2022-09

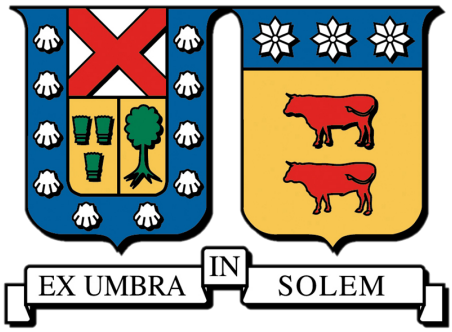
Reconocimiento de patentes en videos de baja calidad

Acevedo La Rivera, Sebastián Ignacio

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE



"RECONOCIMIENTO DE PATENTES EN VIDEOS DE BAJA CALIDAD"

SEBASTIÁN IGNACIO ACEVEDO LA RIVERA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN ELECTRÓNICA

PROFESOR GUÍA: **Agustín González**

PROFESOR CORREFERENTE : **Rudy Malonnek**

SEPTIEMBRE - 2022

Agradecimiento

Agradezco a mi familia que estuvo presente en todo el camino universitario, en especial a mis padres que me forjaron con los principios de esfuerzo y dedicación para culminar este largo viaje en el desarrollo profesional.

Agradezco a mis compañeros de universidad los cuales complementaron mis faltas en el conocimiento con sus puntos de vista, a mis hermanos que son mis modelos de vida y a mis amigos de la vida que en ellos me apoye en los peores momentos.

Por último y no menos importante, agradezco a la universidad por asegurar un ambiente donde las ideas pueden prosperar y a su gran calidad de profesores que me enseñaron todo lo necesario para llegar a este punto en mi vida profesional.

Resumen

Con el paso del tiempo se hace más necesario monitorizar el movimiento vehicular debido a la constante alza en parque vehicular, tanto como para las entidades publicas como las privadas, con este objetivo este proyecto propone un sistema automático de reconocimiento de patentes de Chile utilizando algoritmos modernos como 'You Only Look Once' (YOLO) para la detección de objetos y 'EasyOCR' para el reconocimiento de los caracteres. Se entrenó modelos específicos para el contexto chileno, con la intención de lograr rendimientos similares a los de un sistema embebido con cámara especializada, logrando un sistema que puede operar en tiempo real con el uso de una GPU y que es capaz de detectar más de un 90% de las patentes mayores a 15 píxeles de alto, mientras que el reconocimiento logra hasta un 54% de precisión dependiendo de la calidad de la imagen.

Palabras clave: Reconocimiento óptico de caracteres (OCR), 'Automatic Number Plate Recognition' (ANPR), YOLO, detección de objetos, Python

Abstract

With the passage of time, it becomes more necessary to monitor vehicular movement due to the constant increase in the vehicle fleet, both for public and private entities. With this objective, this project proposes an automatic system for the recognition of Chilean patents using modern algorithms. such as 'You Only Look Once' (YOLO) for object detection and 'EasyOCR' for character recognition. Specific models were trained for the Chilean context, with the intention of achieving performance similar to that of an embedded system with a specialized camera, achieving a system that can operate in real time with the use of a GPU and that is capable of detecting more than one 90% of patents are greater than 15 pixels high, while recognition achieves up to 54% accuracy depending on image quality.

Keywords: OCR, ANPR, YOLO, object detection,Python

Glosario

- **ROI**: Región de interés (Region of interest).
- **OCR**: Reconocimiento óptico de caracteres (Optical character recognition).
- **FPS**: Cuadros por segundos (Frame per seconds).
- **Frame**: Cuadro de imagen
- **GPU**: Unidad de procesamiento gráfico (Graphic processing unit).
- **Bbox**: Cuadro delimitador (Bounding box).
- **Accuracy**: Métrica para evaluar desempeño de modelos de clasificación y detección, corresponde a la división entre las predicciones correctas sobre total de predicciones.
- **Recall**: Métrica para evaluar el ratio de predicciones positivas acertadas sobre total.
- **PID**: Número identificador único de patente.
- **F1-score**: Métrica que combina accuracy y recall en un solo índice.

Índice

Índice de figuras

Índice de tablas

1 Introducción

En los últimos años en Chile se ha disparado el aumento del parque vehicular, desde el 2010 ha aumentado un 81% y solo en el primer semestre del año 2022 ha aumentado un 24,4% más en comparación con el primer semestre del 2021 [3], por esto la urgencia de desarrollar sistemas de monitoreo general de vehículos.

En la actualidad el Estado de Chile utiliza cámaras especializadas en pórticos para monitorear el movimiento vehicular y en caso de las autopistas extraer la patente de autos que no pagan peaje, análogamente la industria privada también prefiere este tipo de soluciones para monitorear estacionamientos y flujo vehicular.

En orden de aumentar la oferta de sistemas de reconocimiento de patentes, este proyecto propone un sistema especializado en patentes de Chile que no funcione en base a una cámara especializada y de alto costo, para poder ser utilizada en una variedad de contextos como podría ser un estacionamiento automático, seguridad de una empresa o edificio, etc.

Los objetivos de este proyecto son lograr un sistema automático de reconocimiento de patentes en lo posible capaz de trabajar en tiempo real, que posea un interfaz gráfica para la facilidad de uso para el usuario y que logre reconocer todas las patentes presentes en un vídeo, por ello debe lograr detectar, seguir y extraer los caracteres de la patente. Además se tiene como objetivo generar un conjunto de pruebas para verificar y evaluar el rendimiento del sistema desarrollado.

Los requisitos funcionales son:

- Detectar todas las patentes presentes en el vídeo.
- Extraer en lo posible, todas las patentes presentes en el vídeo.
- Disponer de una interfaz de usuario que permita ingresar el vídeo, mostrar los resultados y poder guardarlos en un archivo .csv.

2 Tecnologías de Reconocimiento Automático de Patentes

2.1 Explicación de modelo de sistema

En la actualidad existen diversos sistemas que permiten detectar y extraer los caracteres de las patentes presentes en una imagen, si bien presentan diferencias en la implementación, la mayoría comparte una misma estructura que se puede resumir la figura 2.1:

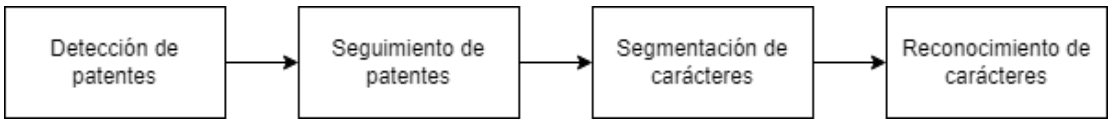


Figura 2.1: Estructura de sistema

Esta estructura de sistema se alimenta de una imagen o un vídeo donde el módulo de detección analiza la imagen y extrae la posición de las patentes presentes, estas posiciones son entregadas al módulo de seguimiento, actualizando la posición frame por frame y extrayendo el segmento de imagen donde está la patente para entregar solo la región de interés (ROI) al módulo de segmentación, la cual es analizada con el objetivo de separar los caracteres presentes, enviando una lista hacia el módulo final de reconocimiento para traducir la imagen de los caracteres a un texto.

El autor de [7] presentó un resumen de las alternativas de *Automatic Plate Number Recognizer* (ANPR) presentes hasta el año 2013, donde se presenta una estructura similar a la expuesta en la figura 2.1 con la diferencia en la ausencia del módulo seguimiento, ya que la investigación está orientada a imágenes y no vídeos. Además en dicha publicación se presentó algunos problemas recurrentes en este tipo de aplicaciones, en cuanto a la detección tanto como el tamaño y color de la patente pueden generar problemas, mientras que para el reconocimiento la baja calidad de imagen y las diferentes posiciones e intensidades de la fuente lumínica pueden generar problemas.

2.2 Sistemas embebidos

Dentro del espectro de soluciones para el problema de extraer la patente de los autos existe el nicho de los sistemas embebidos [2], que consisten en una cámara, generalmente de alta calidad; un emisor de luz, en el espectro no visible; un grabador de vídeo, el cual puede ser local o grabar directo a la nube; múltiples sensores, como detectores de movimiento para desencadenar la captura de la patente; y una interfaz gráfica de usuario, para visualizar los resultados.

En este nicho existen dos grandes competidores, 'Hikvision' y 'Dahua' donde ambos disponen de diversos modelos de cámara que varían su calidad y modo de grabado de vídeo, lo que hace llamativo este tipo de soluciones es enfoque especializado en detectar y reconocer patentes, mejorando considerablemente la precisión con que estos sistemas extraen las patentes, evitando problemas tales como déficit de luz al emitir algún espectro de luz no visible aprovechando la propiedad reflectante.

El principal problema de este tipo de tecnologías es el costo al momento de escalar el sistema, ya que se necesita adquirir una cámara por punto de vigilancia, las cuales son considerablemente más caras que una cámara de uso común.

2.3 Algoritmos de detección

Los algoritmos de detección de objetos se puede dividir en dos familias, los algoritmos basados en *'deep learning'* y los algoritmos basados en características, debido al gran avance en el mundo de la inteligencia artificial y las redes neuronales, los algoritmos basados en *'deep learning'* son los más utilizados debido a su gran precisión y velocidad al detectar [1].

2.3.1 Algoritmos basados en características

Dentro de los algoritmos basados en características el más utilizado es el detector basado en 'Histograms of oriented gradients' (HOG). Este detector separa una ventana de tamaño fijo para analizar y calcula las gradientes a nivel de píxeles, generando una nueva imagen de gradientes para luego separarla cuadriculas de 8 x 8 píxeles, a cada una de estas celdas se le calcula el histograma de 9 bins separados por 20° de la orientación de la gradiente, luego estos histogramas de agrupan en grupos de cuatro celdas generando un vector de 36 valores, por último se normaliza el vector generado con la norma euclidiana y se le entrega como entrada a un clasificador del tipo *'Support Vector Machine'* (SVM) para obtener la posición y el tipo de objeto presente en la imagen. En el diagrama de la figura 2.2 se muestra el pipeline ejecutado para cada ventana.

Haciendo uso del enfoque de ventana móvil, se recorre la imagen para detectar todas las posibles posiciones del objeto, detectando para cada ventana si el objeto está presente en esta, generando para el mismo objeto múltiples lecturas por cada ventana en el que está presente, para evitar múltiples instancias de un mismo objeto, se aplica el algoritmo de *'Non-maximum Suppressions'* (NMS) [10] para filtrar y unificar las instancias calculadas.

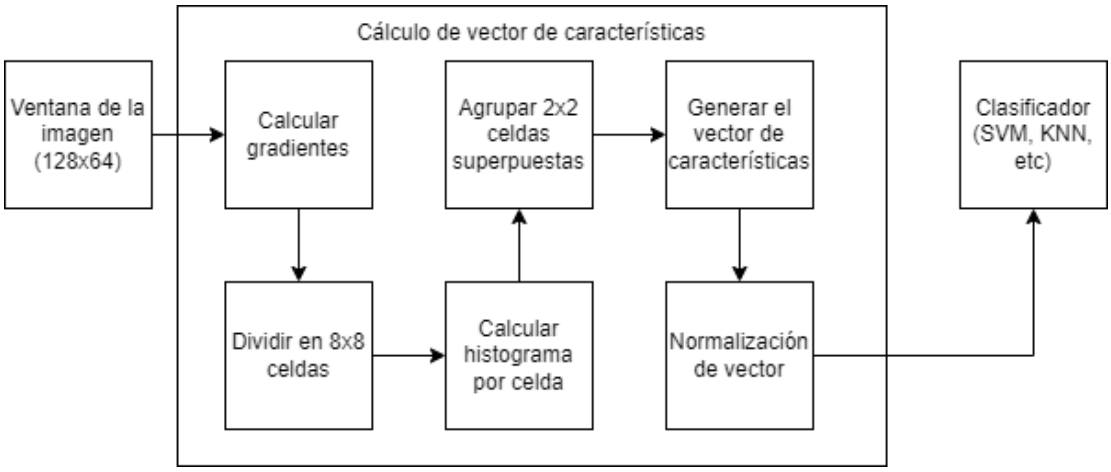


Figura 2.2: Diagrama de flujo de detector HOG.

Si bien este algoritmo puede detectar con una buena precisión, el problema de este algoritmo es su lentitud, debido a la necesidad de realizar cómputos píxel por píxel, quedando casi obsoleto versus la competencia con redes neuronales, pero su sigue siendo útil en contextos de detección de personas.

2.3.2 Algoritmos basados en deep learning

El aprendizaje profundo de máquinas ha revolucionado el rubro de la visión por computador y la detección de objetos, permitiendo crear nuevos algoritmos basados en las *'Convolutional Neural Network'* (CNN), las cuales permiten entrenar una red neuronal para que extraiga características de la entrada mediante parámetros ajustables que actúan como un filtro que recorre la imagen, calculando información de la entrada como bordes, textura, colores, etc.

Dentro de esta familia de algoritmos existen dos tipos, los algoritmos de una o dos etapas, la diferencia que existe entre estos es si predicen el tipo y la posición del objeto detectado en una etapa centralizada o en dos etapas que consisten en primero predecir todos los cuadros delimitadores con métodos convencionales de visión por computador como detección de bordes o filtros Haar, luego la clasificación de cada una de estos cuadros basados en vectores de características con una regresión de cuadros delimitadores.

Un ejemplo de algoritmo de dos etapas es *'Region-based Convolutional Neural Network'* (R-CNN) y para algoritmos de una etapa es *'You Only Look Once'* (YOLO) los cuales se explicaran a continuación.

Region-Based Convolutional Neural Network

Algoritmo de dos etapas que recibe como entrada la imagen completa que se le extraen una gran cantidad de candidatos de cuadro delimitador (cerca de 2000 candidatos) mediante algún método de proposición

de región como búsqueda selectiva, luego a cada una de estas regiones de interés se extrae un vector de características con una CNN previamente entrenada, estos vectores son aplicados como entrada en múltiples SVMs, cada una entrenada para detectar una clase de objeto, prediciendo la clase del objeto y finalmente mediante una regresión de cuadro delimitador, predecir la posición correcta del objeto entre todas las regiones propuestas. En la figura 2.3 se puede observar un diagrama de bloques que resume el algoritmo.

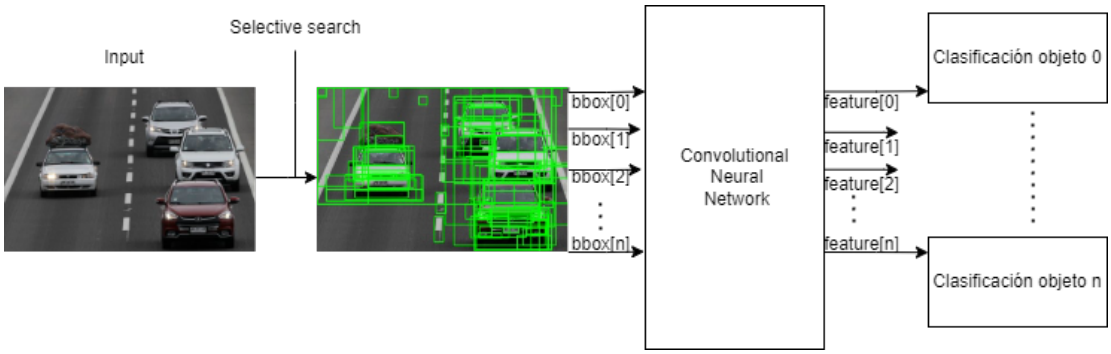


Figura 2.3: Diagrama de bloques R-CNN.

Si bien este algoritmo puede alcanzar una precisión ideal, su principal problema es la lentitud debido a la gran cantidad de regiones propuestas, obligando a ejecutar la extracción de características para cada una de estas, como también al momento de clasificar debido a que es necesario entrenar y clasificar con una SVM por cada clase de objetos, resultando en grandes tiempos de entrenamiento y de predicción.

Con el paso del tiempo este algoritmo ha tenido nuevas versiones que buscan agilizar el tiempo de entrenamiento y de predicción, nuevas versiones como Fast R-CNN se enfocan en optimizar el proceso de extracción de características, mientras que la versión Faster R-CNN se basa en Fast R-CNN pero mejorando el módulo de proposición de regiones, utilizando CNN para proponer estas. En el paper [9] que presenta esta versión, se reporta que dicho sistema puede detectar objetos a 5 FPS en una GPU en el año 2015.

You Only Look Once

En los años 2015 se publicó [8], el cual presentó un nuevo enfoque para la detección de objetos, un nuevo algoritmo de una etapa que logra una velocidad de entrenamiento y detección considerablemente más rápida que los algoritmos de dos etapas ya conocidos, los autores de dicha publicación reportan la

detección a 45 cuadros por segundo.

Una de sus mayores innovaciones es su forma de analizar la imagen, separando en una cuadrícula de n x n cuadrados para ser analizados por separado, facilitando el cómputo en paralelo, acelerando tanto el proceso de la predicción como el de entrenamiento.

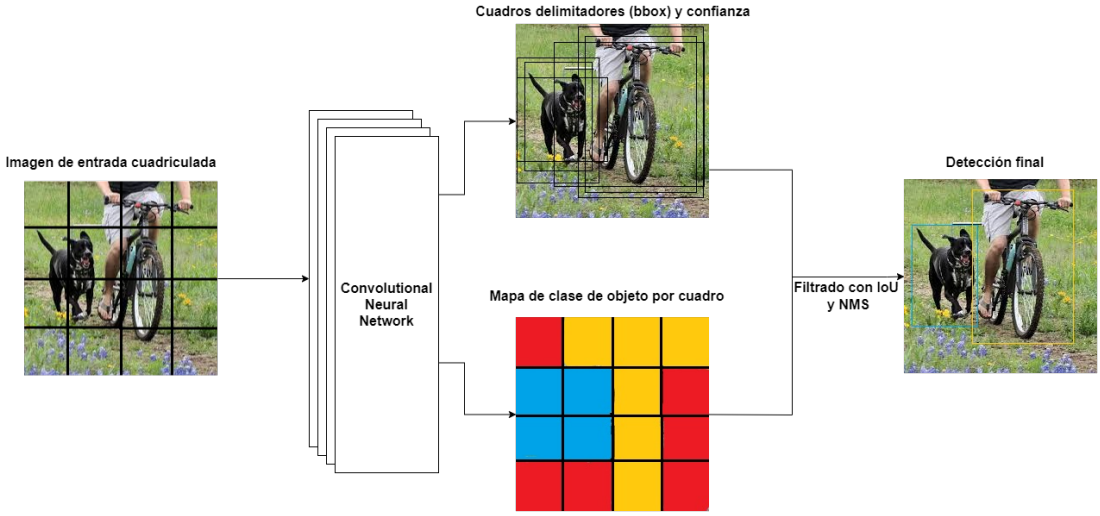


Figura 2.4: Modelo YOLO.

En la figura 2.4 se puede observar un ejemplo paso por paso de la aplicación de este algoritmo, el cual parte separando la imagen de entrada en una cuadrícula de 4x4 rectángulos. A cada uno de estos se aplica a la CNN de una etapa, extrayendo la clase de objeto (si hay alguno presente), candidatos a cuadro delimitador y confianza de predicción de este objeto, los que se pueden apreciar como el mapa de clases con la imagen con cuadros delimitadores, Utilizando esta información el algoritmo filtra mediante la técnica de *'Non-Maximum Suppresion'* (NMS), la cual se basa en la *'Intersection over Union'* (IoU) que se calcula como la fracción entre la intersección entre dos cuadros delimitadores y la unión entre estos, como resultado se selecciona el mejor cuadro delimitador por cada detección.

En cuanto a la matemática de este algoritmo, los mapas de clases y de cuadros delimitadores se pueden expresar como una matriz de tres dimensiones o tensor de 4x4xB, siendo B el tamaño del vector de información $[P_c, X, Y, W, H, C_1, C_2, ..., C_n]$ donde P_c es la probabilidad de presencia de un objeto en el cuadro, (X, Y, W, H) corresponde a la posición, ancho y alto del cuadro delimitador y C_i la confianza de que el objeto clase i se detecta.

2.4 Algoritmos de seguimiento

En la actualidad existen algoritmos de seguimiento de uno o múltiples objetos, sin embargo, para el enfoque de este proyecto solo es de interés un solo tipo de objetos que son las patentes, por ello solo se abordarán las alternativas de seguimiento de objeto simple.

Dentro de los algoritmos de seguimiento de un solo tipo de objeto, la mayoría de estos presentan una estructura similar, que se puede separar en cinco etapas: modelo de movimiento, extracción de características, modelo de observación, método de actualización y método de integración.

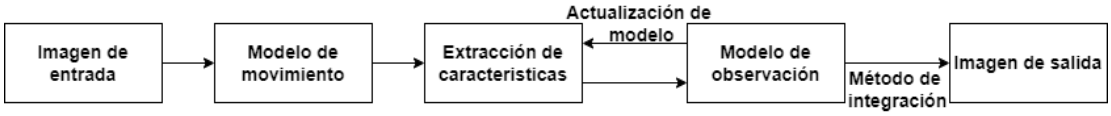


Figura 2.5: Diagrama de flujo de seguimiento de objeto simple.

Las etapas expuestas en la figura 2.5 consisten en:

- El modelo de movimiento se encarga de proponer múltiples posiciones posibles del objeto a seguir, estos modelos se basan en ventanas móviles generalmente.
- La extracción de características se ejecuta para cada una de las posiciones propuestas, para generar vectores para calcular su similitud con el objeto original, las características comunes a extraer son: HOG, histogramas, colores, etc.
- El modelo de observación cumple con la función de determinar si la posición propuesta corresponde al objeto a seguir mediante *'Template matching'*. Si este módulo encuentra una coincidencia se actualiza el modelo para la comparación.
- Finalmente, el método de integración se encarga de filtrar las múltiples posibles posiciones calculadas del objeto para obtener la posición final.

En la familia de seguimiento de un solo objeto existen tres grandes familias de algoritmos, las que se basan en movimiento, los que se basan en la correlación y los basados en CNN y *'deep learning'*, sin embargo en la incumbencia de este proyecto, solo se investigó sobre las alternativas de movimiento y correlación para simplificar la implementación.

OpenCV dispone de múltiples algoritmos de seguimiento, tales como:

- ***MedianFlow Tracker***: Basado en el método de Lucas-Kanade, el cual consiste en extraer el patrón de movimiento de cada píxel, generando un nuevo mapa llamado flujo óptico con el cual se puede predecir la nueva posición del objeto en el siguiente frame.
- ***BOOSTING Tracker***: Algoritmo basado en el algoritmo de AdaBoost, que optimiza los pesos de los clasificadores para focalizar la detección del objeto. El seguimiento se realiza calculando un puntaje por píxel, el que indica si pertenece al objetivo.
- ***Multiple Instance Learning Tracker* (MIL)**: Similar al BOOSTING tracker con la diferencia que este algoritmo postula y asigna un puntaje a múltiples instancias del objeto, haciendo más robusto a ruido, con su mayor problema en su lentitud y no poder eliminar el tracker cuando el objeto se va del cuadro.
- ***Kernelized Correlation Filter Tracker* (KCF)**: Algoritmo que absorbe los métodos de BOOSTING tracker y MIL, con la mejora de la implementación de un filtrado por correlación, para así evitar el seguimiento a una posición donde el objeto ya no este presente, mejorando en su precisión y velocidad.
- ***Minimum Output Sum of Squared Error Tracker* (MOSSE)**: Primer algoritmo que introduce el filtrado por correlación para el seguimiento de objetos, basado en la correlación adaptativa en el espacio de Fourier, este filtrado se encarga de minimizar el error de la salida de la correlación real y la predicha, resultando en un algoritmo mucho más rápido que MIL o KCF, pero con el problema de que falla cuando el objeto cambia de tamaño.

Por otra parte, la librería Dlib dispone de un tracker de correlación basado en la publicación [4], esta propone una mejora para el algoritmo de MOSSE, con el objetivo de mejorar el seguimiento a objetos que cambian de tamaño, utilizando el escalado piramidal para estimar el tamaño del objeto, este escalamiento se ejecuta en la transición del método de integración de la figura 2.5.

2.5 Algoritmos de reconocimiento óptico de caracteres

En los estándares actuales, los algoritmos de reconocimiento óptico de caracteres, OCR por sus siglas en inglés, pueden ser ejecutados de manera online con los proveedores de servicios en la nube como *Azure* o *Amazon web services* con un cobro por solicitud, si bien estos servicios cuentan con una gran precisión al reconocer, fallan en su retraso al enviar hacia el proveedor y esperar los resultados.

Por otro parte, existen alternativas offline y de código abierto para el uso general, los más conocidos

y renombrados son TesseractOCR, desarrollado por inicialmente por Hewlett-Packard y soportado por Google pasado el año 2018; EasyOCR, desarrollado y soportado por JaidedAI; y KerasOCR, desarrollado el año 2019 por Fausto Morales basado en la librería Keras.

En general, todos estos algoritmos funcionan de una forma similar que se puede separar en tres etapas: Preprocesamiento, segmentación de caracteres o palabras, reconocimiento de caracteres. Dentro de los tres algoritmos de código abierto, TesseractOCR tiene una estructura más estándar, la cual se puede observar en la figura 2.6, utilizando el método de umbral adaptativo para binarizar la imagen con el algoritmo Otsu y el análisis de componentes conectadas para segmentar palabras y líneas de la imagen, finalmente la etapa de reconocimiento se ejecuta en dos etapas, donde inicialmente se crea una lista corta de caracteres posibles, para cada carácter a reconocer se calcula un vector de coincidencia con respecto cada carácter posible, los caracteres con mayor coincidencia pasan a la segunda etapa, en la cual se genera una lista de configuraciones posibles a la palabra original y se compara con todas las combinaciones de caracteres, llamadas ”configuration”. Las palabras reconocidas pasan como datos de entrenamiento al clasificador adaptativo para poder reconocer de mejor forma texto que siga al reconocido.

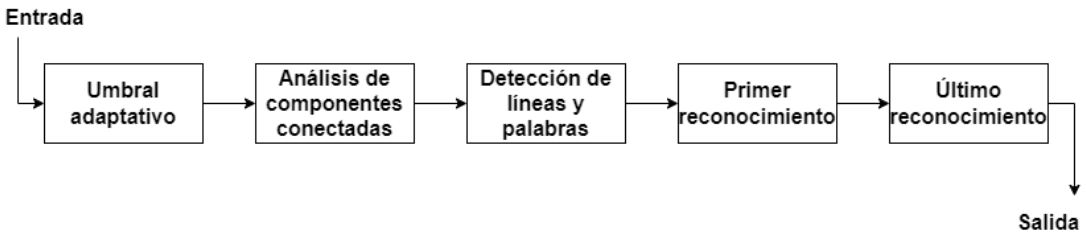


Figura 2.6: Diagrama de bloques de TesseractOCR.

En cuanto al algoritmo EasyOCR la literatura presente en su Git [6] presenta un diagrama de bloques que se resumen en la figura 2.7. En cuanto a todos los procesamientos no se encontró información sobre que métodos utiliza. Para la detección y segmentación de palabras utiliza el algoritmo *'Character-Region Awareness For Text detection'* CRAFT que en pocas palabras puede detectar la posición de cada carácter y los espacios presentes entre ellos, separando las palabras por los espacios detectados.

El modelo de reconocimiento es una mezcla entre una CNN y una *'Recurrent Neural Network'* RNN, utilizando una CNN del tipo RestNet para la extracción de características y una RNN del tipo *'Long Short-Term Memory'* LSTM para capturar el contexto de cada carácter, finalmente mediante *'Connectionist Temporal Classification'* CTC [5] generar palabras como una secuencia de caracteres extraídos y con el decodificador *'Greedy'* elegir la palabra con mayor índice de confianza.

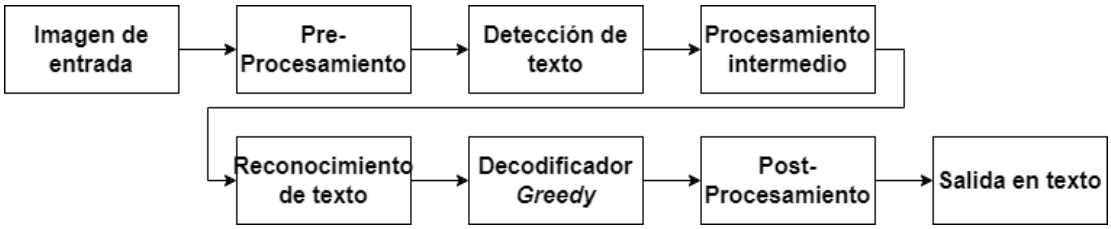


Figura 2.7: Diagrama de bloques de EasyOCR.

A modo de comparación, EasyOCR se enfoca en la rapidez basándose en la librería PyTorch para el cómputo de redes neuronales y CUDA para la aceleración por GPU, en comparación con TesseractOCR es considerablemente más rápido cuando hay uso de una GPU, mientras que en la ausencia de GPU TesseractOCR es más rápido, en cuanto a la precisión, EasyOCR funciona mejor con el reconocimiento de números y Tesseract es mejor para imágenes con solo letras.

2.6 Librerías de interfaz de usuario

Existen varias librerías que cumplen el propósito de entregar funciones, clases y widgets que facilitan el desarrollo de interfaces simples, entre las más utilizadas se encuentran Tkinter basada en el software Tk y PyQT basada en el software QT, ambas permiten desarrollar aplicaciones multiplataformas.

Por una parte, Tkinter se considera el estándar para las aplicaciones desarrolladas en Python, implementando un intérprete de Tcl, un lenguaje de alto nivel que ejecuta cada línea del código como un comando, que junto a Tk permite la creación de interfaces gráficas de usuario. Su implementación se realiza mediante tres unidades básicas: ventana, widget o artilugio y marco, siendo la ventana el cuadro mayor que contiene toda la interfaz, los widgets corresponden a los objetos que interactúan con el usuario como por ejemplo una entrada de texto o un botón y se distribuyen ordenadamente con la ayuda de los marcos que estructuran en cuadros estos widgets.

En cuanto a PyQT es una alternativa interesante debido a que esta separa el desarrollo de la interfaz gráfica de usuario (GUI) con el desarrollo de los scripts que se ejecutan al interactuar con sus widgets, el manejo de eventos se realiza mediante lo que se llama señales y ranura, al momento de interactuar con un botón, este genera una señal hacia una ranura que ejecuta una acción que podría ser una función escrita en código. Una de las ventajas más llamativas de esta alternativa es su software opcional *Qt Designer*, el cual permite crear la GUI incluyendo los widgets de manera visual y enlazando sus señales, para luego exportarlo a un archivo con el código equivalente con un simple comando de consola.

3 Diseño del sistema

Este proyecto busca desarrollar un sistema de reconocimiento de patentes para vídeos. En busca de ese objetivo se propone un sistema que se diseñó en el lenguaje Python y con el paradigma de programación orientada a objetos. En total se desarrolló una clase de interfaz usuario y cuatro clases para ejecutar el bucle de reconocimiento expuesto en la figura 3.1. Si bien el sistema no se diseñó para funcionar a tiempo real, con ayuda de la plataforma CUDA y una GPU se puede lograr funcionamiento a tiempo real.

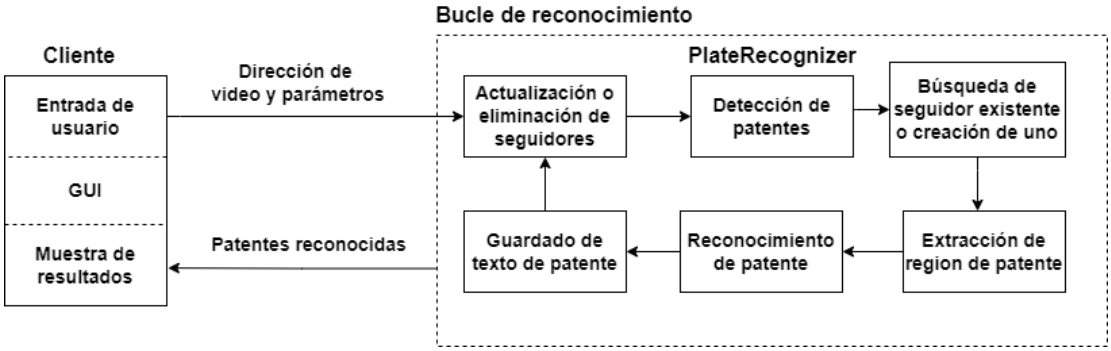


Figura 3.1: Diagrama de bloques del sistema.

El sistema recibe como entrada un archivo de vídeo almacenado localmente e ingresado por el usuario mediante una interfaz de usuario simple que además permite el ingreso de algunos parámetros para modificar levemente el funcionamiento del bucle de reconocimiento, con el objetivo de acelerar el proceso y disminuir la carga en la memoria. El vídeo entregado por el usuario es analizado cuadro por cuadro, detectando las patentes presentes y creando un seguidor por cada una, tanto la detección como la extracción de la patente junto con su reconocimiento se ejecutan cada una cierta cantidad de cuadros ajustado por el usuario, una vez la patente detectada deja de estar presente en el cuadro la interfaz de usuario debe mostrar el resultado del reconocimiento de una manera simple y clara.

El objetivo del sistema propuesto es extraer datos de las patentes presentes en el vídeo, que pueden ser de interés para el usuario para luego ser analizados con el enfoque que se desee, por ello además de mostrar los resultados, la interfaz de usuario también debe permitir crear un archivo en formato *Comma-separated values* (Csv), debido a que este formato es bien utilizado en el mundo de la minería de datos.

3.1 Diseño de interfaz gráfica

El funcionamiento mínimo para la interfaz gráfica tiene que asegurar tres funciones principales:

- Permitir el ingreso del vídeo mediante un explorador de archivos para facilitar el uso para el usuario
- Permitir la modificación de los parámetros de cantidad de detección, reconocimiento por segundo y parámetros del reconocedor.
- Disponer de un conjunto de opciones para ejecutar el reconocimiento, para borrar resultados previos y para guardar el resultado en el archivo .csv.
- Disponer de una tabla que se debe ir rellenando con las patentes ya reconocidas y fuera del cuadro de la imagen. Esta tabla, aparte de la imagen y el texto de la patente reconocida, debe mostrar índices como el de confianza, tamaño y tiempo en el que aparece.
- Asegurar que solo exista un vídeo siendo analizado.
- La interfaz de usuario debe ser multiplataforma, al menos para Windows y Linux.

3.2 Diseño de backend

El cerebro que está detrás de la interfaz de usuario se diseñó como una clase llamada PlateRecognizer, la cual debe ser instanciada por un evento gatillado por un botón de la interfaz de usuario y debe permitir la consulta de su estado en cuanto a las patentes detectadas y las que se dejan de detectar para entregar la información al cliente.

Esta clase debe ser capaz de configurar e instanciar con los parámetros entregados los objetos de detección, seguimiento y reconocimiento para ser utilizados dentro de un bucle ejecutado de forma paralela en un hilo aparte del cliente, con el objetivo de generar un diccionario con la información extraída de las patentes presentes en el vídeo.

El funcionamiento básico se resume en las siguientes funciones:

- Instanciar y configurar los objetos de detección, seguimiento y reconocimiento.
- Generar un hilo con el bucle para el reconocimiento.
- Guardar la información extraída con los objetos de detección, seguimiento y reconocimiento.
- Disponer de métodos para entregar estado e información de las patentes detectadas.
- Mostrar a tiempo real la ejecución el sistema.

3.2.1 Diseño de *PlateDetector*

Clase dedicada solo a la detección de patentes con el modelo YOLO de detección como su único atributo, la cual debe asegurar los siguientes requisitos funcionales:

- Permitir la carga del modelo de detección como atributo.
- Disponer de un método de detección que entregue una salida estandarizada como una lista de arreglos con la posición, ancho y alto de las patentes detectadas.
- Disponer de un módulo que permita extraer la región de interés de una patente dentro de una imagen, admitiendo un porcentaje de holgura configurable como entrada para eliminar detecciones incompletas.
- Filtrar detecciones mediante el índice de confianza.

En cuanto al rendimiento, debe ser capaz de detectar todas las patentes presentes en cada imagen y tener un muy mínimo porcentaje de falsos positivos, para evitar ejecutar el resto del algoritmo en imágenes que no corresponden a patentes.

3.2.2 Diseño de *PlateTracker*

Para la creación, actualización y eliminación de seguidores o tracker, se diseñó la clase PlateTracker que posee como atributo un diccionario que almacena todos los seguidores activos y debe poseer al menos los siguientes métodos:

- Un método para la creación de un tracker específico con una ID única.
- Un método para identificar si existe un tracker ya realizando el seguimiento a una patente entregada como entrada.
- Un método para actualizar las posiciones de todos los trackers activos.
- Un método para eliminar trackers mediante filtrado de calidad del seguimiento o tiempo sin ser detectado.
- Método alternativo para identificar si dos rectángulos están sobrepuestos.

3.2.3 Diseño de *CustomOCR*

En cuanto al reconocimiento de caracteres de la patente, se diseñó la clase CustomOCR que mediante un objeto que extraiga los caracteres de una imagen (EasyOCR, TesserectOCR, etc) sea capaz de filtrar, estandarizar y generar una salida limpia y fácil de manejar.

Esta clase debe ser capaz de distinguir entre líneas importantes y opcionales, como ejemplo para las patentes, identificar la línea de los caracteres de la patente y eliminar las lecturas de los tornillos y de la línea que dice 'Chile'.

Los métodos necesarios son:

- Método de lectura de imagen, la cual debe filtrar por posición y tamaño las líneas encontradas para la elección de la línea correcta, además de eliminar ruido como los producidos por los tornillos
- Método que con una entrada de una lista de resultados extraídos, encontrar la mejor lectura mediante el resultado más frecuente y con un formato correcto.

4 Implementación del sistema

Para la implementación y ejecución del sistema automático de reconocimiento de patentes se utilizó las siguientes librerías:

- **Dlib**: Librería de uso general, se utilizó el seguidor de correlación. Versión 19.22.1.
- **OpenCV**: Librería de visión por computador de código abierto, se utilizó para el manejo de las imágenes y vídeo. Versión 4.5.4.
- **Torch**: Librería de *Machine learning* utilizada para la implementación del modelo YOLO, con implementación nativa de CUDA para la aceleración por GPU. Versión 1.10.2+cu113.
- **EasyOCR**: Librería para el reconocimiento óptico de caracteres. Versión 1.4.2
- **Tkinter**: Librería para la implementación de interfaces de usuario. Versión 8.6
- Librerías básicas como imutils, para calcular los cuadros por segundos del rendimiento del sistema, numpy para las operaciones de arreglos y threading para la implementación del hilo reconocedor.

4.1 Implementación de interfaz gráfica

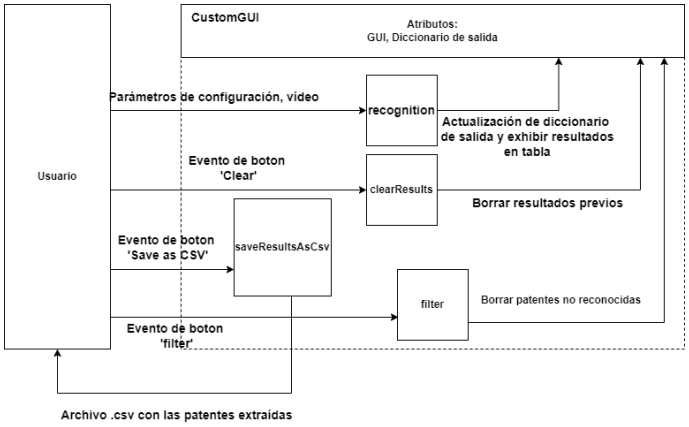


Figura 4.1: Diagrama de bloques de la interfaz gráfica.

La interfaz gráfica implementada se puede observar en la figura 4.2, esta organizada en filas de widgets, como primera fila se tiene tres campos de texto para que el usuario pueda ingresar los parámetros de cuadros por reconocimiento, para configurar la frecuencia del reconocimiento; cuadros por detección, para configurar la frecuencia de detección; y 'Magnification ratio', para aumentar el tamaño de las

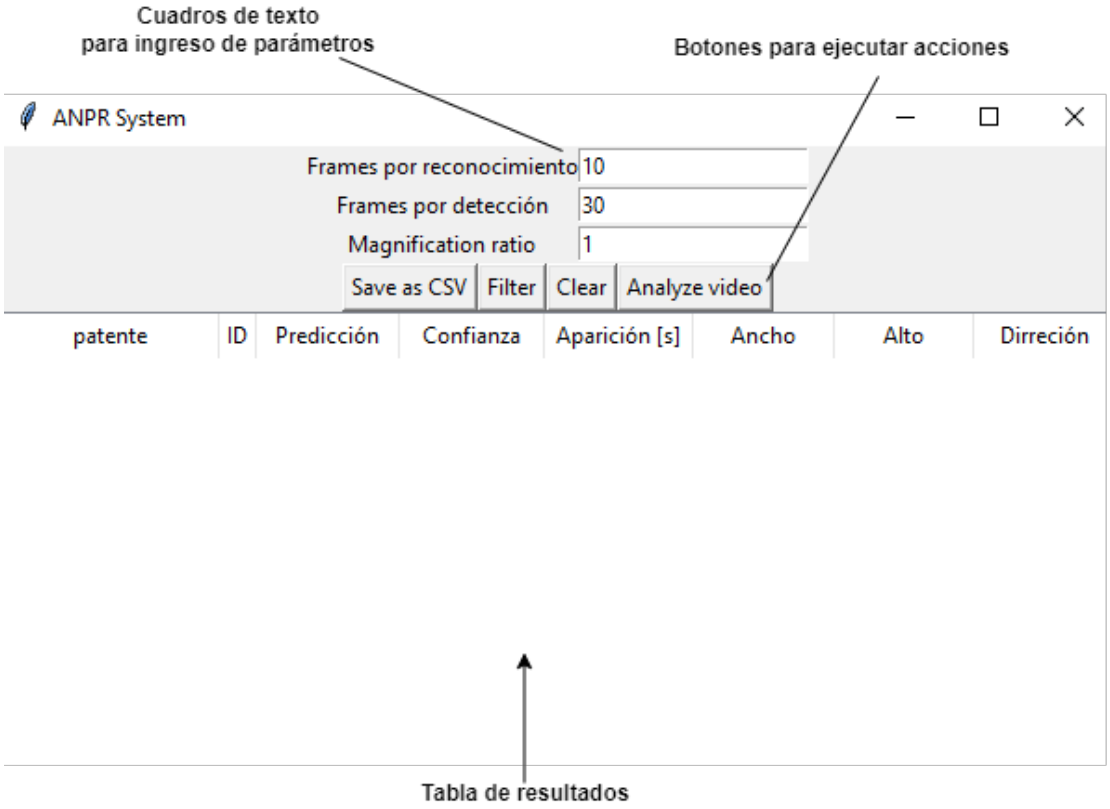


Figura 4.2: Interfaz de usuario.

patentes al ser reconocidas. Modificando estos tres parámetros el usuario tiene la opción de ejecutar el sistema de una forma más rápida pero menos precisa y viceversa.

Las acciones que el usuario puede ejecutar son cuatro, analizar el vídeo, limpiar los resultados anteriores, filtrar las patentes con lecturas parciales y guardar el resultado actual en formato .csv. Para esto se implementa una fila de botones para ejecutar los scripts, donde el más complejo es el botón *Analyze video*, el cual abre un explorador de archivos para ingresar el vídeo de una manera simple para el usuario. Con la dirección del vídeo ingresada y los parámetros en los cuadros de textos se ejecuta el bloque 'recognition' de la figura 4.1, el cual instancia un objeto de clase PlateRecognizer para ejecutar el bucle de reconocimiento, mientras que la interfaz gráfica se mantiene en un bucle para preguntar el estado del reconocedor para mostrar y guardar la información de las patentes que salen de escena.

La tabla de resultados de la figura 4.3 se va llenando a medida que el objeto PlateRecognizer responda el número identificador y entregue la información de las patentes que salen de escena o se dejan de seguir, mostrando una pequeña imagen de la patente junto con su ID, el texto extraído, índice de confianza,

tiempo de aparición, promedio del alto y ancho de dicha patente y la dirección, la cual se obtiene al restar posición final menos la inicial en el eje Y, en caso de ser positivo la patente se dirige hacia la cámara (in) y se aleja de la cámara en caso de ser negativo (out). Si el usuario lo necesita con el botón 'Filter' se eliminan los datos con lecturas incompletas para solo visualizar y guardar lecturas completas.

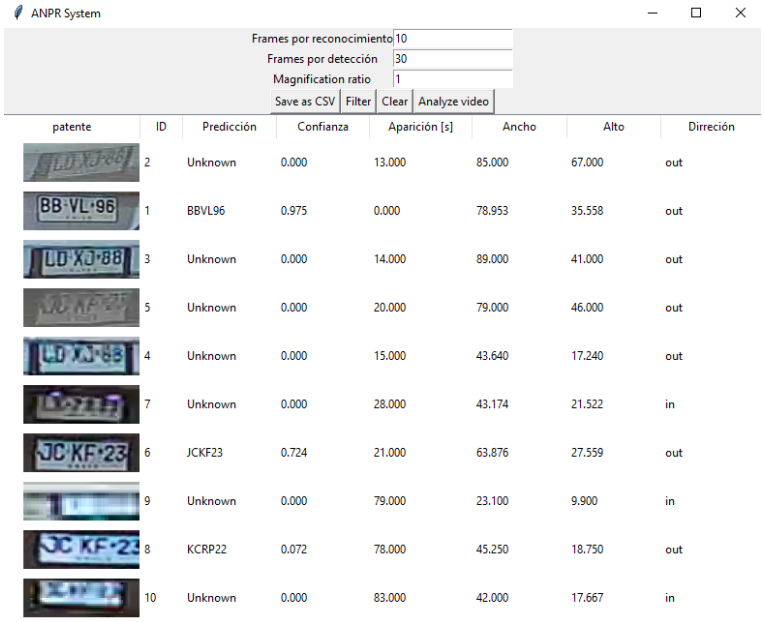


Figura 4.3: Interfaz de usuario con resultados.

4.2 Implementación de PlateDetector

Para la detección se seleccionó el algoritmo YOLOv5 por su gran rapidez y precisión aceptable. Para poder implementar un modelo YOLOv5, se necesitó cargar este modelo dentro del objeto hub de la librería pyTorch, el cual permite cargar un modelo pre-entrenado y el repositorio Git de YOLO para la ejecución de la detección mediante este algoritmo.

El diagrama de bloques se puede observar en la figura 4.4, donde las entradas son solamente el cuadro al momento para detectar y el cuadro sumado a el cuadro delimitador de una patente en especifico para cortar la imagen, para ser entregada al PlateRecognizer para ser guardada.

Conjunto de datos

Para entrenar el modelo de detección se generó un conjunto de 301 imágenes manualmente etiquetadas, donde 224 fueron utilizadas para el entrenamiento y 77 para la validación, 74.5% y 25.5% respectivamente. Las imágenes seleccionadas pueden separarse en dos grupos principales, un grupo de imágenes

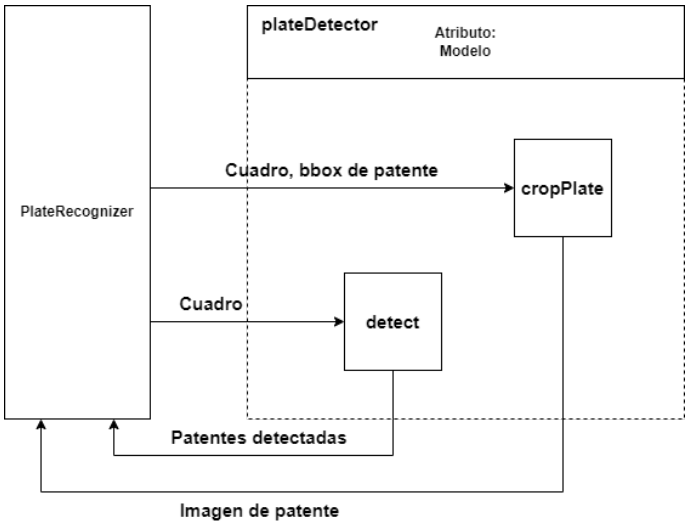


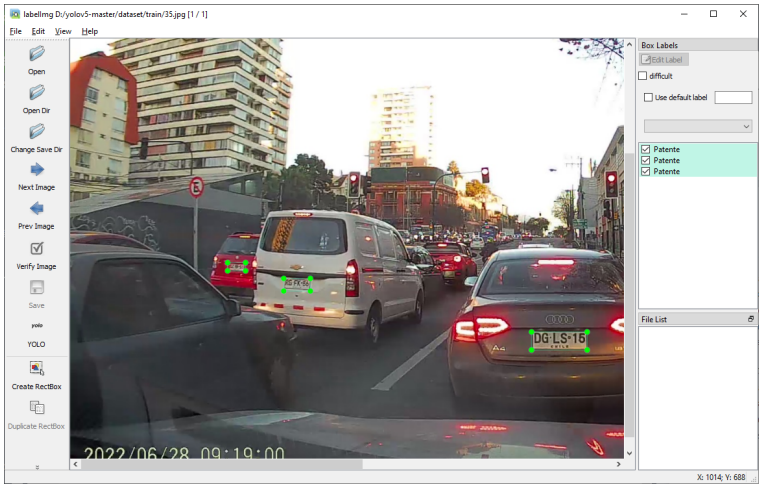
Figura 4.4: Diagrama de bloques del módulo de detección.

con un solo auto en primer plano con buena calidad de imagen y un grupo de imágenes con múltiple autos en diversas posiciones, para mejorar el entrenamiento se generaron dos grupos extras, un pequeño grupo de 13 imágenes de fondo sin presencia de patentes para disminuir las predicciones de falsos positivos y un grupo de imágenes volteadas horizontalmente para hacer el modelo más robusto a otras posiciones de patentes. Un ejemplo de cada grupo se puede observar en figura 4.5.

Figura 4.5: Muestra de los grupos del conjunto de datos. (a) Imagen con auto en primer plano, (b) Imagen de cámara de vehículo, (c) Imagen de autos sin patentes presentes, (d) Imagen reflejada horizontalmente.

En cuanto al etiquetado manual, se utilizó el software de código abierto *Labelimg* que permite trazar los

cuadros delimitadores y guardar la posición de este cuadro en un archivo de texto con el mismo nombre de la imagen, este software tiene como opción el formato de YOLOv5 para el guardado de la información, el cual consiste en un vector separado por espacio con los datos de ID de la clase de objeto, posición del centro, ancho y alto normalizada con respecto al tamaño de la imagen original, 'c Cx Cy w h'. En la figura 4.6 se presenta un ejemplo de etiquetado.




```
0 0.089844 0.721759 0.031771 0.032407
0 0.303385 0.649537 0.018229 0.012037
0 0.460156 0.643519 0.010937 0.007407
```

Figura 4.6: Ejemplo de etiquetado con Labelimg y sus respectivas posiciones en formato para entrenamiento

Entrenamiento del modelo

Por comodidad del entrenamiento del modelo, este se realizó de manera online en Google Colab el cual dispone de entrenamiento acelerado por GPU, para ello se utilizó el dashboard o cuadro de mando *Weights and Biases* (wandb) para subir el conjunto de datos y visualizar los resultados e índices del entrenamiento. El entrenamiento se ejecutó con los siguientes parámetros:

```
python train.py --data custom_dataset.yaml --epochs 100 \
--project project_wandb_name --save-period 1 --weights yolov5s.pt
```

Donde custom_dataset.yaml es un archivo con la cantidad y nombres de las clases a detectar junto con la dirección de los conjuntos de entrenamiento y validación dentro del proyecto wandb de nombre project_wandb_name, se entrenó a 100 épocas, demorando unos 30 minutos aproximados en entrenar el

modelo. El repositorio de YOLOv5 dispone de cinco arquitecturas de modelo dependiendo de su tamaño y complejidad, se seleccionó el modelo pequeño *yolov5s* porque solo se entrenó una clase, por lo que tener un modelo más complejo aportaría mayores tiempos de detección (debido al aumento de parámetros y tamaño de red neuronal) y una mínima mejora en la detección.

Wandb permite observar las predicciones realizadas al conjunto de validación para tener una prueba visual, la cual se puede observar en las figuras 4.7 y 4.8, con buenos resultados y llegando a un 96,7% de precisión para todo el conjunto de validación.

Figura 4.7: Imagen del set de validación con las etiquetas reales.

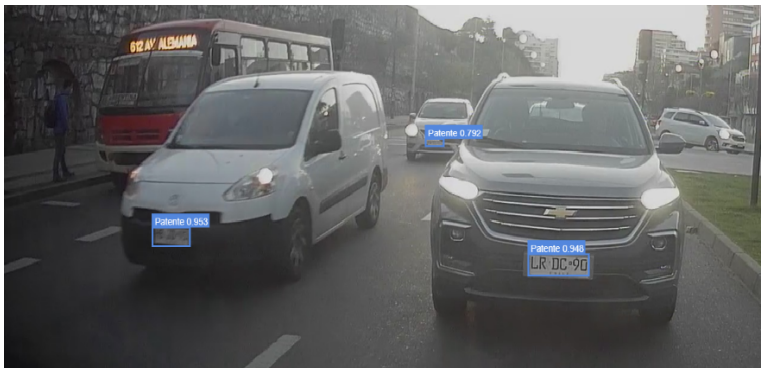


Figura 4.8: Imagen del set de validación con las etiquetas predichas.

Ajustes en la implementación

Una vez entrenado el modelo, se estandarizó la salida de la detección para facilitar su manipulación para el sistema, por defecto PyTorch entrega un tensor con los resultados predichos, mediante unas operaciones simples se extrae la información y se entrega con el formato de un arreglo con la posición de la esquina superior izquierda con su ancho y alto del cuadro delimitador junto con su índice de confianza. Además para filtrar detecciones incorrectas del detector, se fijó un umbral de confianza de 0.4 para tomar la

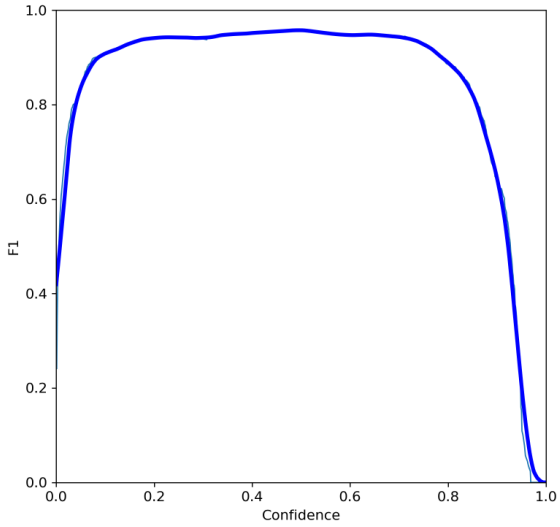


Figura 4.9: Curva F1-score del modelo.

predicción como válida, este umbral se seleccionó basándose en la curva F1 que entregó el entrenamiento, la cual se puede observar en la figura 4.9, donde se puede observar que el máximo puntaje F1 se obtiene en el intervalo de confianza de 0.4-0.7 aproximadamente, por lo que se elige el mínimo de este intervalo, debido a que para el interés de este proyecto es peor no detectar una patente (filtrada por índice de confianza muy bajo) a detectar una patente falsa al no ser filtrada por un umbral muy alto al filtrar.

4.3 Implementación de PlateTracker

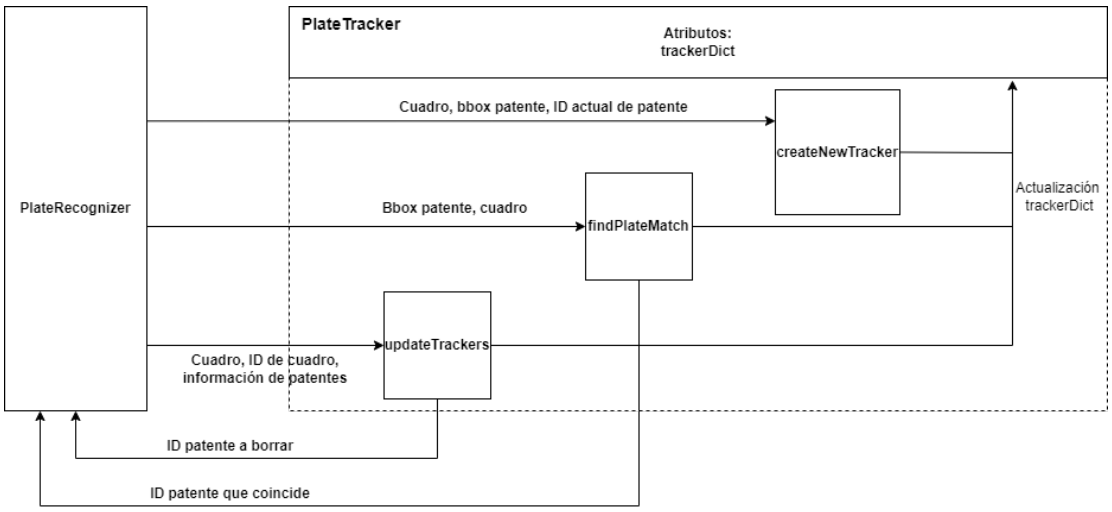


Figura 4.10: Diagrama de bloques del módulo de seguimiento.

El seguimiento de patentes se implementó con el tracker de correlación de la librería Dlib por su

rapidez y capacidad de escalar el tamaño del objeto seguido. El objeto se instancia con el parámetro ingresado por el usuario de cantidad de cuadros por detección para ser utilizado como una condición de eliminación del seguidor como entrada al método constructor, junto con un diccionario de seguidores vacío como atributo.

Para cumplir con los requisitos del diseño de esta clase, se implementó el método *findPlateMatch* con entradas de un cuadro delimitador de una detección de patente y el cuadro actual, este método recorre los seguidores activos buscando alguna coincidencia entre el cuadro delimitador de entrada y las posiciones entregadas por el seguidor, si el centro del cuadro de entrada está contenido en el cuadro del seguidor y viceversa, quiere decir que la patente de entrada es la misma que la patente seguida por dicho tracker y entregando como salida el ID de la patente.

Al detectar una patente que no está siendo seguida, se crea un nuevo seguidor y se guarda con un número único identificador, mediante el método *createNewTracker* se inicializa el tracker con la imagen, el cuadro delimitador y el ID que le corresponde a esa patente entregada como parámetro.

En cuanto a la actualización de los trackers se realiza con el método *updateTrackers* que recorre el diccionario actualizando con el método dlib.correlation_tracker.update(), el cual predice la nueva posición y entrega un puntaje de calidad de la predicción. Este mismo método es el encargado de eliminar los trackers con dos condiciones, la baja calidad de predicción, con un umbral de 8 seleccionado mediante prueba y error y tiempo sin ser detectado, al cumplir con dos periodos de detección sin ser detectado, este seguidor se elimina y entrega su ID.

4.4 Implementación de CustomOCR

La implementación del reconocimiento se realizó con el algoritmo de EasyOCR, implementada en la librería del mismo nombre, como atributo de la clase CustomOCR se instancia el objeto easyocr.Reader() junto con una cadena de texto con los posibles caracteres a reconocer, como las patentes de Chile no contienen la letra ’ñ’ se configuró el objeto Reader con el idioma inglés y con ’BCDFGHJKLPRSTVWXYZ 0123456789 -’ como caracteres válidos, donde aparte de las letras y números posibles de las patentes, se ingresó los caracteres ’ ’ y ’-’ para detectar los tornillos, escudo de Chile y espacios, los cuales tienden a ser reconocidos como estos símbolos, para así poder identificar fácilmente la presencia de estos y ser filtrados en la salida.

El método de reconocimiento del objeto easyocr.Reader entrega un tensor con las predicciones de cada

palabra encontrada, donde para las patentes existen cuatro posibles palabras, cada par de carácter junto con la palabra Chile en la parte inferior, en casos donde la patente tiene baja resolución estos pares de caracteres se pueden juntar en diferentes formas presentes en la figura 4.11 y 4.12.

Figura 4.11: Lectura con palabras separadas

Figura 4.12: Lectura con dos pares juntos

Para cumplir con los requisitos funcionales del diseño, se implementó el método *readText* para juntar todas las lecturas en un solo string con los caracteres reconocidos sin ningún espacio, para ello este método recorre el tensor de salida del objeto Reader utilizando la primera lectura como pivote y concatenando las palabras reconocidas en una posición vertical similar a la del pivote (+-5%) con la intención de filtrar la linea de Chile. Otro problema que puede ocurrir en el reconocimiento es la inclinación de la patente presente en la figura 4.13, para ello también se concatenan las palabras de una altura similar a la del pivote (+-10%).

Figura 4.13: Lectura de patente inclinada.

En cuanto al formato de salida, se tiene como resultado un arreglo de dos datos, el texto reconocido y el índice de confianza de esta predicción. Este índice se calcula como el promedio simple entre los índices de confianza de cada conjunto de caracteres reconocidos. En caso de que el reconocedor no encuentre ninguna caracter, la salida por defecto es 'Unknown'.

Esta clase también dispone de un método para seleccionar el texto de la patente más frecuente recono-

cida, para ello el método *getMostFrequent* recibe como entrada un arreglo de cadenas de texto con las predicciones realizadas, contando las veces que se repiten las patentes con un reconocimiento completo, es decir, con 6 caracteres reconocidos y extrayendo la con mayor cantidad de repeticiones, en caso de que ningún reconocimiento está completo, la salida por defecto es 'Unknown'

Entrenamiento de modelo particular

Con el objetivo de mejorar la precisión al momento de reconocer la patente, se entrenó un modelo de reconocedor EasyOCR con el script oficial del repositorio [6], para ello se generó un conjunto de datos con imágenes de líneas de texto con las dos fuentes utilizadas actualmente para las patentes de Chile, 'Helvética Medium Condensed' y 'FE-Shrift'.

La generación del conjunto de datos se realizó con el software de código abierto 'Text Recognition Data Generator' (trdg), el cual permite generar imágenes aleatorias con las fuentes indicadas, además de permitir agregarles ángulo de inclinación y difuminación. En total se generaron 1000 imágenes para cada fuente con un 50% imágenes claras y sin ángulo, 25% imágenes con un ángulo aleatorio entre -10° a 10° y un 25% imágenes con difuminación gaussiana, con un total de 2000 imágenes, se utilizó un 75% y un 25% de entrenamiento y validación respectivamente. Una muestra del conjunto de datos se presenta en la figura 4.14.

Figura 4.14: Ejemplo del conjunto de datos. (a) Texto con Helvética, (b) Texto con FE-Shrift, (c) Texto con inclinación y (d) Texto con difuminación

Por defecto el entrenamiento se realiza a 300.000 épocas pero por falta de capacidad del computador donde se desarrolló, el primer intento de entrenamiento llegó a las 20.000 épocas y superando la memoria disponible de la GPU, también se intentó entrenar en la nube de Google Colab donde se logró entrenar hasta 150.000 épocas debido a el tiempo limite de la versión gratis de Google Colab. Los dos modelos entrenados con las fuentes de Chile no resultaron ser mejores que el modelo por defecto de EasyOCR, por lo que no se implementaron y los detalles se abordan en el capítulo de pruebas.

4.5 Integración de clases

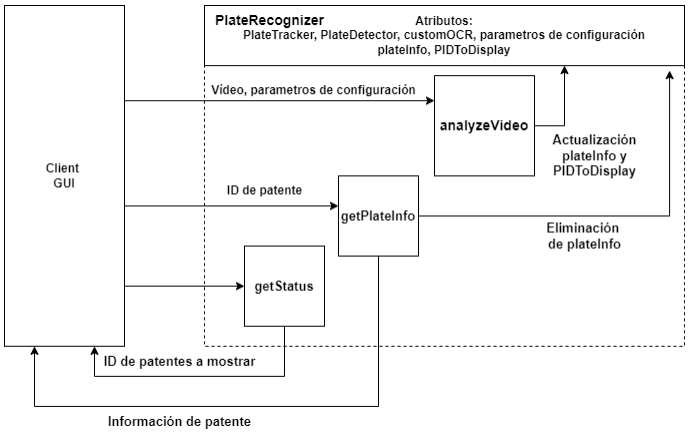


Figura 4.15: Diagrama de bloques del módulo de reconocimiento.

La clase PlateRecognizer es la encargada de integrar las clases de detector, seguidor y reconocedor como sus atributos junto al diccionario de información extraídas de las patentes. Su principal función es ejecutar un hilo con el bucle de reconocimiento referido en la figura 3.1 dentro del método *analyzeVideo*.

El bucle se ejecuta hasta que el vídeo sea analizado completamente cuadro por cuadro, donde para cada frame se actualiza los trackers dentro del objeto PlateTracker, si se elimina un tracker por los filtros explicados anteriormente se agrega el ID correspondiente para ser mostrado en la GUI.

Con el parámetro 'frame por detección', se fija cantidad necesaria de cuadros sin detección para ejecutar una nueva detección del objeto PlateDetector, entregando los cuadros delimitadores de las patentes encontradas, para cada una de estas se ejecuta el método de *findPlateMatch()* del PlateTracker, en caso de no encontrar coincidencias, se crea un nuevo tracker y se inicializa el ítem del diccionario de informaciones con los siguientes tópicos: ancho, alto, número del primer frame encontrada, número del último frame detectada, imagen de la mejor predicción, arreglo de mejor predicción con su índice de confianza y un arreglo vacío para guardar las predicciones.

Dentro del mismo bucle, se extrae y reconoce las patentes que están siendo seguidas cada x cuadros configurados por el usuario en el campo de texto de 'frames por reconocimiento', la extracción se la imagen de la patente se realiza con una holgura del 10% para evitar imágenes de la patente cortada y perder información. El reconocimiento se ejecuta mediante el método *readText()* a cada patente extraída y se actualiza la información de esta, agregando el alto, ancho, predicción e índice de confianza.

Para consultar el estado y mostrar los resultados en la interfaz gráfica, en la clase PlateRecognizer se implementó los métodos *getStatus() y getPlateInfo()*, los cuales entregan los ID de las patentes listas para ser mostradas y la información de dicha patente, de manera que la GUI pregunta cada 0.5 segundos el estado y en caso de existir una patente lista, se solicita la información y se muestra en pantalla. Al ejecutar *getPlateInfo()* se calcula la predicción final con el método *getMostFrecuent* de la clase CustomOCR.

5 Pruebas y resultados

Para las pruebas realizadas, se contó con tres vídeos grabado desde una cámara montada en la parte trasera con resolución 1920x1080 y delantera con resolución 2560x1440 y un conjunto menor de fotos de autos en primer plano. Todas las pruebas fueron realizadas en un computador con un procesador I5-10600KF con una frecuencia de 4.1 GHz base, con una GPU Nvidia GTX 1660 con 6 Gb de memoria y 16 Gb de RAM.

5.1 Pruebas de detección

El conjunto de imágenes de prueba para la detección consistió en un total de 69 imágenes, donde 23 de ellas son de autos en primer plano y 46 son imágenes extraídas de los vídeos, en la figura 5.1 se presentan una muestra por grupo de imágenes. Dentro de las imágenes existen un total de 155 patentes reales, con un máximo de 4 patentes en una sola imagen

(a) (b)

Figura 5.1: Muestra de conjunto de pruebas. (a) ejemplo de auto en primer plano, (b) ejemplo de imagen de cámara delantera.

Las pruebas en general se separaron en cinco grupos dependiendo de la altura real de la patente para observar el comportamiento para diferentes resoluciones de patente y entender de una mejor manera el rendimiento del detector de objetos. Teniendo una altura mínima de 4 píxeles y máxima de 54 píxeles, los grupos se separan en los intervalos de 0-14, 15-24, 25-34, 35-44, 45-54 píxeles de alto. La distribución de patentes en los grupos se puede observar en la figura 5.1.

La prueba consistió en ejecutar la detección en cada imagen y comparar los cuadros delimitadores con el cuadro real ingresado manualmente, mediante el índice *Intersection over Union* (IoU) se verifica si el cuadro detectado corresponde a una patente real si resulta un IoU mayor a 0. Algunos resultados

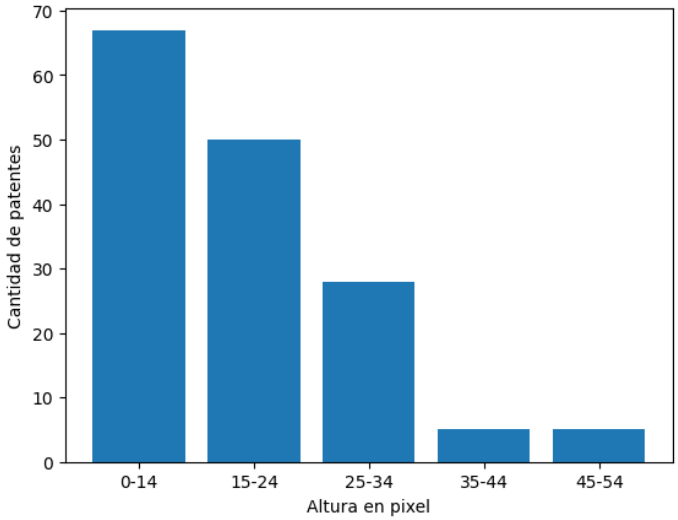


Figura 5.2: Distribución de patentes por altura.

interesantes se pueden observar en la figura 5.3, donde en la imagen (a) se puede apreciar una detección incorrecta inducida por un número de teléfono en el vehículo, mientras que en la imagen (b) se observa una patente sin ser detectada debido a su pequeño tamaño.

Figura 5.3: Imágenes con el cuadro real (Azul) y predicho (Rojo). (a) Imagen con un positivo falso y (b) Imagen con un negativo falso.

En esta prueba si se encuentra una detección con un IoU mayor a 0, lo que indica que existe una intersección entre estas, esta detección se cuenta como una patente correctamente detectada (True positive), en caso de que no se detecte una patente real, se cuenta como una patente no detectada (False Negative) y si la detección entrega un cuadro que no corresponde a una patente real, es decir con IoU igual a 0, se

cuenta como un patente detectada erróneamente (False Positive). Los resultados se resumen en el gráfico de la figura 5.4.

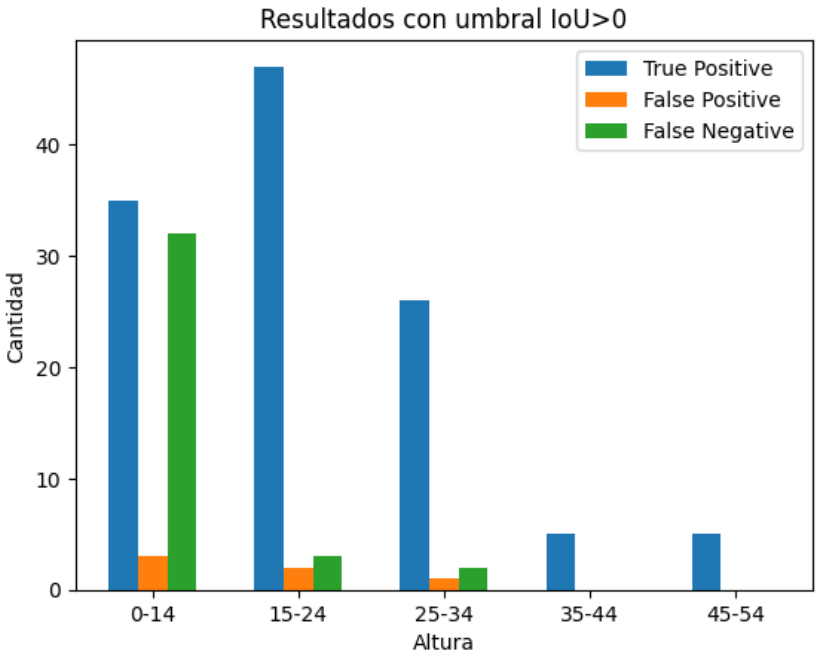


Figura 5.4: Resultados por grupo.

Los resultados indican que, en las patentes de tamaño pequeño, la detección no tiene un rendimiento óptimo, aunque las patentes detectadas son más de la mitad de las patentes reales. Esto puede ser beneficioso para el sistema completo, ya que no se detectarían patentes con muy baja resolución, lo que sería muy complejo realizar el reconocimiento a estas, evitando tiempo de cómputo inútil en cuanto a resultado final. Por otra parte, se observa que las patentes más grandes son detectadas sin ningún problema y en general existe un número despreciable de detecciones falsas.

Con la información del gráfico de la figura 5.4 se calculó los puntajes de *accuracy, presicion y recall* de cada grupo, para medir la calidad de predicción y el rendimiento en cuanto a cantidad de patentes detectadas sobre patentes reales. Estos resultados se presentan en el gráfico de la figura 5.5.

Los resultados de este gráfico indican que para patentes sobre 15 píxeles de alto la predicción es acertada en al menos 9 de cada 10 patentes, dentro de estos puntajes el más importante para esta aplicación es el recall, ya que es de interés detectar la mayor cantidad de patentes reales, el cual es mayor a 0.9 en todos los grupos menos el de menor tamaño. En cuanto al grupo de 0-14 se puede observar una gran precisión,

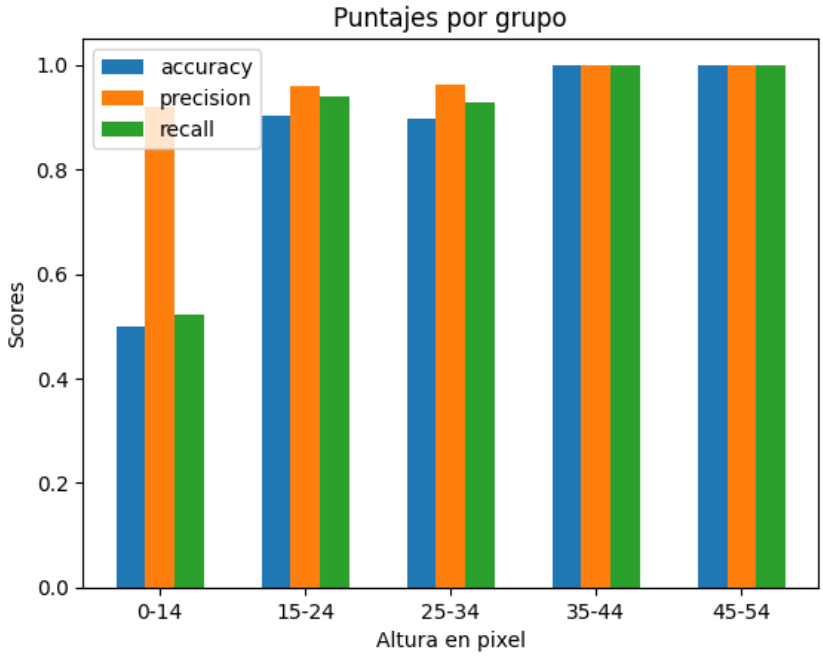


Figura 5.5: Puntajes por grupo.

pero un accuracy y recall bajos, esto se debe a que la precisión es la cantidad correcta de predicciones y en este grupo casi la mitad de las patentes no fueron detectadas, es decir que no hubo muchas predicciones, pero fueron en su mayoría correctas.

Otro parámetro para medir la calidad de la detección es el IoU, de forma que entre mayor sea este índice mejor es la detección, ya que el solapamiento es mayor hasta el punto donde son el mismo cuadro con IoU igual a 1. Con intención de plasmar la calidad de la detección se graficó el promedio del índice IoU de las detecciones correctas por grupo, obteniendo el gráfico presente en 5.6. Dicho gráfico indica que la calidad de detección es mejor entre más grande es la patente, llegando a un promedio de 0.92 para el grupo de mayor tamaño.

Algunos usuarios podrían solicitar un umbral de IoU mayor para que la detección cuente como correcta, entre los más utilizados son IoU mayor a 0.5 o 0.75, con la intención de probar el detector frente a esos requisitos, se gráfica los puntajes de accuracy, precision y recall con los nuevos umbrales, obteniendo los resultados presentes en la figura 5.7:

En los gráficos de la figura 5.7 se puede observar la diferencia con respecto al gráfico 5.5, donde los grupos de 0-14, 15-24 son los únicos afectados para el caso de umbral al 0.5, perdiendo algunas de sus

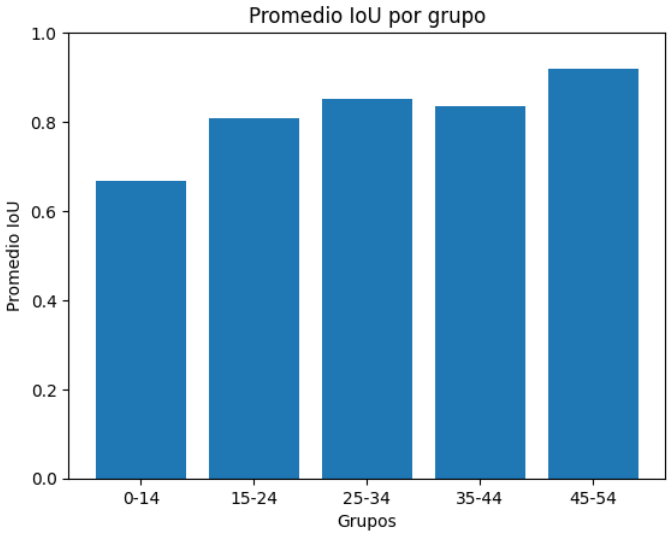


Figura 5.6: Promedio de índice IoU por grupo.

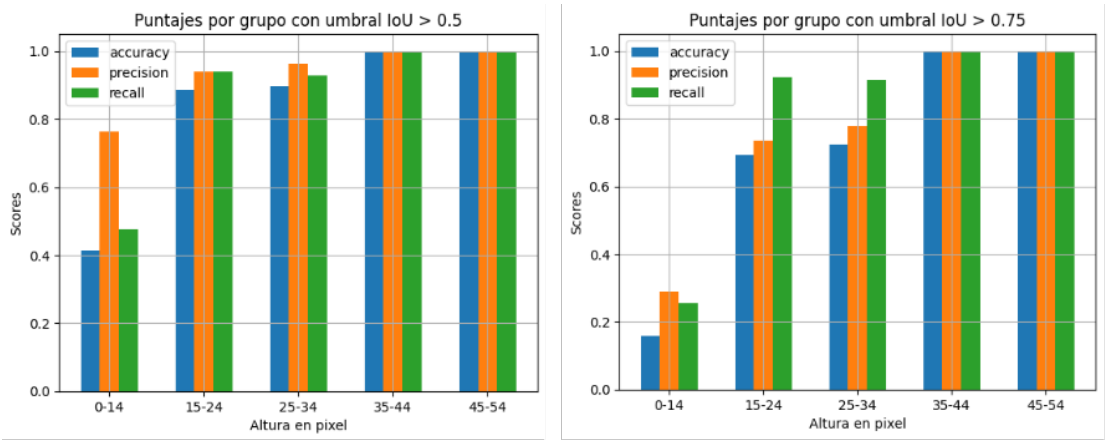


Figura 5.7: Puntajes por grupo para umbral de IoU mayor a 0.5 y 0.75.

detecciones correctas debido al bajo valor de IoU, mientras que al aplicar el umbral de 0.75 además de los grupos anteriores, el grupo de 25-34 también disminuye sus puntajes. Es importante destacar que para el grupo de patentes de tamaño 35-44 y 45-54 los puntajes se mantiene, ya que todas sus detecciones tienen un valor de IoU mayor a 0.75. Estos resultados son esperables, ya que al aumentar el puntaje mínimo de IoU necesario para ser aceptado como detección correcta, las detecciones menos precisas, es decir con menor IoU, son identificadas como patentes no detectadas, disminuyendo los índices de accuracy, precision y recall.

Por último, en cuanto al tiempo de detección, en promedio tardó 88 ms equivalente a poder detectar en tiempo real a 11.3 FPS y modificando el parámetro ’frames por detección’ se podría lograr la detección

a tiempo real para vídeos de 30 FPS. Los puntajes teniendo en cuenta todas las predicciones y patentes reales para los distintos umbrales se resumen en la tabla 5.1:

Umbral IoU	Accuracy	Precision	Recall
0	0.73	0.95	0.76
0.5	0.69	0.90	0.75
0.75	0.48	0.63	0.68

Tabla 5.1: Puntajes generales por umbral.

5.2 Pruebas de seguimiento

Para probar el rendimiento de la clase PlateTracker, se reunió un conjunto de 17 fragmentos de vídeo con una duración de 10 cuadros consecutivos de un vídeo a 30 FPS, es decir la duración total es de 0.33 segundos, en cada cuadro se etiquetó manualmente la posición de las patentes presentes para ser utilizada como posición real. En los fragmentos de vídeo hay un mínimo de 2 patentes y un máximo de 4 patentes siendo seguida simultáneamente, para poder observar como afecta al tiempo de actualización del seguidor a medida que existen más seguidores activos.

En la figura 5.8 se puede observar el seguimiento de tres patentes en uno de los fragmentos de vídeo, donde se evidencia el correcto seguimiento en cada cuadro, este resultado se puede estudiar de mejor manera observando el gráfico presente en la figura 5.9, considerando que el gráfico presenta fluctuaciones importantes, se puede observar una tendencia leve a disminuir el índice de IoU a medida que los cuadros avanzan, lo que quiere decir que el seguimiento empeora con el tiempo, debido a un cambio de escala no seguido o simplemente el seguidor no calcula la posicion correctamente, para evitar este problema se recomienda realizar una detección para actualizar el seguidor cada cierto tiempo con la posición y tamaño correctas, en el desarrollo de este proyecto se trabajó con 10 cuadros por detección.

Las mayor función de la clase de seguidor es poder calcular la nueva posición de la patente de una forma correcta, para luego extraerlas correctamente dentro de sus margenes, al realizar la prueba al conjunto de ejemplo de la imagen 5.8, se extrajo las 10 muestras de las 3 patentes seguidas, resultando las patentes presentadas en 5.10. Las patentes del grupo (a) corresponden a la curva azul, grupo (b) a la curva naranja y el grupo (c) a la curva verde del gráfico 5.9.

La cantidad de patentes a seguir en todo el conjunto de prueba fueron un total de 49 patentes, de las cuales se logró realizar el seguimiento de una forma correcta por los 10 cuadros a 41 de estas y 8 patentes se perdió el seguimiento en alguno de los cuadros, implicando que un 83.67% de las patentes detectadas son

Figura 5.8: Muestra de 3 cuadros de un fragmento de vídeo con posición real (Azul) y posición estimada (Verde)

correctamente seguidas al menos por un horizonte de 10 cuadros. Un ejemplo visual de un error se puede observar en 5.11, donde en un cuadro la posición estimada es muy lejana a la patente real y al siguiente cuadro el seguidor se borra por mal seguimiento.

En cuanto al tiempo de actualización de los seguidores, los resultados de la prueba indican que este tiempo crece linealmente con respecto a la cantidad seguidores activos y actualizados, esta tendencia se puede observar en el gráfico de 5.12 con una pendiente de aproximadamente 5 milisegundos por seguidor activo.

5.3 Pruebas de reconocimiento de caracteres

El conjunto de pruebas para verificar el funcionamiento de la clase CustomOCR se generó extrayendo las patentes detectadas de tres vídeos de prueba, con un total de 2066 imágenes de patentes y un total de 112 patentes únicas. Al igual que las pruebas de detección los datos se distribuyeron en grupos dependiendo de la altura de la patente.

Dentro del conjunto de pruebas existen tanto patentes que se pueden leer al ojo humano como también

Figura 5.9: Evolución del índice IoU por cuadro.

patentes difíciles de leer, algunos ejemplos se muestran en la figura 5.13, mientras que la distribución de patentes en los grupos segmentados por altura se muestra en la figura 5.14.

La prueba consistió en reconocer todas las patentes del conjunto de prueba con dos valores de 'Magnification ratio' (mr), el por defecto de 1 y un valor de 5 para probar cómo mejora el reconocedor. El reconocimiento se midió con tres indicadores: 'Accuracy', como cantidad de reconocimiento completo y correcto sobre la cantidad de imágenes; similitud, como la similitud de secuencia entre la patente reconocida y la real; y tiempo de reconocimiento, como el tiempo promedio que toma el reconocer cada patente. El resultado del indicador 'Accuracy' se evidencia en la figura 5.15.

En los gráficos de la figura 5.15 se puede observar el mal rendimiento del modelo entrenado, llegando a ser casi imperceptible con respecto a los resultados con el modelo por defecto de EasyOCR. Esto se debe a la imposibilidad de realizar correctamente el entrenamiento y completar las 300.000 épocas que recomienda el desarrollador de EasyOCR, esto podría mejorar aumentando las capacidades o pagando una membrecía para entrenar el modelo en la nube de forma más profesional.

En cuanto al rendimiento del modelo por defecto, se puede observar un gran aumento en su accuracy con un mr mayor, llegando a un 54% como el mejor índice para el grupo de 60-79 píxeles de alto, lo que quiere decir que más de la mitad de las patentes fueron reconocidas a plenitud y un 46% de las patentes

Figura 5.10: Patentes extraídas del vídeo de ejemplo.

fueron reconocidas parcialmente o no fueron reconocidas. Sin embargo el índice es bastante bajo, esto se puede ocasionar por la calidad de grabación de los vídeos utilizados, donde una cámara estática podría generar mejores resultados.

Otro resultado no esperado es que a las patentes con mayor altura disminuye la calidad del reconocimiento, esto se puede deber a que la altura también influya en el grado de inclinación de la patente, resultando un mayor grado a mayor altura.

La mejora del rendimiento del reconocedor tiene un efecto en lo rápido del reconocimiento, donde con mr = 1 tarda máximo 0.038 segundos aproximadamente y con mr = 5 tarda 0.22 segundos por patente, esto se debe a que el valor de mr indica que tanto mejoramiento de la imagen se realiza en el preprocesamiento de la imagen previo al reconocimiento.

El índice de similitud se calcula como la razón entre la cantidad de coincidencias en la secuencia de caracteres y la cantidad total de caracteres, siendo 1 en caso de que las cadenas de texto son iguales y 0 si no existe ninguna letra en común, con este indicador se puede verificar el reconocimiento parcial de las patentes. En la figura 5.16 se muestran los resultados, donde se observa una buena similitud para las patentes de por sobre 20 píxeles de alto, tanto para el reconocedor con mr = 1 y 5, llegando a un 72,8% de promedio para el caso de mr = 5 y patentes por sobre 20 píxeles de alto.

Por último, se buscó comprobar si el filtrado por recurrencia de lectura mejora el rendimiento y evita salidas incorrectas, los resultados se resumen en los gráficos de la figura 5.17. En cuanto al accuracy se disminuyó luego de filtrar las predicciones por recurrencia, esto se puede deber a que inicialmente las patentes más grandes o mejor grabadas se extrajeron más veces que las patentes de mala calidad y

Figura 5.11: Secuencias de cuadros con un error en el seguimiento. Ordenadas de superior a inferior.

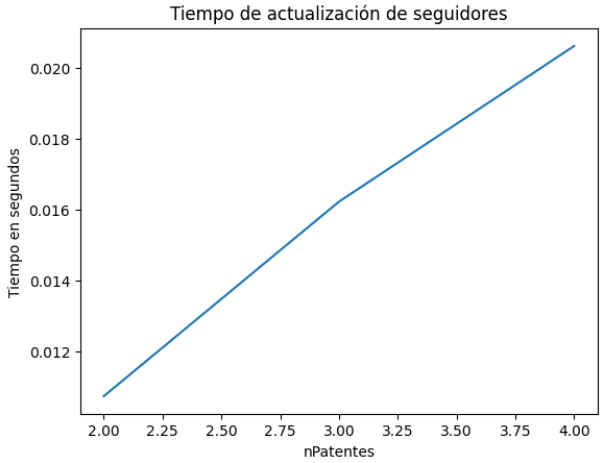


Figura 5.12: Relación entre tiempo de actualización del seguimiento y cantidad de seguidores.

al unificar estas predicciones disminuyen las predicciones correctas de una forma más acentuada que las patentes de baja calidad, de igual forma la similitud también disminuye debido a que este filtrado elimina las predicciones parciales y solo entrega predicciones totales.

Figura 5.13: Muestras del conjunto de pruebas. (a) Patentes legible y (b) Patentes difíciles de leer.

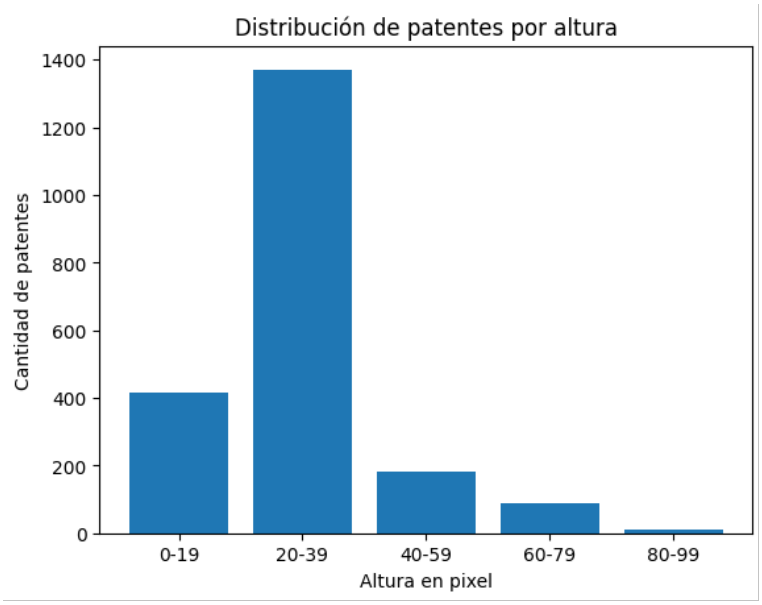


Figura 5.14: Distribución de patentes por altura.

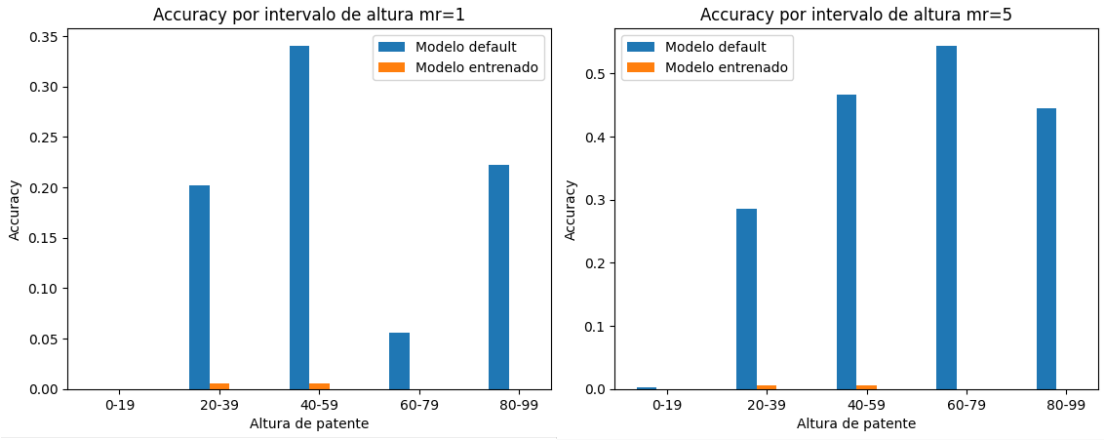


Figura 5.15: Accuracy con ’magnification ratio’ de 1 y 5.

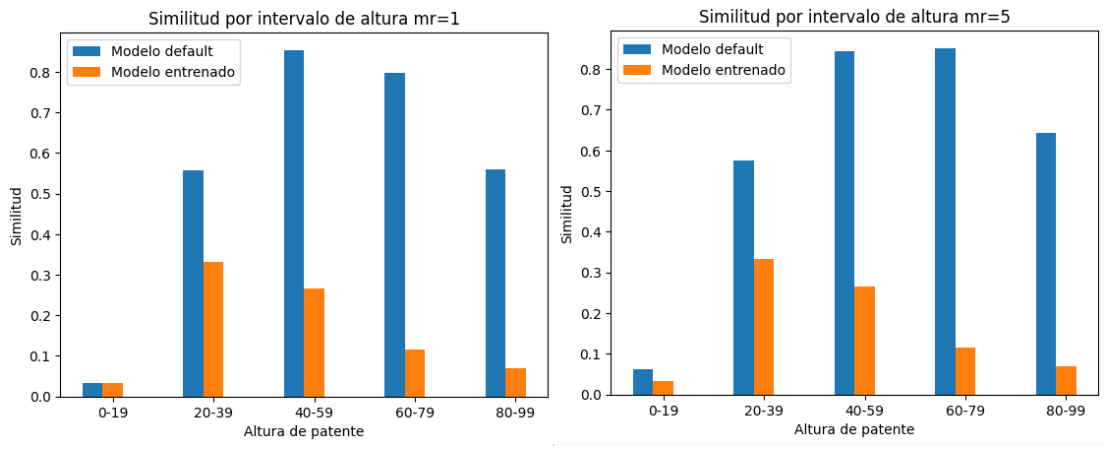


Figura 5.16: Similitud con mr = 1 y 5.

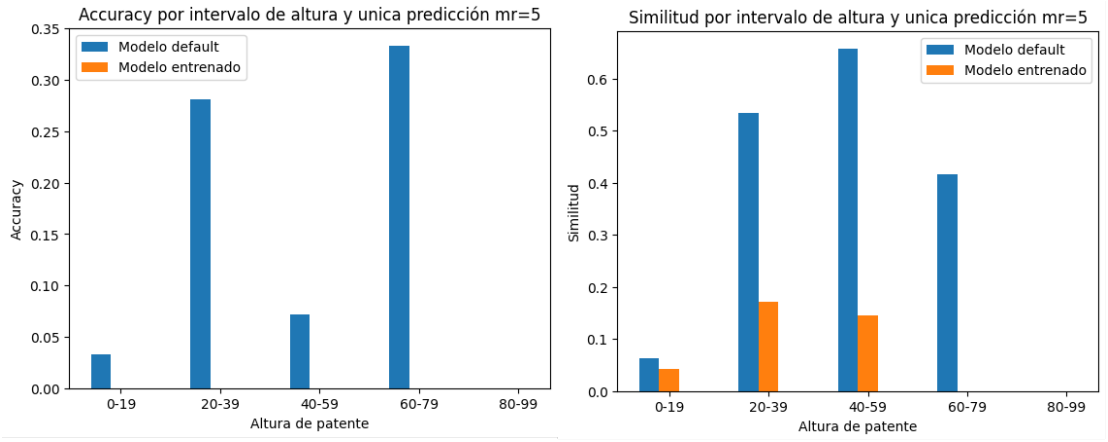


Figura 5.17: Accuracy y similitud con las predicciones filtrada por recurrencia.

5.4 Demostración del funcionamiento del sistema

A continuación se demuestra el funcionamiento del sistema completo mediante un simulacro de uso, al ejecutar el comando ’python3 Client.py’ se abre la ventana de la figura 5.18:

Al apretar el boton ’Analyze video’ se abre un explorador de archivos con la dirección donde se ejecutó el comando como dirección inicial, un ejemplo del explorador de archivos se muestra en la figura 5.19. Una vez ingresado el vídeo, se inicializa el algoritmo de reconocimiento y se muestra en una ventana extra el vídeo analizado con el texto extraído e índice de confianza sobre la patente, como muestra la figura 5.20.

Una vez que alguna patente se salga de la escena o se deja de detectar por 2 veces el parámetro de ’Frames por detección’ se extrae y muestra el reconocimiento realizado, como se puede observar en la figura 5.22.

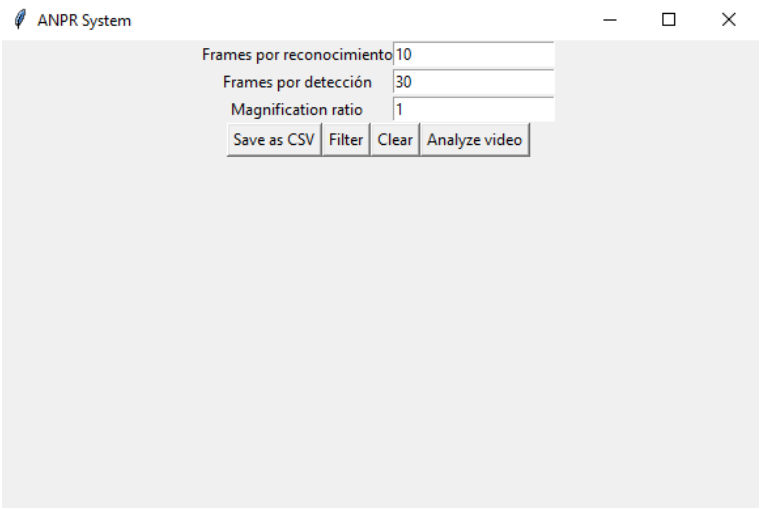


Figura 5.18: Interfaz gráfica inicial.

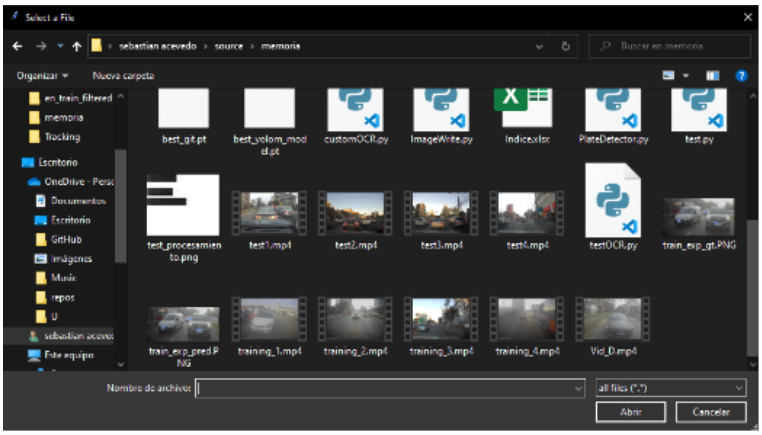


Figura 5.19: Explorador de archivos para ingreso de vídeo.

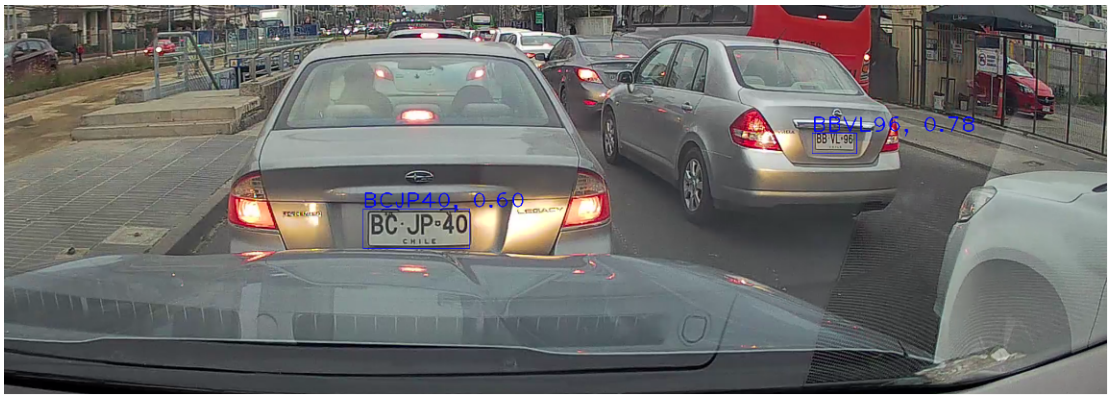


Figura 5.20: Ventana de muestra del vídeo analizado a tiempo real.

Finalmente si el usuario así lo desea, puede guardar los resultados en un archivo .csv, para el ejemplo el archivo generado se muestra en la figura 5.23.

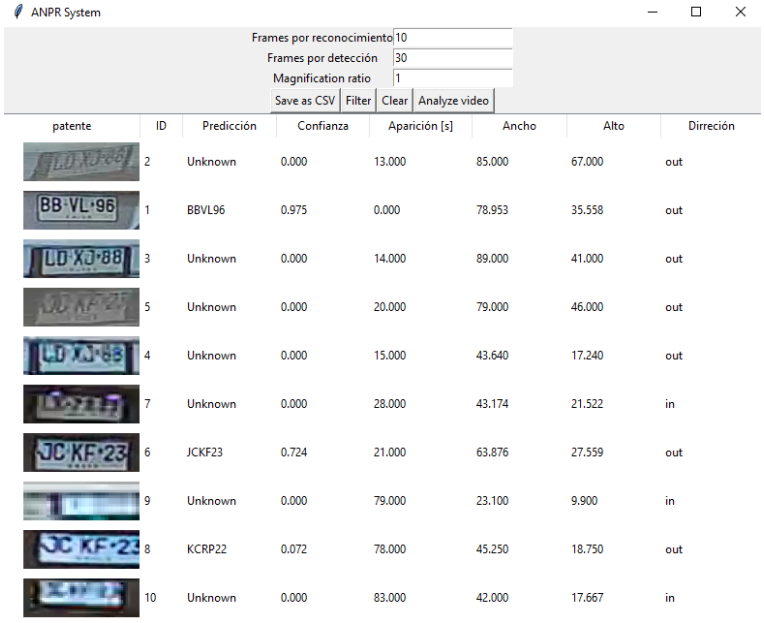


Figura 5.21: Interfaz gráfica con resultados de reconocimiento de patentes.

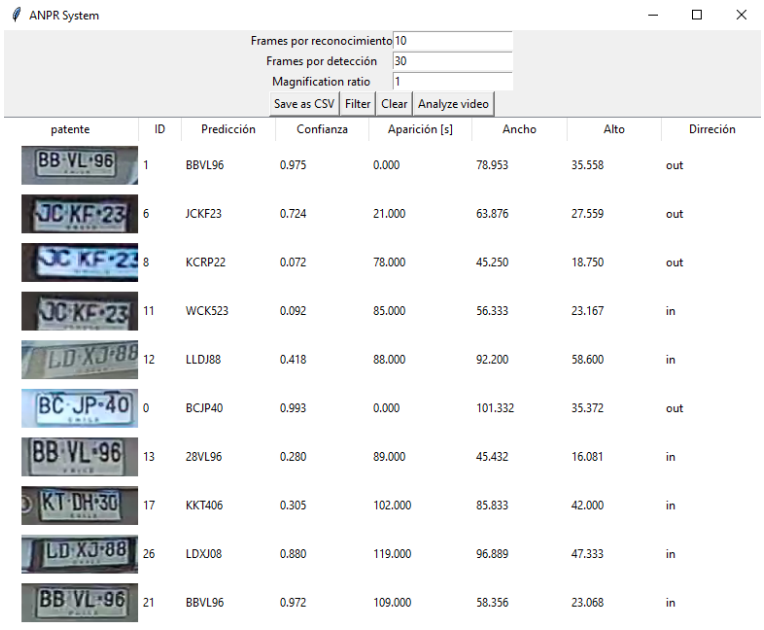
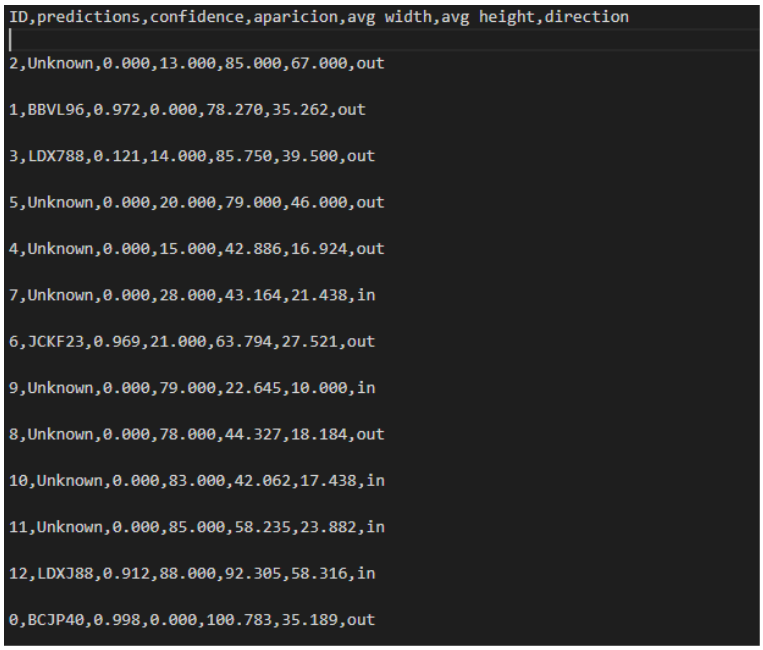


Figura 5.22: Interfaz gráfica con resultados de reconocimiento de patentes filtrados.



```
ID,predictions,confidence,aparicion,avg width,avg height,direction
2,Unknown,0.000,13.000,85.000,67.000,out
1,BBVL96,0.972,0.000,78.270,35.262,out
3,LDX788,0.121,14.000,85.750,39.500,out
5,Unknown,0.000,20.000,79.000,46.000,out
4,Unknown,0.000,15.000,42.886,16.924,out
7,Unknown,0.000,28.000,43.164,21.438,in
6,JCKF23,0.969,21.000,63.794,27.521,out
9,Unknown,0.000,79.000,22.645,10.000,in
8,Unknown,0.000,78.000,44.327,18.184,out
10,Unknown,0.000,83.000,42.062,17.438,in
11,Unknown,0.000,85.000,58.235,23.882,in
12,LDXJ88,0.912,88.000,92.305,58.316,in
0,BCJP40,0.998,0.000,100.783,35.189,out
```

Figura 5.23: Contenido de archivo .csv con resultados de ejemplo.

6 Conclusión

Tras la investigación y desarrollo de este proyecto, se logró implementar correctamente los algoritmos de detección, seguimiento y reconocimiento de patentes, donde los problemas más significativos resultaron en el reconocimiento de caracteres debido a la complejidad de este proceso sobre todo con caracteres de tamaño menor.

A raíz de este proyecto, se pueden generar nuevas oportunidades en el ámbito del reconocimiento de patentes en Chile, ampliando las formas en que se enfrenta este problema, demostrando que se puede desarrollar un sistema que logre resultados aceptables, apoyándose en las nuevas tecnologías basadas en la inteligencia artificial sin una necesidad de un sistema embebido especializado en la operación de detección y reconocimiento de patentes.

En cuanto a la detección, este proyecto logró cumplir con los objetivos planteados al iniciar este proceso, cumpliendo con el entrenamiento de un modelo de detector de objetos basado en el algoritmo YOLO, llegando a detectar al menos un 90% de las patentes presentes en el vídeo con un rapidez destacable e incluso para estándares más precisos se mantuvo sobre el 85% de patentes detectadas sobre 14 píxeles de alto con una IoU mayor a 0.75. Este modelo es la mayor contribución del proyecto por su gran precisión y rendimiento.

Análogamente, la clase implementada para realizar el seguimiento de patentes también cumplió con los objetivos planteados, asignando un número identificador único por patente y alcanzando un 83,67% de posiciones estimadas en un horizonte de 10 cuadros sin perder información de la patente, permitiendo que la detección se pueda realizar cada un número de cuadros y así acelerar el proceso de extraer patentes para ser reconocidas.

El trabajo a futuro para mejorar este proyecto es optimizar el modelo de reconocimiento óptico de caracteres, él resultó ser el proceso con peor resultados del sistema completo, debido a la imposibilidad de entrenar un modelo particular con las fuentes de las patentes de Chile, sin embargo ajustando parámetros como el ’Maginification ratio’ se logró mejorar el resultado desde un máximo de 34% a 54% de precisión en la lectura completa para patentes con calidad aceptable en cuanto a tamaño y claridad de la imagen.

Por último, el objetivo de desarrollar una interfaz de usuario simple se cumplió a plenitud, disponiendo de una interfaz que permite al usuario ingresar el vídeo a analizar de una forma fácil, además de permitir que el usuario modifique parámetros importantes para el rendimiento del sistema en función a su necesidad

en cuanto a velocidad de procesamiento o precisión del reconocimiento.

Referencias

[1] Gaudenz Boesch. Object detection in 2022: The definitive guide [en línea]. <https://viso.ai/deep-learning/object-detection/: :text=on[consulta: 1 Septiembre 2022].

[2] Jairo Rojas Campo. Qué es lpr, cómo funciona y sus aplicaciones [en línea]. <https://www.tecnoseguro.com/faqs/que-es-lpr-como-funciona-y-sus-aplicaciones>, [consulta: 1 Septiembre 2022].

[3] Curiocity. Aumento del parque automotriz en chile desafía a mejorar infraestructura vial y transporte público [en línea]. <https://www.curiocity.cl/noticias/aumento-del-parque-automotriz-en-chile-desafia-a-mejorar-infraestructura-vial-y-transporte-publico/>, [consulta: 1 Septiembre 2022].

[4] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. Bmva Press, 2014.

[5] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[6] JaidedAI. Repositorio git easyocr[en línea]. <https://github.com/JaidedAI/EasyOCR>, [consulta: 1 Marzo 2020].

[7] Chirag Patel, Dipti Shah, and Atul Patel. Automatic number plate recognition system (anpr): A survey. *International Journal of Computer Applications*, 69(9), 2013.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[10] Rasmus Rothe, Matthieu Guillaumin, and Luc Van Gool. Non-maximum suppression for object detection by passing messages between windows. In *Asian conference on computer vision*, pages 290–306. Springer, 2014.