son proveídos pero es posible personalizarlos.

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."[24]

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.
    - Umbralización.
- Lenguaje de programación Python.
- OpenCV para procesamiento de imágen.
- Maquina 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 resulta-

dos 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, dónde 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 tres grandes módulos: detección de texto, procesamiento de imágenes y reconocimiento de caracteres.

Se seleccionó esta arquitectura ya que la solución propuesta para la problemática planteada se puede describir en tres diferentes etapas, cuyas funcionalidades son independientes entre sí, pudiéndose definir las interacciones entre las entradas y salidas. En la figura **??** 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. La detección de texto basado en el aprendizaje profundo de máquinas, el procesamiento de imágenes con OpenCV el filtrado de la información y por último la Máquina de Soporte Vectorial 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 set 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 Máquinas de Soporte Vectorial 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. 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 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 debió 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 set e entrenamientos creada.

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 Python3, la librería OpenCV de Intel.

Así, es posible las etapas del software en los siguientes 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 listado xxx en el Anexo) 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 o herramientas que permitan detectar la localización de la placa. Considerando que la placa contiene texto, es posible utilizar técnicas avanzadas como los detectores de texto, como por ejemplo máquinas de aprendizaje basadas en aprendizaje profundo. Estas herramientas creadas a partir de Machine Learning, permiten detectar el texto en una imagen según las características de su entrenamiento.

El modelo EAST es un detector de texto basado en el aprendizaje profundo y OpenCV. Dado que el objeto del presente proyecto es reconocer los caracteres de la placa, es posible emplear módulos de detección de texto ya desarrollados. Se empleó el modelo EAST para detectar la localización de la placa en la fotografía y, tomando en consideración el área de la placa, se extrae el contenido de interés.

4.5.2. Segmentación de los caracteres.

El módulo de segmentación (Ver listado xxx en el Anexo) se encarga de detectar y extraer los caracteres de interés dentro de la placa. 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:

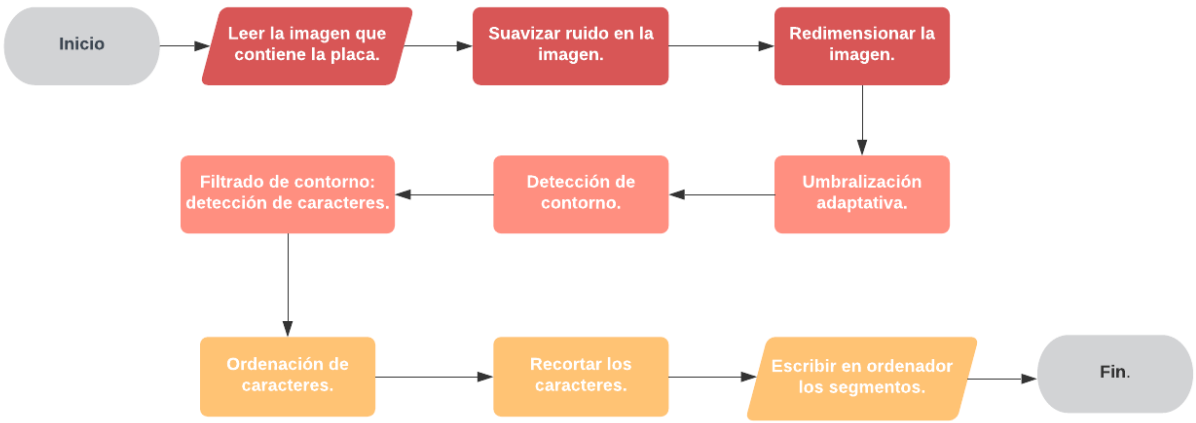

Figura 4.4. Proceso de segmentación de los caracteres de una matrícula.

1. **Suavizado de ruido:** En primer lugar, es necesario suavizar el ruido de la imagen que contiene la placa limpiándola de características no deseadas. Una característica no deseada puede aparecer debido a que la placa físicamente posee desperfectos que

intervienen significativamente en la detección de contornos o, por otro lado, si la fotografía es tomada fuera de las condiciones ideales. Para realizar esto se suavizó el 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:** 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 son colocados en blanco y el fondo en negro. En este proyecto se utilizó 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 kernel gaussiano. Es importante que la umbralización sea adaptable por región ya que factores externos como la iluminación pueden afectar significativamente el proceso de binarización.

4. **Detección de contorno:** La detección de contornos se aplica sobre la imagen umbralizada, dónde 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 de la imagen depende de los parámetros ingresados en la función. En este caso, se detectan los contornos cerrados en la imagen.

5. **Filtración de contornos:** La filtración de contorno consiste en seleccionar los contornos de interés. Se empleó lógica condicional a fin de verificar si el contorno clasifica 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. 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. De esta forma se estableció

el filtro de contornos dónde se detectan los caracteres de la matrícula. 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 extrajeron recortando 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.

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 las ecuaciones (2.1) 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 *Módulo de entrenamiento* (Ver listado xxx en el Anexo) del algoritmo.

El clasificador es un método para determinar la clase más probable de una entrada desconocida con respecto a un número de ejemplos de dichas 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. Es decir, 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 procesamientos de imágenes que ofrece OpenCV para convertir esa entrada en un vector plano que pudiera leer el algoritmo de clasificación. En el diagrama 4.5 se puede observar el proceso de clasificación:

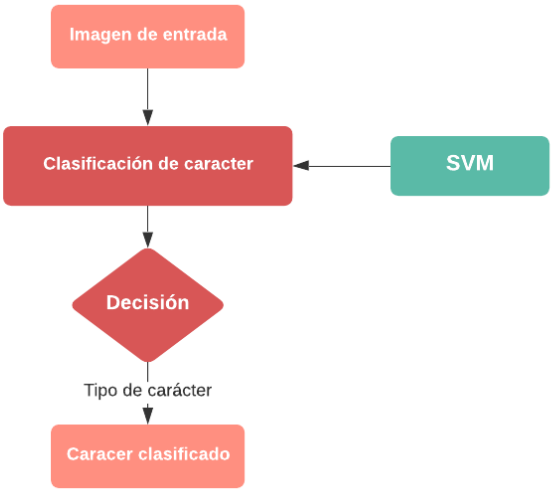

Figura 4.5. Diagrama de proceso de clasificación

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 los cuales se encuentran las letras del abecedario español (excluyendo la $\tilde{N}$) 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 Kernel Lineal

empleando las herramientas proporcionadas por la librería SKlearn SVM. El entrenamiento de una Máquina de Soporte Vectorial 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.

De esta forma, en el módulo de entrenamiento se encarga de encontrar el modelo que mejor se ajusta a la base de datos proporcionada (Ver listado xxx en el Anexo) .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 óptimos resultados en cuánto a detección y clasificación.

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 dichas pruebas la entrada al sistema fue con imágenes a diferentes resoluciones, ángulo de captura y distancia dado que no se dispone de un sistema físico real que obtenga entradas uniformes. 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 el correcto 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 la misma.

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.

Figura 5.1. Placas que participaron en la construcción de la base de datos.

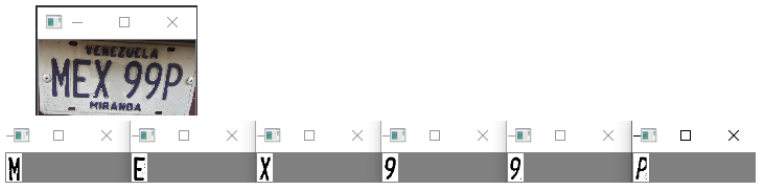

Figura 5.2. Procesamiento de la placa 82 de la base de datos.

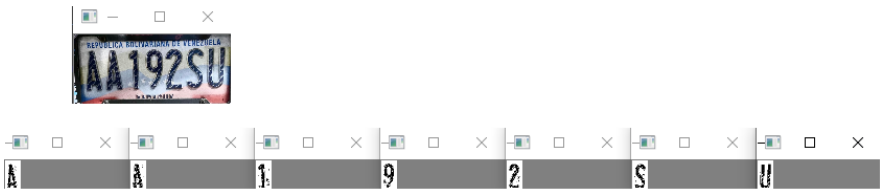

Figura 5.3. Procesamiento de la placa 161 de la base de datos.

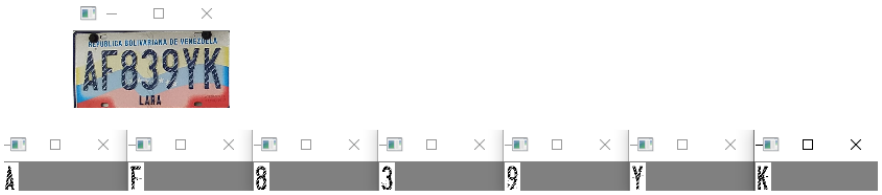

Figura 5.4. Procesamiento de la placa 229 de la base de datos.

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 son algunos de los casos dónde el Módulo de detección de caracteres detectó erróneamente y por lo tanto se considera un fallo. El resultado de estas muestras son clasificado cómo errores debido a que fallaron en detectar algunas de las letras de la placa.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 es 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 Kernel en el proceso de umbralización. A pesar que la Umbralización es adaptable por zonas, el tamaño del Kernel con el que recorre la imágen no lo es. En este proyecto el tamaño del kernel 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 matemáticamente los valores introducidos al modelo por medio de la extracción de características coincidan 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.

A continuación se presenta la Tabla 5.1 dónde se registró la placa, caracteres detectados y reconocidos a fin de evaluar su desempeño:

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	MEX15	MEX15L
251	AH850DM	AH850DM	AH850DM
126	AE192BA	AE192BA	AE192BA
179	AB218KN	AB218K	AB218A
229	AF839YK	AF839YK	AF839YK
234	AA453UD	AA45UD	AA45UD

El sistema falló en la detección de caracteres de 7 placas de 50. Este fallo se origina debido a que el Módulo de detección de caracteres no logró encontrar al menos 6 caracteres dentro de la matricula.

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

En la Tabla 5.2 se puede observar las analíticas obtenidas de un número total de 50 muestras que participaron en el entrenamiento de la base de datos. En este experimento se

obtuvo una tasa de éxito de 0.844 en la detección de caracteres. Mientras que en el rendimiento de reconocimiento de caracteres obtuvo una tasa de éxito 0.989.

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. Las condiciones 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.

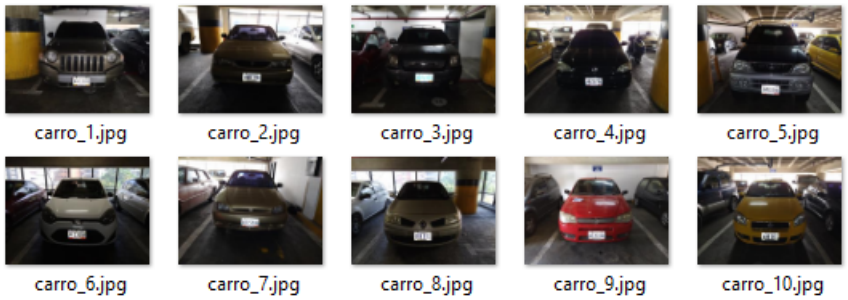

Figura 5.8. Placas que no participaron en la base de datos

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 5.8 el módulo logró detectar 9. Las placas detectadas se observan en la Figura 5.9.

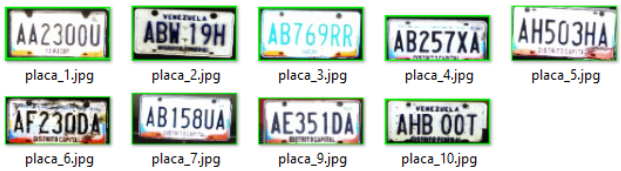

Figura 5.9. Resultado del *Módulo de detección de placa* procesando entradas que no participaron en la base de datos.

Tabla 5.3. 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. De las 9 placas ingresadas al siguiente módulo se detectaron todos los caracteres. Los resultados de detección se presentan a continuación:

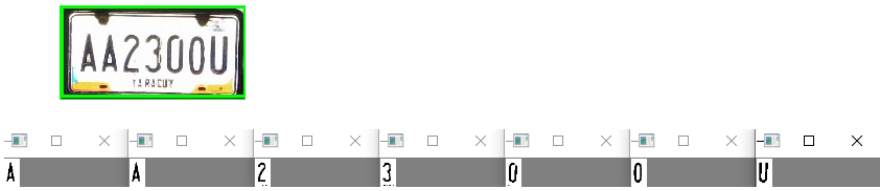

Figura 5.10. Extracción de caracteres de la placa 1

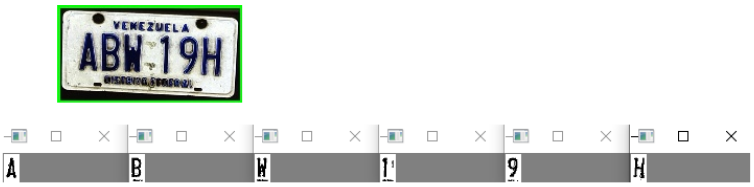

Figura 5.11. Extracción de caracteres de la placa 2

Figura 5.12. Extracción de caracteres de la placa 3

Figura 5.13. Extracción de caracteres de la placa 4

Figura 5.14. Extracción de caracteres de la placa 5

Figura 5.15. Extracción de caracteres de la placa 6

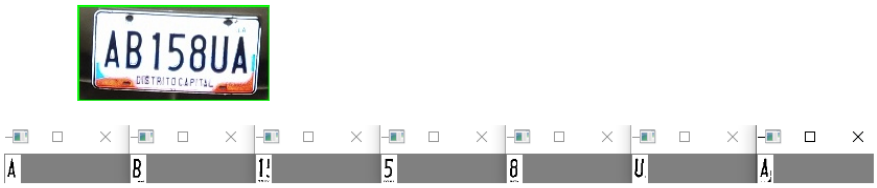

Figura 5.16. Extracción de caracteres de la placa 7

Figura 5.17. Extracción de caracteres de la placa 9

Figura 5.18. Extracción de caracteres de la placa 10

A continuación se presenta la Tabla 5.4 dónde se registró la placa, caracteres detectados y reconocidos a fin de evaluar su desempeño:

Tabla 5.4. 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 5.15 el algoritmo no reconoció correctamente el número 0 clasificándolo como una O. Este error ocurrió debido a los factores externos que afectan las características físicas de la placa. Como se puede notar esta placa se encuentra bastante deteriorada por lo que las manchas agregan bastante ruido para el algoritmo de reconocimiento. Las irregularidades físicas que se aprecian matricula 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 influenciado por las manchas alrededor y provoca una incorrecta extracción.

Tabla 5.5. Desempeño con nuevas entradas

Carácter	Tasa de exito	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

En la Tabla 5.5 se puede observar las analíticas obtenidas de las 10 capturas. La tasa de éxito obtenida en la detección de carácter fue de 1.0. Por otro lado, la tasa de éxito en el reconocimiento de caracteres es de 0.934 en reconocimiento de caracteres.

5.1.3. Tiempos de ejecución

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

Por otro lado, en la Tabla 5.6se puede observar los tiempos de procesamiento de imagenes y reconocimiento de caracteres obtenidos en la prueba de *Reconocimiento de caracteres que no forman parte de la base de datos.*

Tabla 5.6. 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 promedio se obtiene que el tiempo de procesamiento de imagen es 0.624s y tiempo de reconocimiento de caracteres 0.266s.

CAPÍTULO VI

CONCLUSIONES

CAPÍTULO VII

RECOMENDACIONES

Apéndice I

Código del programa

Programa 1. 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 = './muestras/carro_10.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)
    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(plate, plate, 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()


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

Programa 2. 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
    break

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

    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_detect
    break

    if plate_detected:
    break

    if not plate_detected:                          # THIRD CASE: REMOVE THE NOISE SMOOTHIN

    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
    print('Tercer caso')
    break

    return plate
```

Programa 3. 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 functi
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]

"""print('Height =' + str(height), 'Width =' + str(width), 'Height new =' + str(height
'Width new =' + str(width_new),'Start_x =' + str(start_x), 'Final_x' + str(final_x),
'Start_y =' + str(start_y), 'Final_y' + str(final_y))"""

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

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

thresh_image = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.TH

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 valu

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 (
# 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
```

Programa 4. 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_SIMPL
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 5. Librería pre procesamiento de imágenes para la detección 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 *
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

Besides, resizing returns the same image twice because the extraction module needs it

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

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

thresh_image_inv = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv
```

```
thresh_image = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.TH
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 carac

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_prin
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)
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

return tupla[0]


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

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
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 the tupla n in character list.
# For more information about this coordinates check the Bounding Rectangle function re
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 pri
width_outer = 25
height_outer = 45


#Then the rectangle is constructed with these outter width and heigt and the x and y c
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 tre
```

```
_, image = threshold_image(image)

return image
```

Apéndice II

TÍTULO DEL ANEXO

Apéndice III

TÍTULO DEL ANEXO

REFERENCIAS

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

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

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

[4] A. Fernández, "Desarrollo de un modelo computacional para un sistema de reconocimiento de matrículas a travÃľs de una imagen proveniente de una cÃąmara de tráfico," Trabajo de Grado, Universidad Central de Venezuela, 2011.

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

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

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

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

[9] R. Gonzales, R. y Woods, *Digital image processing*, 3rd ed., 2007.

[10] R. Magro, "Binarizacion de imagenes digitales y su algoritmia como herramienta aplicada a la ilustriacion entomologica," *BoletÃŋn de la Sociedad EntomolÃşgica Aragonesa (S.E.A.)*, vol. 53, p. 445, 2013.

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

[12] ——. (2019) Accessing image properties, image shape. Consultado en: https://docs.opencv.org/master/d3/df2/tutorial_py_basic_ops.html. Fecha de consulta: 30-06-2020.

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

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

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

[16] ——. (2019) Miscellaneous image transformations, adaptative treshold. Consultado en: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html. Fecha de consulta: 30-06-2020.

[17] ——. (2019) Morphological operations, eroding and dilating. Consultado en: https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html. Fecha de consulta: 30-06-2020.

[18] ——. (2019) 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. Fecha de consulta: 30-06-2020.

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

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

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

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

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

[24] U.P.E.L., *Manual de trabajos de grado, EspecializaciÃşn y MaestrÃŋa y Tesis Doctorales.*, 3rd ed. FEDUPEL, 2006, ch. 2.