UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO DE UN SISTEMA INTELIGENTE PARA LA DETECCIÓN DE PLAZAS DE ESTACIONAMIENTOS LIBRES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA

CAROLINA DEL CARMEN GARCÍA GUERRERO

PROFESOR GUÍA:
ANDRÉS CABA RUTTE

MIEMBROS DE LA COMISIÓN:
FRANCISCO RIVERA SERRANO
FRANCISCO CASADO CASTRO

SANTIAGO DE CHILE
2022

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: CAROLINA DEL CARMEN GARCÍA GUERRERO
FECHA: 2022
PROF. GUÍA: ANDRÉS CABA RUTTE

DISEÑO DE UN SISTEMA INTELIGENTE PARA LA DETECCIÓN DE PLAZAS DE ESTACIONAMIENTOS LIBRES

Actualmente, la evolución del parque automotriz ha ido en aumento, dada la gran demanda que existe por la adquisición de nuevos vehículos, hecho que no solo ocurre en nuestro país, sino que a nivel el mundial. Lo anterior ha desencadenado un sin número de efectos negativos, entre los que destacan el aumento de la contaminación, la creciente congestión vehicular, la dificultad constante a la hora de buscar un estacionamiento, entre otros. Efectos que se intensifican ante la realización de eventos masivos, como partidos de fútbol, conciertos, visita a centros comerciales, y especialmente en horario punta, cuando ocurre la entrada o término de la jornada laboral.

Dado lo anterior, es que este trabajo busca enfocarse en proponer una solución eficiente que permita resolver el problema de la búsqueda de una plaza vacía para estacionar, disminuyendo así el tiempo invertido por el conductor para lograr estacionarse.

Para lograr dar solución al problema planteado, se diseñó un sistema inteligente, a partir del entrenamiento de una red neuronal convolucional, que permite detectar mediante la captura de imágenes de un estacionamiento, las plazas que se encuentras vacías y ocupadas, además de indicar el número total de vacantes libres.

Por otro lado, para el entrenamiento de la red se construyó una base de datos con imágenes tomadas a un estacionamiento residencial, ubicado en un condominio en la comuna de Algarrobo. Cabe destacar que el gran desafío que presenta este estacionamiento, es no poseer un separador distintivo, a diferencia de estacionamientos de este tipo que presentan delimitadores marcados en color blanco o amarillo.

Finalmente, a partir de la base de datos creada, se realizan 3 experimentos para evaluar el desempaño del modelo, basándose principalmente en utilizar diferentes condiciones de luz ambiental para los conjuntos de entrenamiento, validación y test. Dado lo anterior, es que se logran detectar el estado de 24 plazas, al aplicar el algoritmo sobre el estacionamiento de estudio, obteniéndose un mAP sobre el 90 % en casa uno de los experimentos realizados.

Si lo intentas puedes perder, si no lo intentas ya has perdido.

Agradecimientos

Quiero agradecer, en primer lugar, a Dios por ser mi principal soporte y acompañarme en este camino. Porque de él, y por él, y para él, son todas las cosas. A él sea la gloria por los siglos. Amén (Romanos 11:36).

Agradezco a mi profesor Guía Andrés Caba por su compromiso y constante apoyo en este trabajo, así como también en toda mi estadía en el departamento de Ingeniería eléctrica. Así como también, a mi profesor co-guía Francisco Rivera, quien fue la fuente de inspiración para la realización de este trabajo. Finalmente agradecer a Francisco Casado por brindar su apoyo y consejos.

Agradezco a mi familia, en especial a mi mamá por ser mi soporte todos estos años, e impulsarme a seguir adelante y luchar en la adversidad. A mi papá y mis hermanos Cheska, Luis, Luisana y Alejandro por estar siempre a mi lado. A mi novio Nicolás Carvajal, por ser mi compañero y brindarme su apoyo incondicional y comprensión a lo largo de mi estadía en la universidad.

A mis tíos maternos, Mariana y Víctor, por darme ánimos y ser mis compañeros de viaje. A mis tías paternas, Verónica y Kaki, por compartir su hogar y cuidarme en mis primeros años de universidad.

A mis amigas Selma y Maka quienes fueron mis primeras compañeras en plan común, y aunque tomamos carreras separadas, su amistad sigue perdurando.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación y Antecedentes

En la actualidad, el aumento de la congestión vehicular en las zonas urbanas es un tema que afecta cada vez más a los conductores, no solo en nuestro país, sino que alrededor de todo el mundo. Por un lado, se encuentra el tráfico constante que se genera en las grandes ciudades, sumándole a ello la dificultad para encontrar estacionamientos, sobre todo en horario punta, cuando se ingresa o se termina la jornada laboral. Por otro lado, se encuentran los eventos masivos a los que se les atribuye gran pérdida de tiempo a la hora de estacionar, como cuando se asiste a un centro de entretención, centro comercial o Mall, partidos de fútbol, etc.

Respecto a la situación recurrente de la congestión vehicular producto de la búsqueda de estacionamiento, surge la pregunta ¿cuánto tarda un automovilista en encontrar un estacionamiento? o ¿qué problemas puede ocasionar a la salud? A nivel internacional, se han realizado algunas investigaciones, enfocándose en las grandes ciudades del mundo. Los resultados no son muy alentadores respecto a este tema, ya que según un estudio realizado por IBM [1], por medio de una encuesta a 8.192 conductores de 20 ciudades de cinco continentes, se expone la problemática reflejada en la pérdida de tiempo del conductor, con un tiempo promedio de 20 minutos para encontrar una plaza vacía. Además, se informó que el problema afecta un 30 % del congestionamiento urbano [2], sumado a la gran cantidad de gases contaminantes emitidos mientras se realiza la búsqueda, añadiendo el impacto en la salud, lo cual provoca un alto nivel de estrés en el conductor.

Por otro lado, a nivel nacional, no existen estudios específicos relacionados a esta problemática, sin embargo, es posible basarse en los datos estadísticos del parque automotriz [3] para notar que la congestión vehicular es un problema. Según estos datos, representados en la figura 1.1, donde se muestra la evolución del parque de vehículos motorizados en los últimos años, es posible notar que son cada vez más las personas que adquieren un automóvil. Además, dado los datos mostrados en la figura 1.2, se debe tener en cuenta que el flujo vehicular se encuentra mayormente concentrado en la Región Metropolitana, donde de los 5,7 millones de vehículos que circulan en Chile, un poco más de 2 millones lo hacen solo en la capital (38,94 %), sumado al hecho de que la congestión solo se concentra en puntos estratégicos de ciertas comunas. Así, los resultados revelan que quienes sufren mayor congestión

corresponden a las comunas de Santiago, Peñalolen, Las Condes, Maipú y Vitacura, tal como se muestra en la figura 1.3 [3].

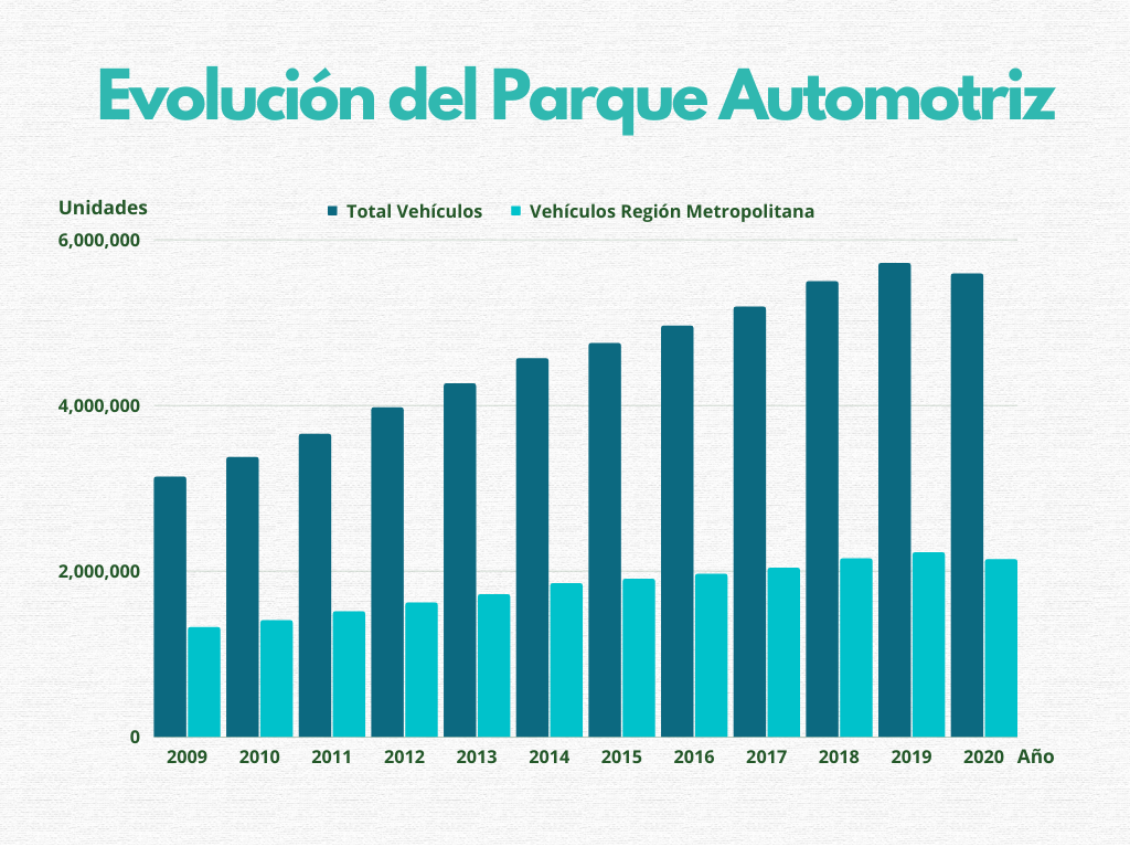


Figura 1.1: Evolución del Parque Automotriz.
Elaboración propia a partir de los datos del parque automotriz [3].

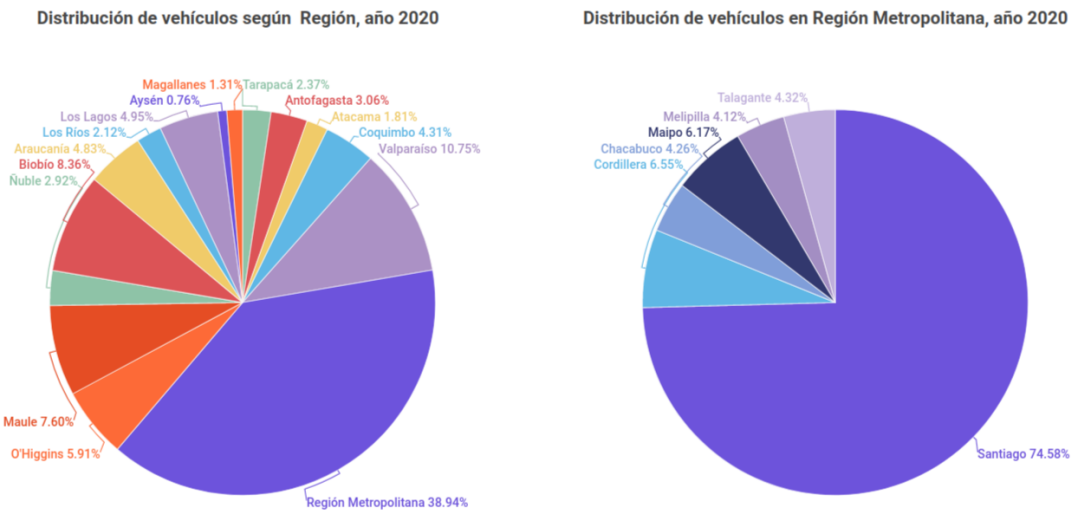


Figura 1.2: Distribución de vehículos según región, año 2020.
Elaboración propia a partir de los datos del parque automotriz [3].

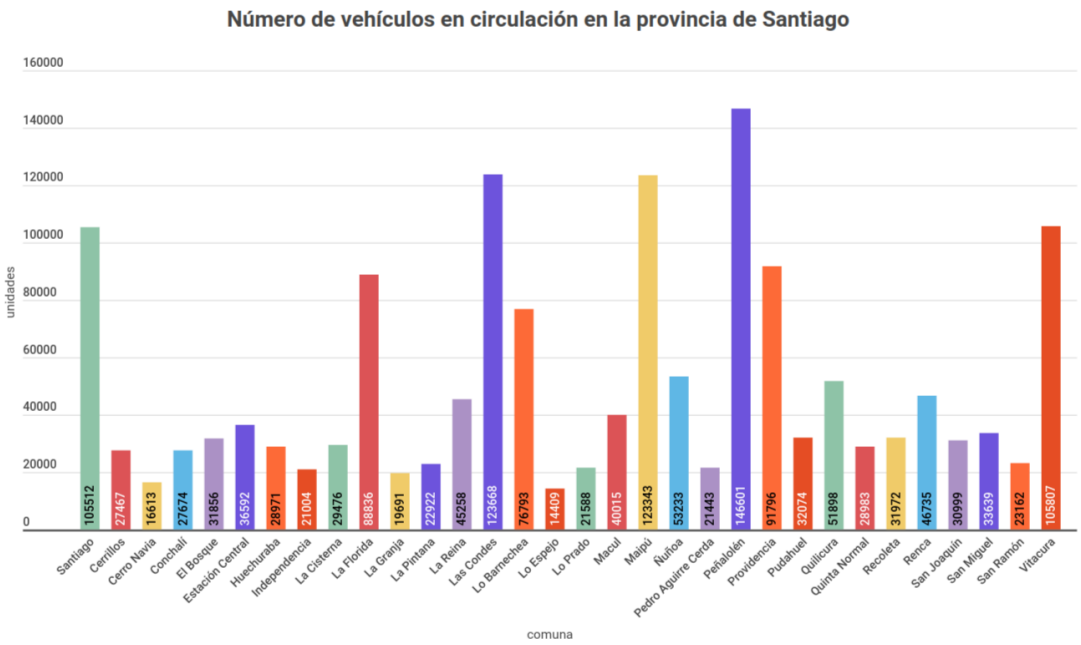


Figura 1.3: Distribución de vehículos según comuna de la Región Metropolitana, año 2020. Elaboración propia a partir de los datos del parque automotriz [3].

1.2. Descripción del problema y propuesta de la solución

Dado los problemas planteados en el apartado anterior, se propone diseñar un sistema inteligente de visión computacional, que permita determinar las plazas vacías y ocupadas en un determinado estacionamiento, además de mostrar al usuario en el estado de cada una de ellas.

El primer paso para encontrar una solución, consiste en identificar cuáles son las variables relevantes del sistema. En este sentido se debe considerar que el diseño propuesto debe ser robusto a las condiciones ambientales donde se instalará el sistema y, además, se debe considerar las diferentes topologías de los estacionamientos.

Las topologías de los estacionamientos se pueden dividir en 2 grandes grupos:

- *Outdoors*: Los estacionamientos del tipo *Outdoors* corresponde a aquellos donde los lugares de aparcamiento se encuentran al aire libre, expuestos directamente a las condiciones ambientales. Estos pueden dividirse en dos subgrupos: playa y calle. Los primeros se caracterizan por formar parte, en la mayoría de los casos, de un recinto privado, presentando casi siempre delimitadores específicos para cada plaza y en algunas ocasiones pueden contar con un techo que cumple la función de proteger los autos, ya sea del sol o la lluvia. Los segundos tipo "calle", se caracterizan por estar ubicados en las vías públicas, sin delimitaciones específicas, en los costados de las vías de tránsito vehicular. Además, pueden ser o no pagados y en definitiva, dado el lugar donde se ubican, son los que se encuentran en mayor cantidad y generan más congestión vehicular.
- *Indoors*: Los estacionamientos de tipo *Indoors*, corresponden a aquellos que presentan

una estructura física donde pueden ingresar los autos para estacionar. Estos se caracterizan por ser privados y tener plazas delimitadas para cada automóvil. Se dividen principalmente en 2 subgrupos: Subterráneo y Edificios. Los primeros son aquellos cuyas plazas de estacionamientos se encuentran en niveles inferiores, generalmente varias plantas. Los tipos edificio, son similares a lo anterior, solo que en vez de plantas subterráneas las plantas se encuentran en niveles superiores. También se da el caso de que se presente una combinación de los tipos subterráneos con los tipos edificio.

Un resumen de lo expuesto anteriormente, con su respectivo ejemplo gráfico, se puede observar en la figura 1.4.

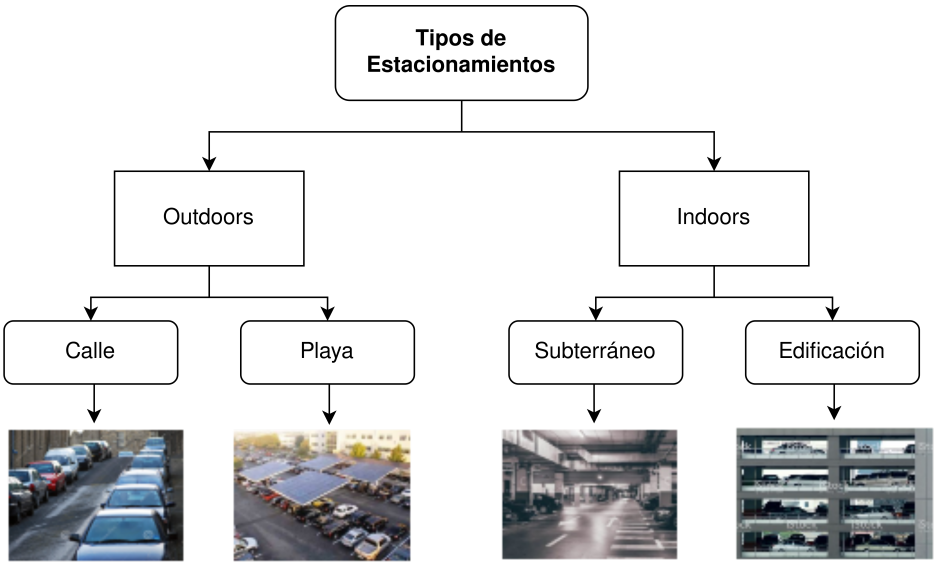


Figura 1.4: Tipos de Estacionamientos.
Elaboración Propia.

Dado las diferentes topologías descritas, es que el algoritmo solo será implementado para una de ellas, específicamente, un estacionamiento *"Outdoor"* tipo *playa*. Lo anterior es debido a que el estacionamiento disponible para realizar este estudio corresponde a uno de este tipo.

Por lo tanto, a partir de este estacionamiento de estudio, el sistema propuesto funcionará como sigue: en primer lugar, se instalará una cámara que sea capaz de monitorear cada una de las plazas de estacionamientos. Las imágenes capturadas por la cámara serán ingresadas como entrada a un algoritmo previamente entrenado, que detectará cada plaza de estacionamiento y evaluará si ésta se encuentra libre u ocupada. Posteriormente, comunicará esta información a una interfaz mapeada con las plazas de estacionamiento, indicando el estado actual de cada una de ellas.

Un ejemplo ilustrativo de la propuesta del sistema se puede observar en la figura 1.5.

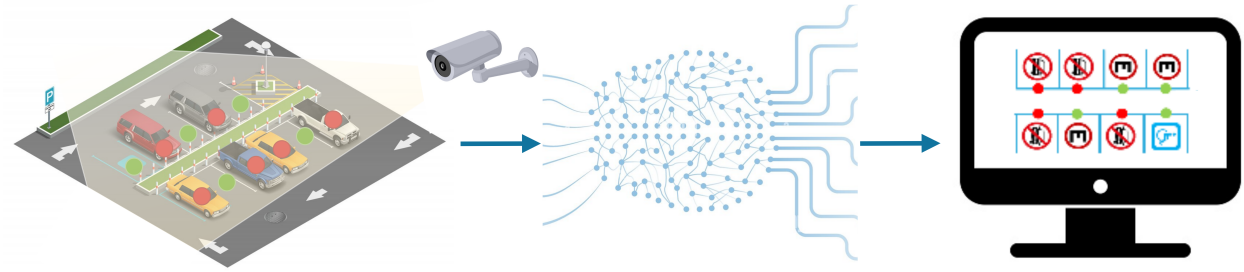


Figura 1.5: Ejemplo Ilustrativo.
Elaboración propia.

1.3. Objetivos

La sección de objetivos se puede dividir en dos: objetivo generales y específicos, entre los que destacan:

- **Objetivo General:**
  Implementar un sistema inteligente, mediante el uso de algoritmos de visión computacional, para determinar el mapeo de las plazas de un estacionamiento *Outdoor*, así como el estado en que se encuentra cada una de ellas, ya sea libre o ocupada.
- **Objetivos Específicos:**
  1. Construcción de una base de datos con imágenes de plazas de estacionamientos, con muestras de plazas vacías y ocupadas.
  2. Implementación de un algoritmo de visión artificial, para la detección de plazas disponibles /ocupadas.
  3. Diseño de una interfaz gráfica para computador, que muestre al usuario el mapeo de las plazas de estacionamiento detectadas, así como el estado de cada una de ellas.

Dado lo anterior, este informe presentará en 6 capítulos el trabajo realizado, iniciando en el capítulo 1 con la introducción del problema y los antecedentes. Luego en el capítulo 2 se detallará la base teórica necesaria para resolver el problema. En el capítulo 3 se mostrará el estado del arte, con investigaciones similares hasta la actualidad. Posteriormente, en el capítulo 4 se explicará la metodología seguida y en el capítulo 5 los resultados obtenidos, alcanzándose un mAP sobre el 90 % en cada experimento realizado. Finalmente, en el capítulo 6 se presentarán las conclusiones y el trabajo futuro.

Capítulo 2

Marco Teórico

2.1. Red Neuronal

Una red neuronal es un modelo computacional que se inspira en el funcionamiento de la corteza cerebral de los mamíferos. Es decir, los modelos de redes neuronales artificiales pueden entenderse como un conjunto de unidades de procesamiento básicas, que están estrechamente interconectadas y operan en las entradas dadas para procesar la información y generar los resultados deseados [4].

El perceptrón, la red neuronal básica, es un clasificador binario de modelo lineal con una relación entrada-salida simple como se muestra en la Figura 2.1. Se puede observar que se están sumando n números de entradas multiplicadas por sus pesos asociados y luego enviando esta "entrada neta" a una función escalonada, con un umbral definido. Típicamente, la función de activación, es una función escalonada de Heaviside con un valor umbral de 0.5. Esta función generará un valor binario único de valor real (0 o 1), según la entrada.

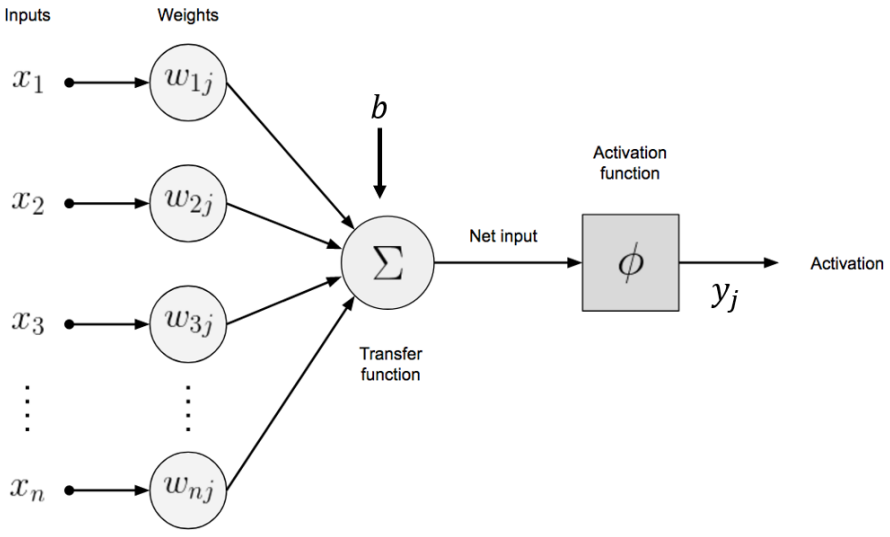


Figura 2.1: Neurona artificial para un perceptrón [5].

Dada la figura 2.1, se tiene entonces que la salida a la capa del perceptrón vendrá dada por:

$$y = \phi(\sum_{\mathrm{i}=1}^{n} w_{\mathrm{i}j} x_{\mathrm{i}} + b) \tag{2.1}$$

Donde:

- y_j = Salida de la red en la iteración j.
- $w_{\mathrm{i}j}$ = Peso asociado a la entrada i, en la iteración j.
- x_{i} = Entrada i.
- b = bias
- ϕ = Función de Activación.

El perceptrón solo es utilizado para resolver problemas lineales, sin embargo, en la práctica lo que se necesita es resolver problemas más complejos. Dado lo anterior, para lograr ello se utilizan redes neuronales multicapa, las cuales a diferencia del perceptrón, consta de más capas ocultas. En la figura 2.2 se puede apreciar un ejemplo de red neuronal multicapa, la cual consta de dos capas ocultas, intercaladas entre una entrada y una capa de salida.

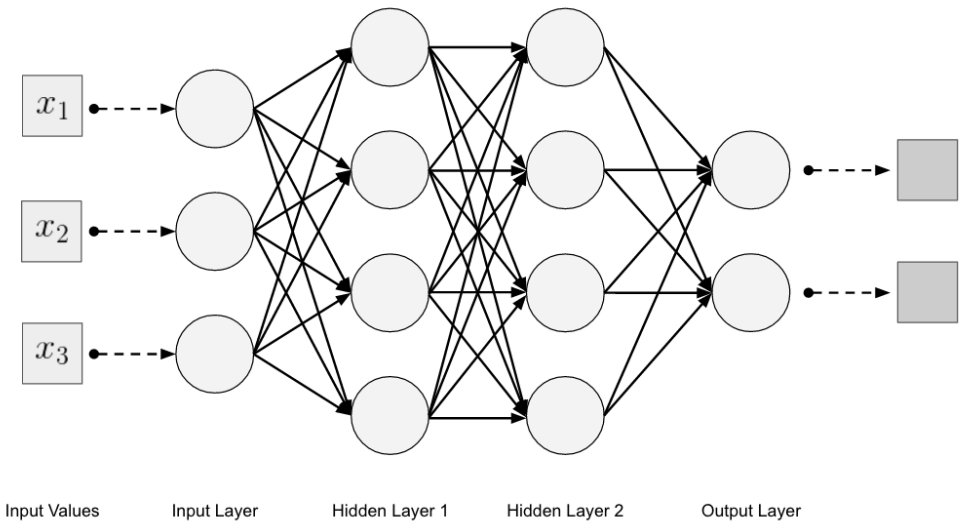


Figura 2.2: Topología de una Red Neuronal Multicapa [5].

A continuación se describe cada una de las capas:

- *Input Layer* o Capa de Entrada:

  Esta capa es la forma en que obtenemos datos de entrada (vectores) alimentados a nuestra red. El número de neuronas en una capa de entrada suele ser el mismo número que el de la función de entrada a la red. Las capas de entrada van seguidas de una o más capas ocultas [5].

- *Hidden layer* o Capa Oculta:

  Corresponde a las capas intermedias, que se encuentran conectadas entre la capa de entrada y la capa de salida. Los valores de peso en las conexiones entre las capas son cómo las redes neuronales codifican la información aprendida extraída de los datos de entrenamiento sin procesar. Las capas ocultas son la clave para permitir que las redes neuronales modelen funciones no lineales[5].

- *Output layer* o Capa de Salida:

  La capa de salida corresponde a la respuesta o predicción de nuestro modelo, basada en los datos ingresados a la capa de entrada. Dependiendo de la configuración de la red neuronal, la salida final puede ser una salida de valor real (regresión) o un conjunto de probabilidades (clasificación). Esto está controlado por el tipo de función de activación que usamos en las neuronas en la capa de salida. La capa de salida normalmente utiliza una función de activación sigmoidea o softmax para la clasificación [5].

2.2. Red Neuronal Convolucional(CNN)

Otro tipo de red neuronal corresponde a las denominadas CNN, las cuales son utilizadas especialmente para datos de alta dimensión (por ejemplo, imágenes y videos). Las CNN funcionan de una manera muy similar a las redes neuronales estándar. Sin embargo, una diferencia clave es que cada unidad en una capa CNN es un filtro bidimensional (o de alta) dimensión que se convoluciona con la entrada de esa capa. Esto es esencial para los casos en los que se requiere aprender patrones donde la entrada es de alta dimensión. Los filtros de CNN incorporan el contexto espacial al tener una forma espacial similar (pero más pequeña) a la entrada y reduciendo, significativamente, la cantidad de variables aptas para aprender.

La CNN aprende a asignar una imagen determinada a su categoría correspondiente al detectar una serie de representaciones de características abstractas, que van desde las más simples hasta las más complejas. Estas características discriminatorias se utilizan luego dentro de la red para predecir la categoría correcta de una imagen de entrada. El clasificador de red neuronal es similar al de una red neuronal multicapa, con la diferencia que la extracción de características, que en una red neuronal multicapa se hace de forma manual, en una CNN las capas convolucionales previas permiten realizar esta operación de forma automática, sin la necesidad de un experto [4].

2.2.1. Capa Convolucional

Corresponde a la capa donde se realiza la convolución de la imagen de entrada con un conjunto de filtros o kernels. Una convolución se define como una operación matemática que describe una regla sobre cómo fusionar dos conjuntos de información.

La operación de convolución, que se muestra en la Figura 2.3, se conoce como el detector de características de una CNN. La entrada a una convolución pueden ser datos sin procesar o una salida de mapa de características de otra convolución. A menudo se interpreta como un filtro en el que el kernel filtra los datos de entrada para determinados tipos de información [5].

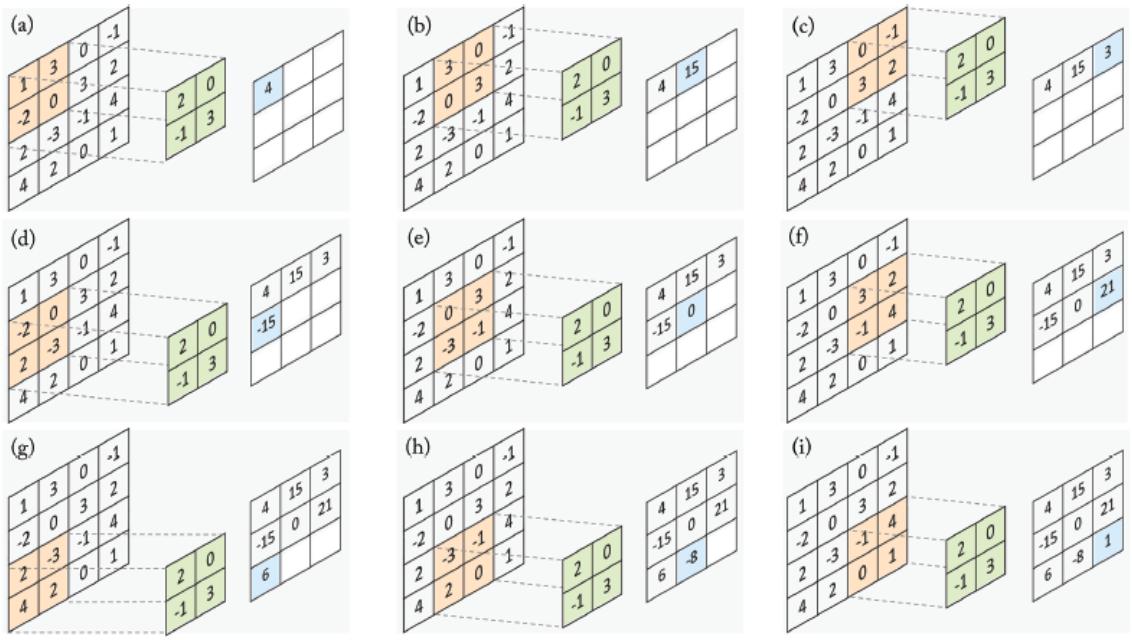


Figura 2.3: Ejemplo de la operación Convolución [4].

En la figura 2.3 se puede apreciar cómo, en cada paso, el kernel se multiplica por los valores de los datos de entrada, creando una sola entrada en el mapa de características de salida.

2.2.2. *Max-Pooling*

Corresponde a la operación que se realiza posterior a la capa de convolución. Este procedimiento usualmente se utiliza para reducir información espacial, consiguiendo reducir el sobreajuste (*overfitting*) en la red.

El procedimiento consiste en mover una ventana a lo largo de la imagen, aunque esta vez, a diferencia de la convolución, se escogerá el valor máximo para el caso de max-pooling de dicha ventana, tal como se muestra en la figura 2.4. Por otro lado, también es posible implementar otras operaciones en vez del máximo, como el *average pooling* donde se escoge el promedio.

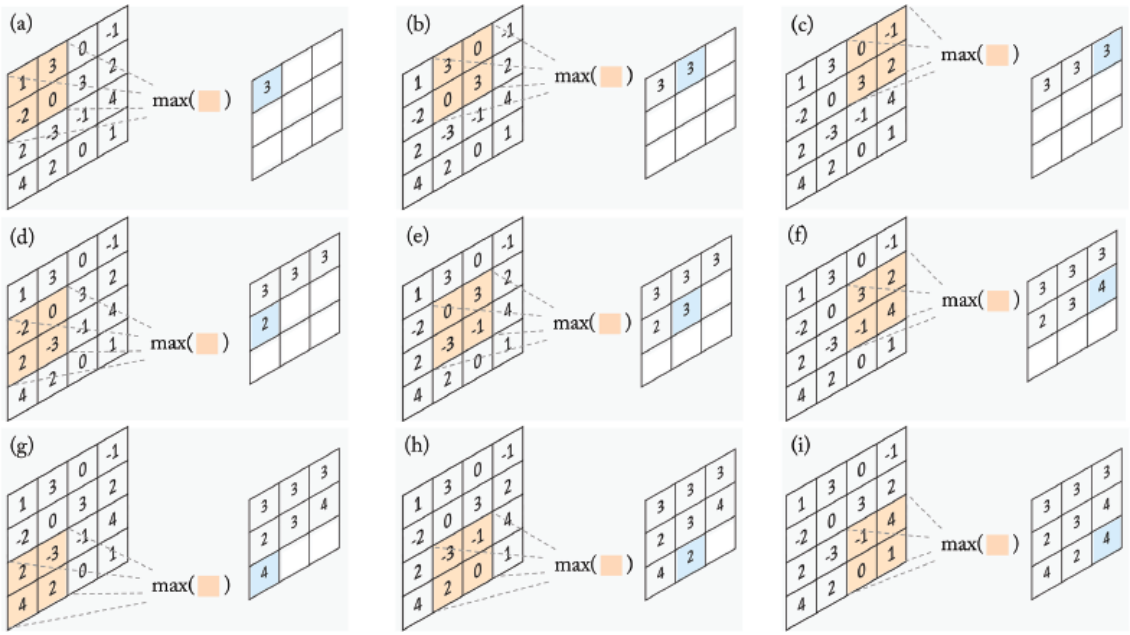


Figura 2.4: Ejemplo de operación Max-Pooling [4].

2.2.3. Capa *Fully Connected*

Corresponde a la capa que se suele utilizar al final de las redes convolucionales profundas. Por lo cual, cada píxel de la capa anterior se considera como una neurona separada al igual que en una red neuronal regular. La función principal de esta capa radica en determinar a que clase pertenece la imagen de entrada, por lo tanto, esta capa estará compuesta por tantas neuronas como número de clases existan.

Por otro lado, cabe destacar que la salida de esta capa entrega una probabilidad de pertenecer a cierta clase, por lo que, finalmente, el usuario determina que la clase entregada será la que entregue la mayor probabilidad de pertenencia [4].

2.2.4. Función de Activación

Una función de activación no lineal puede entenderse como un mecanismo de conmutación o selección, que decide si una neurona se disparará o no, dadas todas sus entradas

Tanto las capas convolucionales, como las capas "Fully Connected" a menudo van seguidas de una función de activación no lineal. Esta función de activación toma una entrada de valor real y la acota dentro de un rango pequeño como [0, 1] y [–1, 1]. La aplicación de una función no lineal después de las capas es muy importante, ya que permite que una red neuronal aprenda mapeos no lineales [4].

Las funciones de activación que se utilizan comúnmente en *"Deep Learning"* se pueden apreciar en la figura 2.5.

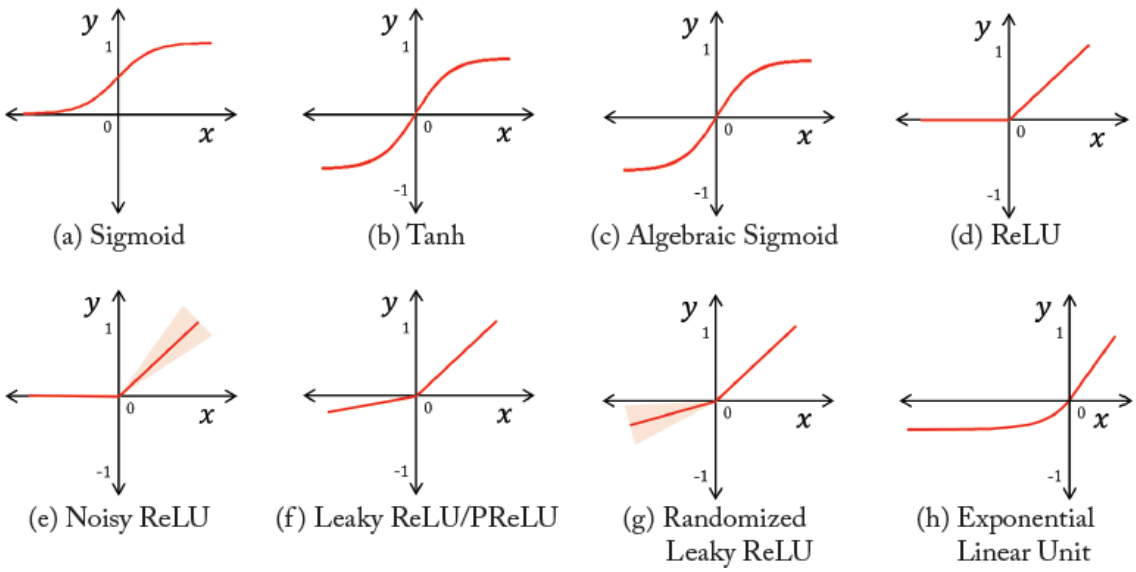


Figura 2.5: Ejemplo de algunas Funciones de Activación utilizadas en Redes Neuronales [4].

2.2.5. *Dropout*

Es una técnica de regularización para evitar el sobreajuste (*overfitting*) de la red neuronal. La técnica consiste en ignorar voluntariamente de manera aleatoria algunas neuronas durante el entrenamiento, es decir, se eliminan algunas conexiones ocultas de la red con una probabilidad configurable (normalmente 0.5)[4].

2.3. Detección de Objetos

Un detector, a diferencia de un clasificador que se encarga solo de predecir la clase de un objeto en una imagen, cumple la función de identificar la localización de uno o varios objetos dentro de una imagen, así como definir el espacio que ocupa dentro de ella, enmarcando un cuadro alrededor de su extensión.

Una representación gráfica del funcionamiento del detector se puede apreciar en la figura 2.6, la cual se divide en tres etapas principales [6]:

- ***Backbone*:** corresponde a la arquitectura base del modelo que actúa como extractor de características, y suelen ser, básicamente, arquitecturas de clasificación, como *VGG16, ResNet-50, Darknet53*, entre otras.
- ***Neck*:** corresponde a un subconjunto, denominado en ingles "bag of special" que, básicamente, recopila mapas de características de diferentes etapas del *backbone*. En términos simples funciona como un agregador de características.
- ***Head*:** corresponde al detector de objetos, ya que su función es encontrar la región donde el objeto podría estar presente. Las arquitecturas mas utilizadas para esta etapa corresponden a *YOLO, SSD RetinaNet*, etc., para el caso de un detector de una etapa (One Stage Detector), y *Faster R-CNN, R-FCN* para un detector de dos etapas (Two Stage Detector).

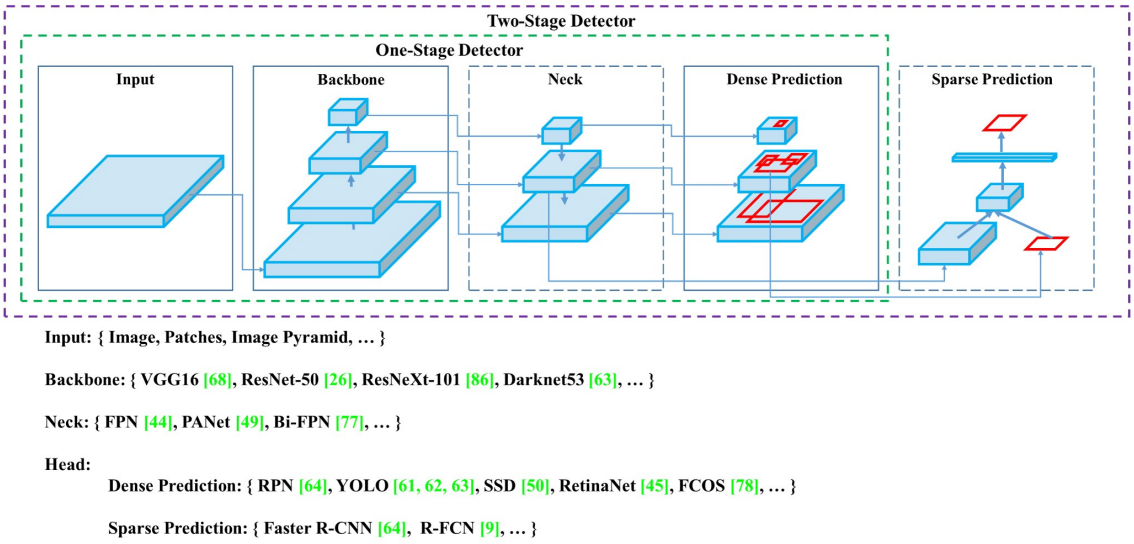


Figura 2.6: Representación arquitectura detector de objetos [7].

En la figura 2.7 es posible observar el desempeño de los detectores más utilizados, a partir de su velocidad y precisión. Cabe destacar que todas las pruebas fueron realizadas sobre el conjunto *COCO*, para diferentes detectores, donde se comparó los FPS (*frame per second*) según tres GPU, ya sea *Maxwell*, *Pascal* o *Volta*.

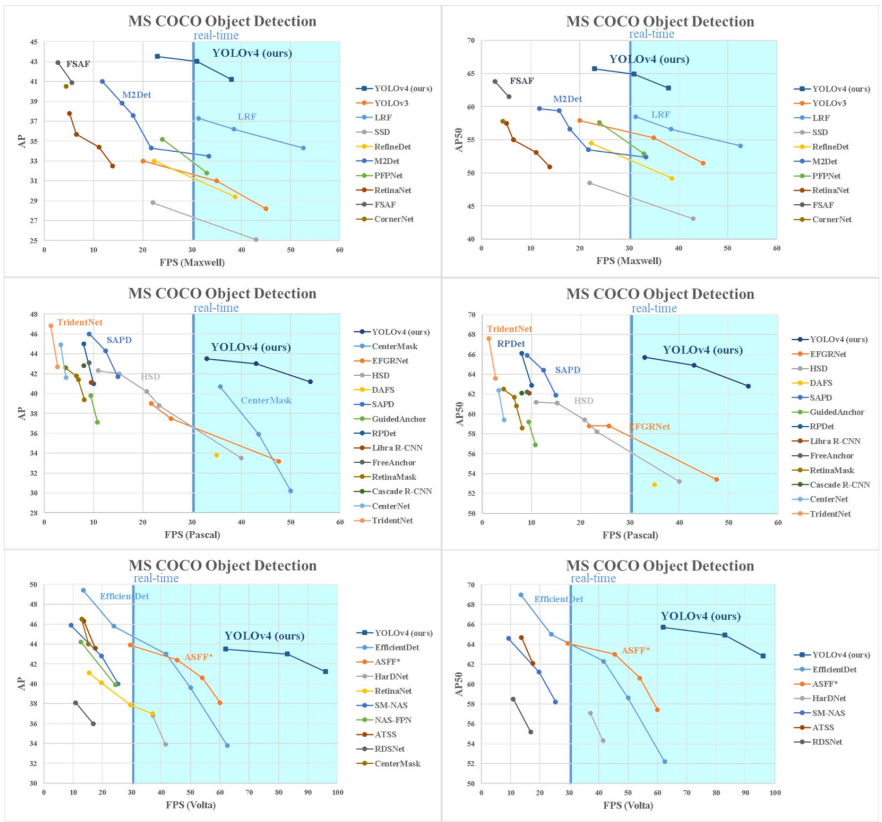


Figura 2.7: Comparación de velocidad y precisión de diferentes detectores [7].

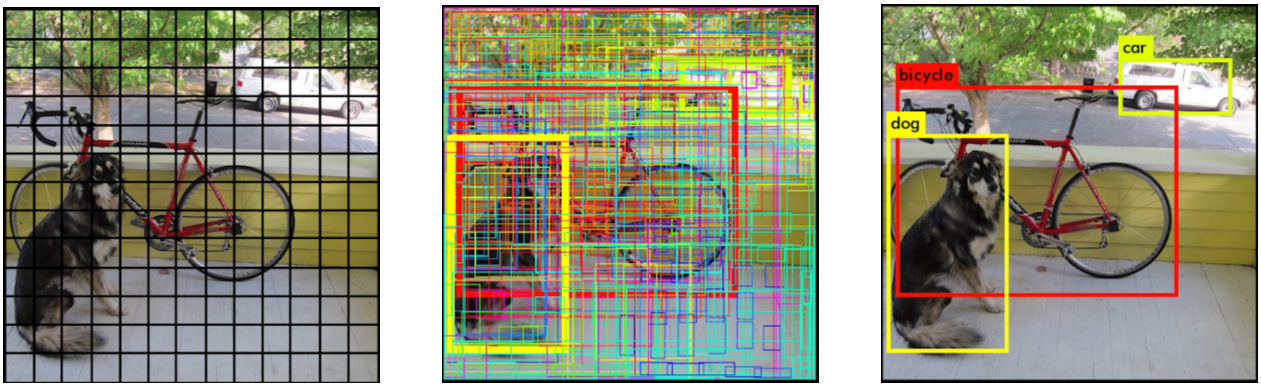
A partir de los datos mostrados en la figura 2.7, es que el detector escogido para la solución de este problema corresponde a YOLO, debido a su alta precisión y buen desempeño en aplicaciones en tiempo real, comparado con los otros detectores.

2.3.1. YOLO

YOLO (*You Only Look Once*), corresponde a un algoritmo utilizado para la detección de objetos en tiempo real, utilizando para ello una sola red neuronal, tanto como para la clasificación como para la predicción de los cuadros delimitadores de los objetos detectados. Su mayor atractivo radica en que se encuentra optimizado, de tal manera que presenta un alto rendimiento en la detección y puede ejecutarse mucho más rápido que la ejecución de dos redes neuronales separadas para detectar y clasificar objetos por separado.

Su funcionamiento se basa, a grandes rasgos, en 3 etapas [8]:

- La primera etapa consiste en dividir la imagen en una cuadricula de SxS, tal como se muestra en la figura 2.8a.
- El segundo paso consiste en que cada una de las celdas anteriores, predice N posibles "Bounding Boxes" y calcula el nivel de incertidumbre, obteniéndose así SxSxN posibles cajas, tal como como se muestra en la figura 2.8b.
- Finalmente, en el último paso, se procede a eliminar las cajas que se presentan un nivel de incertidumbre por debajo de cierto umbral. Luego, a las cajas restantes se le aplica lo que se conoce como "non-max suppression", que cumple la función de eliminar posibles objetos que fueron detectados por duplicado, tal como se muestra en la figura 2.8c.



(a) División Imagen. (b) Bounding Boxes. (c) Non-Max Suppression.

Figura 2.8: Ejemplo funcionamiento de detección YOLO [8].

2.3.2. Métricas y Conceptos Relevantes

Antes de definir las métricas que serán utilizadas para establecer que tan bueno será el modelo, resulta necesario conocer algunos conceptos, los cuales se presentan a continuación:

Dado un conjunto de clases binarias, denominadas $clase_0$ y $clase_1$ se tiene que:

- Verdadero Positivo: este concepto, representado generalmente por las siglas **TP** (del

inglés True Positive), indica la cantidad de casos que el algoritmo arroja como perteneciente a la $clase_1$, cuando realmente es de la $clase_1$.

- Verdadero Negativo: este concepto, representado generalmente por las siglas **TN** (del inglés True Negative), indica la cantidad de casos que el algoritmo arroja como perteneciente a la $clase_0$, cuando realmente es de la $clase_0$.
- Falso Positivo: este concepto, representado generalmente por las siglas **FP** (del inglés False Positive), indica la cantidad de casos que el algoritmo arroja como perteneciente a la $clase_1$, cuando realmente es de la $clase_0$.
- Falso Negativo: este concepto, representado generalmente por las siglas **FN** (del inglés False Negative), indica la cantidad de casos que el algoritmo arroja como perteneciente a la $clase_0$, cuando realmente es de la $clase_1$.

Accuracy, Precision, Recall e IOU

El *accuracy* es una métrica que cumple la función principal de otorgar información de cuantos *inputs* la red clasificó de forma acertada, sobre el total ingresado, sin embargo, esta métrica no debe usarse por sí sola, sobre todo en clases desbalanceadas. Por esta razón, es que se utiliza en conjunto con la métrica de *precision* y *recall*. Así, con la métrica de precisión es posible medir la calidad del modelo en las tareas de clasificación, entregando información sobre su rendimiento con respecto a los falsos positivos. En cambio, recall, también conocido como sensibilidad, nos da información sobre el rendimiento de un clasificador con respecto a falsos negativos. Finalmente, la métrica IOU (intersección sobre la unión) especifica la cantidad de superposición entre el cuadro delimitador predicho y el real. Por lo tanto, un IoU de 0 significa que no hay superposición entre las casillas, en cambio, un IoU de 1 significa que la unión de las cajas es igual a su superposición, lo cual indica que se superponen por completo.

El accuracy, precision, recall e IOU se puede calcular a partir de la siguientes expresiones:

$$Accuracy = \frac{TP + TN}{TP + TN + FT + FN} \tag{2.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall(TPR) = \frac{TP}{TP + FN} \tag{2.4}$$

$$IOU = \frac{Area \quad \text{de} \quad Interseccion}{Area \quad \text{de} \quad Union} \tag{2.5}$$

Además, la representación gráfica de las tres últimas métricas se puede observar en la figura 2.9.

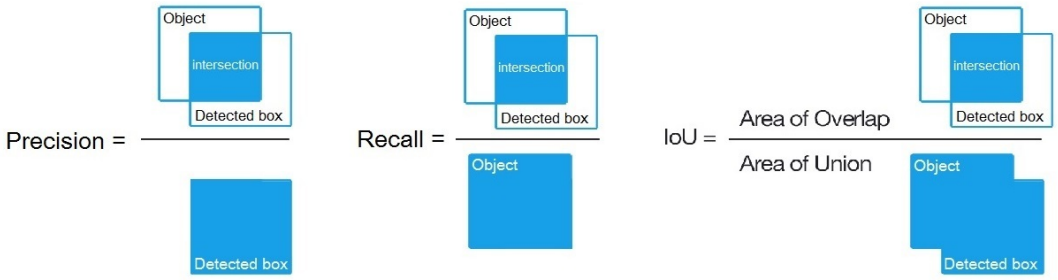


Figura 2.9: Representación de las métricas, Precision, Recall e IOU [9].

F_1 *score, AP y mAP*

Dada las definiciones anteriores, es posible notar que mientras mayor sea la *precisión*, mayor confianza tendrá el modelo cuando clasifique una muestra como positiva. Por otro lado, mientras mayor sea el *Recall*, más muestras positivas clasificará el modelo correctamente. Dado lo anterior, es que resulta necesario establecer un umbral para mantener un equilibrio entre estas dos métricas, ya que, si hay un alto recall y baja precisión existe la posibilidad de tener mayor cantidad de falsos positivos. En cambio, si la precisión es alta, pero el recall bajo, entonces el modelo es preciso cuando clasifica una muestra como positiva, pero puede clasificar solo algunas de las muestras positivas. Por esta razón se utiliza una métrica denominada f_1 score, la cual se calcula a partir de la ecuación 2.6, y mide el equilibrio entre precisión y recuperación. Por lo tanto, un valor de f_1 alto, significa que tanto la precisión como recall son altas, y un f_1 bajo, implica un mayor desequilibrio entre la precisión y recall.

$$f_1 = 2\frac{Precision * Recall}{Precision + Recall} \tag{2.6}$$

Para el calculo de mAP, se deben realizar los siguientes pasos [10]:

- Primeramente, a partir del dataset de estudio, para cada predicción se calcula el IOU, según cada cuadro etiquetado.
- Luego se genera una curva de *precision* versus *Recall* para cada clase y se calcula la precisión promedio (AP), tal como se muestra en la ecuación 2.7. Así, esta curva representa el rendimiento de un modelo con respecto a los verdaderos positivos, los falsos positivos y los falsos negativos en un rango de valores de confianza.

$$AP = \sum_{k=0}^{n-1} [Recall(k) - Recall(k+1)] * Precision(k) \tag{2.7}$$

- Finalmente, el mAP corresponde al promedio del AP de todas las clases, tal como se muestra en la ecuación 2.8.

$$mAP = \frac{1}{N} \sum_{\mathrm{i}=1}^{N} AP_{\mathrm{i}} \tag{2.8}$$

Capítulo 3

Estado del Arte

3.1. Investigaciones

En la actualidad, gracias al crecimiento del mercado IoT y el estudio de nuevas tecnologías, existen diversas investigaciones relacionadas con el tema de estacionamientos inteligentes o también conocidos como "Smart Parking". Las investigaciones actuales se pueden dividir en dos grandes categorías: se encuentran las que proponen como solución el uso de sensores por cada plaza de estacionamiento y las que utilizan un número reducido de cámaras, instaladas en lugares estratégicos que, mediante algoritmos de inteligencia artificial, determina si la plaza está vacía u ocupada.

3.1.1. Soluciones con sensores fijos

Entre las investigaciones con respecto a este tipo de implementación, la más destacada corresponde a un proyecto en San Francisco, denominado *SFpark*, el cual adopta una estructura de red de sensores inalámbricos, donde los datos fluyen desde los sensores de estacionamiento y los parquímetros hasta un almacén de datos a través de una red inalámbrica tipo malla. El sistema ofrece una solución completa de estacionamiento dentro y fuera de la vía pública y ofrece una serie de beneficios: más fácil de encontrar estacionamiento (el tiempo de búsqueda disminuye en un 43 %) y menor congestión (el volumen de tráfico disminuye en un 8 %). Otros proyectos similares son FASTPRK (magnético), Street Parking System (magnético), Smart Santander (ferromagnético), GEOmii (magnetómetro) y Smart Parking (infrarrojos), los cuales utilizan diferentes técnicas de detección, asociadas a otro tipo de sensores [11].

Otro enfoque diferente es el empleado en el proyecto Integrated Smart Parking Solution, dirigido por Siemens. En este caso se utilizan sensores de tipo radar que se montan en los faroles de la calle para escanear áreas más grandes. De esta forma los radares monitorean no solo los flujos de tráfico, sino también los espacios de estacionamiento, que pueden usarse para controlar el tráfico o para facilitar a los conductores encontrar un lugar para estacionar. Entre sus principales ventajas se encuentra que, en primer lugar, no se ve afectado por el clima o las condiciones de luz. En segundo lugar, detecta más que solo lugares de estacionamiento, ya que puede medir la velocidad del vehículo, los flujos de tráfico y los flujos de peatones. En

tercer lugar, se monta en farolas, lo que alivia los cambios de infraestructura [11].

En la figura 3.1 es posible observar un resumen de los proyectos implementados hasta la actualidad:

Name	Year	Sensing Technology	Sensor/spot	Accuracy**	Roadworks	Notes
ParkNet [8]	2010	Sonar	< 1	95%	No	Crowdsource sensing. Environmental fingerpinting to reduce GPS errors
ParkSense [9]	2013	Mobile Phones Wi-Fi	1	83%	No	Use of beacons between mobile phone and Wi-Fi to infer parking status
Street Parking System [10]	2013	Magnetic	1	98%	Yes	82 sensors are deployed at Shenzhen Institute of Advanced Technology
SFpark [5]	2014	Magnetometer	> 1	86%	Yes	Complete on/off-street solution
Smart Santander [11]	2014	Ferromagnetic	1	NS	Yes	Part of Smart City Project 375 sensors were deployed
FASTPRK [6]	Cnt.	Magnetic	1	95%	Yes	Company portfolio including sensing, analysis, open data interface and App
GEOmii [12]	Cnt.	Magnetometer	1	86%	Yes	Real time and predicted future parking space availability
Integrated Smart Parking [13]	Cnt.	Radar	< 1	NS	Mount	Also sensing traffic flow
Parking Spotter [14]	Cnt.	Sonar/Radar	< 1	NS	No	Crowdsource Sensing Ford proprietary application
Smart Parking [15]	Cnt.	Infrared + Magnetic	NS	NS	Yes	Company portfolio including sensing, guiding, payment and management

*- Cnt: Continuous, on-going projects — **- based on reported results, which may vary with selected parameters and scenarios — NS: Not Specified

Figura 3.1: Comparación de proyectos de estacionamiento [11].

3.1.2. Soluciones utilizando algoritmos de inteligencia computacional

Por otro lado, existe un enfoque diferente para solucionar este tipo de problemas, que se basan en el monitoreo mediante la captura de imágenes a través de cámaras, las cuales corresponden a los input de un algoritmo, por ejemplo red neuronal, y arrojan como resultado si la plaza esta ocupada o libre.

Entre las investigaciones actuales, una de las mas destacadas corresponde a la propuesta en el paper “Car parking occupancy detection using smart camera networks and Deep Learning"[12]. Para esta investigación se emplea las imágenes de la base de datos *CNRPark* y *PKLot*, cuyo conjuntos se detallan en la tabla 3.1.

Subset	free	busy	Total
CNRParkA	2549	3622	6171
CNRParkB	1632	4781	6413
CNRPark	4181	8403	12584
CNRParkOdd	2201	3970	6171
CNRParkEven	1980	4433	6413
CNRPark	4181	8403	12584
PKLot2Days	27314	41744	69058
PKLotNot2Days	310466	316375	626841
PKLot	337780	358119	695899

Tabla 3.1: Separación de los conjuntos utilizados para entrenamiento [12].

Donde:

- CNRPark A:contienen imágenes tomadas de una cámara con vista central.
- CNRParkB: contienen imágenes tomadas de una cámara con vista lateral.
- CNRParkOdd : contiene imágenes de espacios con números impares.
- CNRParkEven : contiene imágenes de espacios con números pares.
- PKLot2Days: se seleccionan para cada estacionamiento y para cada condición climática las imágenes de los dos primeros días en orden cronológico.
- PKLotNot2Days: resto de las imágenes.

La arquitectura utilizada para el entrenamiento fueron mAlexNet y mLeNet, que se muestra en la figura 3.2, las cuales son conocidas por ser exitosas en reconocimiento visual [13, 14].

net	*conv1*	*conv2*	*conv3*	*fc4*	*fc5*
mLeNet	30x11x11+4 pool 5x5+5 -	20x5x5+1 pool 2x2+2 -	-	100 ReLU	2 soft-max
mAlexNet	16x11x11+4 pool 3x3+2 LRN, ReLU	20x5x5+1 pool 3x3+2 LRN, ReLU	30x3x3+1 pool 3x3+2 ReLU	48 ReLU	2 soft-max

Figura 3.2: Arquitecturas de la Red CNN empleadas para el entrenamiento.

Así, los resultados obtenidos, utilizando los conjuntos de imágenes y las arquitecturas señalada anteriormente, se pueden observar en la figura 3.3, donde se aprecia que se realizaron diferentes experimentos, alternando los conjuntos de entrenamiento y test, las cámaras utilizadas, las arquitecturas de la red y el learning rate (tasa de aprendizaje).

Por otro lado, en la figura 3.4 es posible observar la comparación del mejor resultado para el algoritmo propuesto y otros modelos de investigaciones anteriores.

train	*test*	*net*	*base lr*	*accuracy*
A (even)	A (odd)	mLeNet	0.001	0.993
		mAlexNet	0.01	0.996
A (odd)	A (even)	mLeNet	0.001	0.982
		mAlexNet	0.005	0.993
B (even)	B (odd)	mLeNet	0.001	0.861
		mAlexNet	0.01	0.911
B (odd)	B (even)	mLeNet	0.001	0.893
		mAlexNet	0.005	0.898
MULTI CAMERA EXPERIMENTS				
train	*test*	*net*	*base lr*	*accuracy*
A	B	mLeNet	0.0001	0.843
		mAlexNet	0.001	0.863
B	A	mLeNet	0.001	0.842
		mAlexNet	0.0005	0.907

Figura 3.3: Resultados obtenidos para múltiples experimentos [12].

	INTRA-DATASET			**INTER-DATASET**	
train	PKLot-2D	CNRP.-Odd	CNRP.-Even	PKLot-2D	CNRP.
test	PKLot-N2D	CNRP.-Even	CNRP.-Odd	CNRP.	PKLot
model					
mAlex	0.981	0.901	0.907	0.829	0.904
LPQu	0.966	0.869	0.815	0.646	0.398
LPQgd	0.970	0.877	0.813	0.617	0.410
LPQg	0.957	0.869	0.816	0.638	0.438
LBPuri	0.874	0.850	0.763	0.653	0.497
LBPu	0.951	0.868	0.800	0.643	0.467
LBPri	0.879	0.865	0.820	0.642	0.487
LBP	0.945	0.874	0.872	0.631	0.529

Figura 3.4: Comparación con otros algoritmos [12].

Dado lo anterior, se puede notar que los resultados arrojados por esta investigación son exitosos, dado el alto accuracy obtenido y la variabilidad de las imágenes consideradas. Sin embargo, cabe destacar, que esta investigación solo desarrolla el algoritmo como tal, pero no lo implementa en ningún estacionamiento en especifico.

3.2. Bases de datos disponibles en la literatura

Las base de datos de uso libre ya disponibles, se detallan a continuación:

3.2.1. Find a Car Park:

Find a Car Park corresponde a una Base de datos de uso libre disponible en Kaggle [15]. Las capturas de imágenes fueron realizadas a partir de una cámara de Raspberry, obteniéndose así aproximadamente 1100 muestras de imágenes con plazas libres y 2200 muestras de imágenes con plazas ocupadas.

Un ejemplo de las imágenes disponibles en esta base de datos se puede apreciar en la figura 3.5.

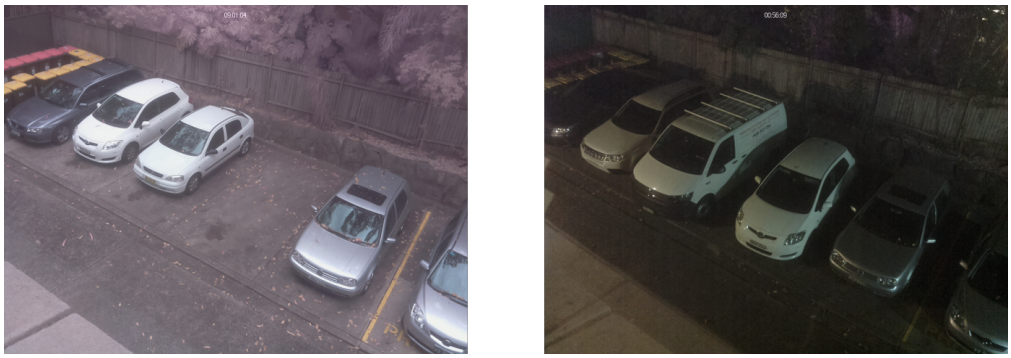


Figura 3.5: Ejemplo Base de Datos Find a Car Park.

3.2.2. CNRPark+EXT:

Esta base de datos contiene aproximadamente 150,000 imágenes etiquetadas según los espacios de estacionamiento vacíos y ocupados, construido a partir de un estacionamiento con 164 plazas.

CNRPark + EXT extiende el conjunto de datos preliminar CNRPark, que estaba compuesto por 12.000 imágenes recopiladas en diferentes días de julio de 2015 de 2 cámaras.

Así, el subconjunto adicional, llamado CNR-EXT, está compuesto por imágenes recolectadas desde noviembre de 2015 hasta febrero de 2016 bajo diversas condiciones climáticas por 9 cámaras con diferentes perspectivas y ángulos de visión. CNR-EXT captura diferentes situaciones de condiciones de luz e incluye patrones de oclusión parcial debido a obstáculos (árboles, farolas, otros autos) y autos con sombra parcial o global [16].

Un ejemplo de las imágenes disponibles en esta base de datos se puede apreciar en la figura 3.6.

Figura 3.6: Ejemplo Base de Datos CNRPark+EXT.

3.2.3. PKLot:

Esta base de datos contiene, aproximadamente, 12.000 imágenes capturadas de dos estacionamientos diferentes en días soleados, nublados y lluviosos. Además, el primer estacionamiento tiene dos ángulos de captura diferentes.

Las imágenes están organizadas en tres directorios (parking1a, parking1b y parking2). Cada directorio contiene tres subdirectorios para diferentes condiciones climáticas (nublado, lluvioso y soleado). Dentro de cada subdirectorio, las imágenes están organizadas por fecha de adquisición.

Cada imagen de la base de datos tiene asociado un archivo XML que incluye las coordenadas de todas las plazas de aparcamiento y su etiqueta (ocupada / desocupada). Al usar los archivos XML para segmentar el espacio de estacionamiento, se podrán obtener alrededor de 700.000 imágenes de espacios de estacionamiento [17].

Un ejemplo de las imágenes disponibles en esta base de datos se puede apreciar en la figura 3.7.

Figura 3.7: Ejemplo Base de Datos PKLot.

3.3. Implementaciones en Chile

Existen ya en el mercado empresas que ofrecen soluciones para el problema estudiado, incluso se han masificado aún más debido a la popularidad de los avances de dispositivos inteligentes, que permiten mayor acceso al usuario a estas herramientas. En Chile las empresas más destacadas en este rubro corresponden a Meste, Parkassist y Urbiotica.

3.3.1. Meste

Meste es una empresa cuya misión es ser proveedores de soluciones de ingeniería en el área de tecnología de: Grabación de la voz, Biométria, Parking y Seguridad. Entre sus proyectos más destacados en la subárea de estacionamientos, se encuentra un proyecto para la empresa de Kidzania[18]. La ubicación de este parque requería contar con urgencia con un sistema que permitiera a los clientes encontrar de forma rápida y eficiente los estacionamientos disponibles. Por ello, Meste ofreció como solución un sistema de guiado automático del tipo plaza a plaza para mejorar los flujos de los estacionamientos. Para lograrlo, se basó en tecnologías de luces y señalizadores, detectores especialmente desarrollados para estacionamientos, equipado con luces piloto, revisando constantemente la entrada y salida de cada vehículo y conectándose mediante la información almacenada en un servidor. Los resultados pueden apreciarse en la figura 3.8.

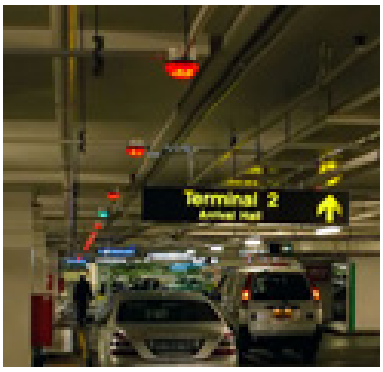


Figura 3.8: Solución empleada por Meste para Kidzania.

3.3.2. Parkassist

A diferencia de Meste, Parkassist es una empresa internacional, con oficina en nuestro país, que se especializa en mejorar los servicios entregados por propietarios y operadores de estacionamientos. Su enfoque es ayudar al usuario a tener la mejor experiencia y evitar el estrés a la hora de estacionarse. Para ello ofrece un servicio de orientación con la última tecnología, mediante el uso de sensores inteligentes y señalización personalizable [19]. Además, ofrece servicios extras, como mostrar el número de estacionamientos disponibles por nivel al momento de la llegada del usuario. Los resultados de la implementación de este servicio se muestran en la figura 3.9.

Figura 3.9: Solución empleada por Parkassist.

3.3.3. Urbiotica

Urbiotica, al igual que Parkassist, es una empresa internacional, que también ofrece sus servicios en Chile. Se especializa en soluciones inteligentes utilizando tecnología IoT y sensores inalámbricos aplicado al sector urbano, de tal manera de obtener finalmente *Smart Cities*. La solución ofrecida consiste en desplegar sensores U-Spot en las plazas de estacionamiento exteriores y U-Flows en las entradas y salidas de los aparcamientos de los dos centros comerciales, en el sector de Las Condes, Chile [20]. Los datos que captan se transforman en información útil que informa a los clientes sobre la disponibilidad de plazas mediante paneles informativos y aplicación móvil que permite conocer en tiempo real la situación de plazas disponibles así como la regulación definida. El funcionamiento de la implementación de este servicio se muestra en la figura 3.10.

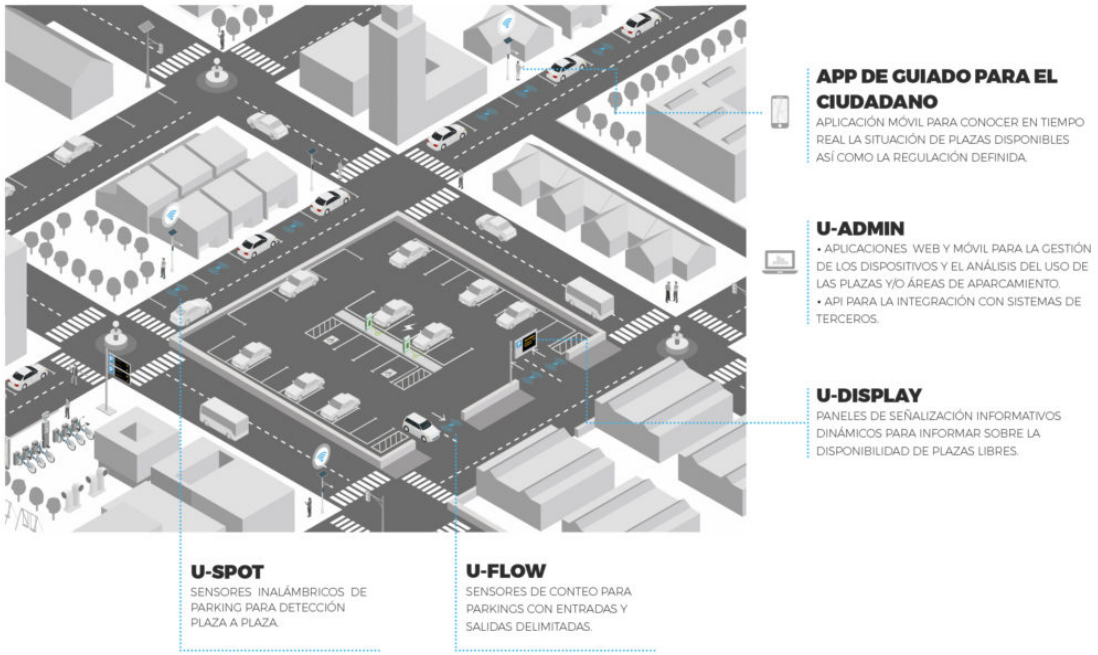


Figura 3.10: Solución empleada por Urbiotica.

Dado lo anterior, se puede observar que los sustitutos ofrecidos por las 3 empresas que controlan el mercado son similares y consisten en la instalación de sensores, letreros y paneles

programables que indican la dirección de vacantes disponibles, donde la que más destaca es Urbiotica, ya que la solución propuesta, aparte de todo lo anterior, incluye la implementación de una aplicación móvil que guía al usuario hacia una vacante disponible. Sin embargo, ninguna de las implementaciones anteriores contempla en su solución el uso de procesamiento de imágenes y algoritmos de inteligencia artificial que se proponen en esta memoria.

Capítulo 4

Diseño de la Solución

La metodología utilizada para dar solución al problema planteado, se puede dividir en 3 etapas principales; la primera etapa corresponde al proceso de captura de imágenes, en el cual se deben establecer los parámetros óptimos para generar una base de datos que utilizará el sistema de detección. Luego de la captura de imágenes, la segunda etapa consiste en la implementación del algoritmo detector de plazas disponibles/ocupadas, para lo cual se hará uso de una red preentrenada y la arquitectura de YOLOV4. Además, una vez que se realice el entrenamiento, las métricas presentadas en la sección 2.3.2, permitirán evaluar si este es bueno o no. Finalmente, en la última etapa luego de encontrar los hiperparámetros óptimos, se implementará el modelo en el hardware Jetson Nano Nvidia, donde, además, se creará una interfaz de usuario amigable para observar los resultados del algoritmo de detección aplicados a una imagen o vídeo.

Un resumen, de lo anterior, se puede observar en el diagrama de la figura 4.1.

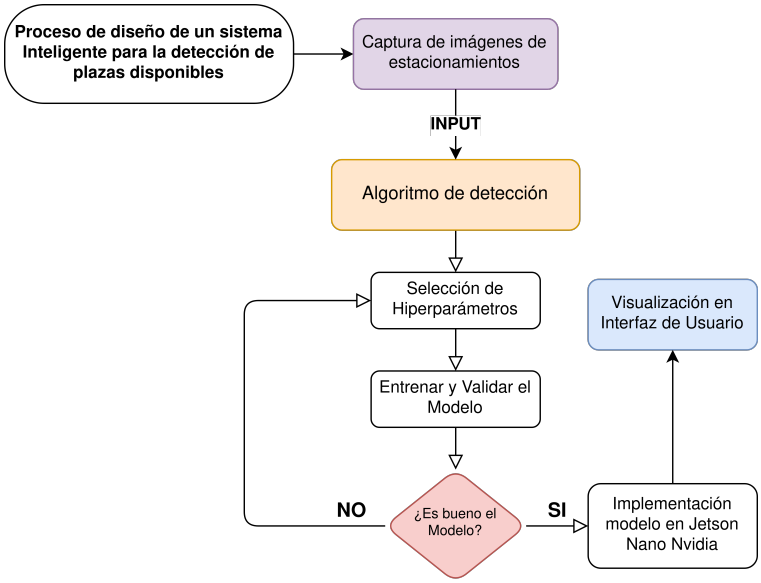


Figura 4.1: Diagrama de la Metodología seguida.
Elaboración propia.

En las siguientes secciones se explicará en más detalle cada etapa del proceso que permitió dar solución al problema.

4.1. Base de datos recopilada

4.1.1. Captura de las imágenes

Para realizar las pruebas del sistema creado se recopiló una nueva base de datos, a partir de un estacionamiento outdoor ubicado en un condominio. Las tomas de imágenes fueron realizadas desde un departamento ubicado en el tercer piso, perteneciente a dicho lugar, obteniéndose así una vista superior de cada plaza. Una representación de lo anterior se puede visualizar en la figura 4.2.

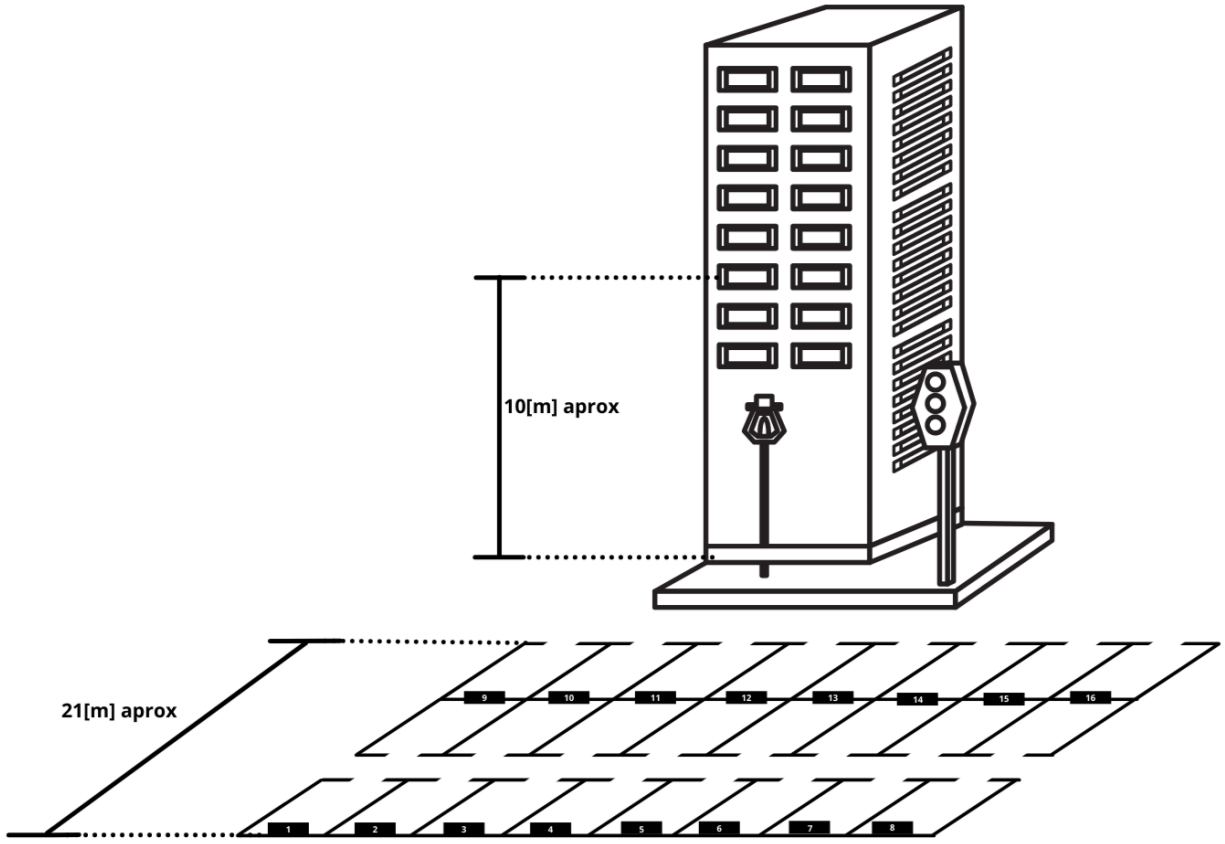
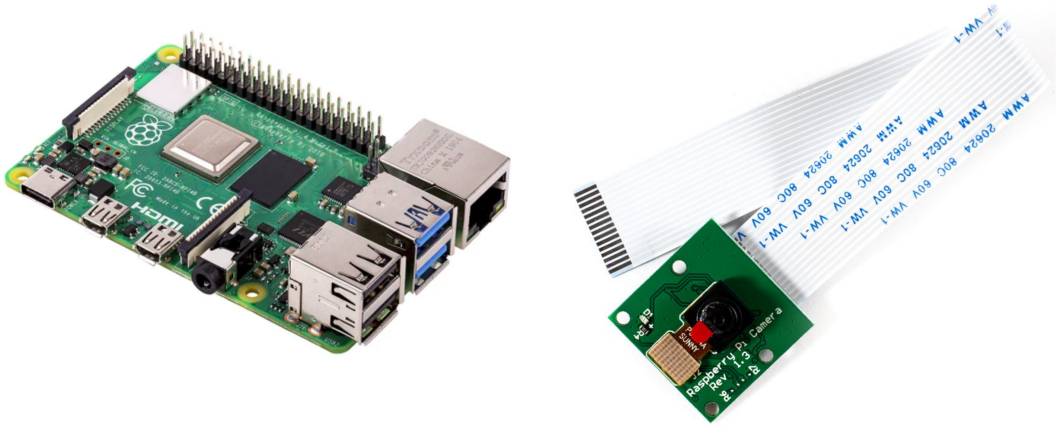


Figura 4.2: Representación ilustrativa del estacionamiento de estudio.
Elaboración propia.

El sistema implementado para la captura de imágenes consta de una Raspberry Pi 4 modelo B con 4GB de memoria RAM, y una Cámara Raspberry Rev 1.3 de 5 MP, el cual se muestra en la figura 4.3, cuyos precios y proveedores se pueden apreciar en el anexo A.



a) Raspberry Pi 4B b) Cámara Rev 1.3 5MP

Figura 4.3: Hardware Utilizado para la captura de imágenes.

Dada la baja capacidad de la cámara, las muestras de imágenes capturadas durante el periodo nocturno presentaban mala calidad, siendo imposible casi observar cada plaza de estacionamiento, tal como se muestra en la subfigura 4.4a. Por esta razón, es que se realizaron diferentes configuraciones en la programación de la captura de imágenes, dependiendo si era de día o de noche. Especialmente en el caso nocturno, la cámara debía realizar una mayor apertura para lograr captar un mayor nivel de luz y así, obtener imágenes con mayor calidad y más nitidez. Sin embargo, si se utilizara este mismo método para el caso diurno, capturaría demasiado luz, lo que arroja imágenes demasiados claras, como se observa en la subfigura 4.4b.

(a) Ejemplo captura noche (b) Ejemplo captura día

Figura 4.4: Ejemplos de captura de imagen con parámetros no optimizados

Luego de realizadas las modificaciones, se lograron mejores condiciones de luz, obteniéndose la calidad óptima, tanto para el día y noche. Cabe destacar que cada captura se realizaban un intervalo de 5 minutos, dado que al ser un estacionamiento residencial no había mayor variabilidad, por lo que no se requería de tomas exactamente continuas. Además, se estableció una hora en la cual debían actualizarse los parámetros de captura para pasar de un modo diurno a nocturno, a las 19:00 hrs, y de modo nocturno a diurno nuevamente a las 7:00 hrs. Las líneas de código escritas en python que llevan a cabo estas instrucciones se muestran en el Anexo B.1.

En la figura 4.5 se muestra un ejemplo de la base de datos recopilada luego de implementar las configuraciones antes mencionadas.

(a) Ejemplo captura noche. (b) Ejemplo captura día.

Figura 4.5: Ejemplo Base de Datos Recopilada.

4.1.2. Asignación de etiqueta

Una vez obtenida la base de datos, resulta necesario realizar el proceso de asignación de las etiquetas free (espacio disponible) o busy (espacio ocupado) a cada plaza, según sea el caso de cada imagen. Para lograr lo anterior, se hizo uso del tool gráfico *LabelImg*[21] para realizar las anotaciones, cuyo ejemplo se puede apreciar en la figura 4.6, en el cual se muestra cada espacio delimitado por un rectángulo, siendo el color verde asignado para delimitar los espacios ocupados y el color morado para delimitar los espacios libres.

Una vez asignada la etiqueta y guardada, el programa genera un archivo txt, en donde cada fila representa, con un total de 5 valores separados por un espacio, al rectángulo que delimita la plaza. El primer valor corresponde a un número entero que va desde 0 hasta el número de clases etiquetadas, que en este caso corresponde a dos clases, en donde 0 representa a la clase denominada *free* y 1 representa a la clase denominada *busy*. El segundo y tercer valor, corresponde a números decimales, y representa la coordenada $(x_{\mathrm{i}}, y_{\mathrm{i}})$ del centro de la

máscara rectangular que delimita la $plaza_i$. Por último, el cuarto y quinto valor representan el ancho y alto de la máscara rectangular, respectivamente. Cabe destacar que estos últimos 4 valores son normalizados por el ancho y alto en píxeles de la imagen, de ahí que su valor se encuentre en notación decimal. Además, se crea un archivo único por carpeta denominado *classes.txt*, el cual contiene tantas filas como número de clases diferentes fueron etiquetadas. En este caso posee solo dos filas, en donde la primera contiene el string *free* y la segunda el string *busy*.

Figura 4.6: Ejemplo uso de LabelImg para etiquetado de la base de datos.

Como se puede notar en la figura anterior, el estacionamiento de estudio posee un total de 24 espacios para estacionar (8 por cada fila), los cuales no presentan un delimitador notorio entre cada plaza, como es común ver en estacionamiento típicos, donde suelen ser demarcados con líneas blancas o amarillas. Sin embargo, en este caso, solo es posible notar un delimitador poco distintivo en forma de "L" para indicar la separación, además de un marcador en un pequeño bloque blanco que indica una numeración única, asignada a cada departamento, lo cual es posible observar en la figura 4.7. Cabe destacar que en el caso de la segunda fila de estacionamientos, la numeración se encuentra por el lado contrario al que fue capturada la imagen, por lo que la numeración no queda visible en el plano de estudio.

Figura 4.7: Ejemplo delimitadores presentes en el estacionamiento de estudio.

Dado lo anterior, es que se realizaron las siguientes consideraciones al momento de asignar las etiquetas:

- ▷ Se delimitó cada plaza con un rectángulo, de tamaño tal que, en caso de estar desocupado, logrará incorporar el bloque blanco y al menos una marca separadora (las que son en forma de "L"). Si bien es cierto la idea era incorporar las dos marcas laterales en forma de "L", dado el ángulo desde que fue capturada la imagen, esto no fue posible, ya que el plano no quedó del todo frontal. Además, los rectángulos del programa *LabelImg* no poseen la función de rotar, de manera de marcar mejor la plaza con respecto al ángulo de captura.
- ▷ Como se mencionó anteriormente, en el caso de la segunda fila de estacionamientos, la numeración en el bloque blanco no quedaba visible en el plano de captura, y no era posible la instalación de una cámara desde esa vista, por ello es que para las demarcaciones de los rectángulos de esa fila se incluyó dentro de éste el bloque de la tercera fila, de esta forma la segunda y tercera fila comparten el mismo bloque y quedan superpuestos, como se observa en la figura4.6 de ejemplo.
- ▷ Otro punto importante a destacar es que, debido al ángulo de captura, en muchas ocasiones el vehículo sobrepasaba el área delimitada por estacionamiento, tal como se observa en el rectángulo azul en la figura 4.6. En este caso, el vehículo se muestra fuera del área, sin embargo, se optó por delimitar siempre el área correspondiente a la plaza de estacionamiento como si estuviese vacía, sin importar que ciertas partes del vehículo quedaran fuera de este contorno. De esta forma siempre se respeta el área de cada plaza.

4.1.3. Dataset Obtenido

En la tabla 4.1 se puede apreciar un resumen de la cantidad de imágenes capturadas, etiquetas asignadas y distribución por cada clase. Cabe destacar que la cantidad de imágenes

obtenidas fue alrededor de 5000 imágenes, las cuales se dividieron en dos subconjuntos, denominados *Dataset Captura Imagen Día* y *Dataset Captura Imagen Noche*, correspondientes a las capturas optimizadas según día y noche, respectivamente.

Por otro lado, también es posible notar que la cantidad de imágenes etiquetadas para el Dataset Día es de menos del 10 % del total de ese dataset, y aún menos del 5 % para el Dataset Noche. Lo anterior es debido a que el estacionamiento de estudio es de tipo residencial, por lo que se presenta poco flujo vehicular, es decir, en algunos momentos del día el estado de las plazas se mantiene constante hasta incluso por horas, presentándose mayor variabilidad en horarios matutinos y durante la tarde, que coincide con el horario laboral de entrada y salida o también conocido como horario punta. En el caso nocturno es aún menos variable, ya no suele existir mucho flujo de vehículos durante la noche, sobre todo porque la pandemia aún se encuentra presente y no hay gran presencia de locales abiertos, ya sea de entretenimiento o comercio, lo cual reduce las posibilidades de transitar durante la noche. Cabe destacar que, en el caso del fin de semana, tanto durante el día como la noche, el flujo de vehículos aumentaba, dado que en el condominio de estudio era común que los departamentos se arrendaran, sobre todo porque Algarrobo es considerado un atractivo destino turístico.

	Dataset Captura Imagen Día	Dataset Captura Imagen Noche	Total
Cantidad de Imágenes Capturadas	2685	2571	5256
Cantidad de Imágenes Utilizadas para Etiquetado	170	89	259
Cantidad de etiquetas Free	3408	1725	5133
Cantidad de etiquetas Busy	672	411	1083
Total Etiquetas	4080	2136	6216

Tabla 4.1: Resumen de cantidad de imágenes capturadas y cantidad de imágenes etiquetadas.

Resulta necesario destacar, que si bien es cierto se podría haber realizado las asignaciones de etiqueta a todas las imágenes capturadas, esto no aportaba variabilidad al conjunto, que es lo que se busca al momento del entrenamiento. Por otro lado, aunque se buscaba seleccionar las capturas de modo tal que no hubiese tomas repetidas, sí existen casos sin cambio en el estado de las plazas de estacionamiento, pero si en las condiciones de luz ambiental a lo largo del día o la noche, las cuales sí fueron consideradas.

4.2. Modelo Utilizado

Para lograr identificar el estado de las plazas de estacionamiento se utilizó el algoritmo YOLO V4 (You Only Look Once), lo cual se llevó a cabo en las siguientes 3 etapas:

4.2.1. División de dataset para entrenamiento

La primera etapa consiste en realizar la separación de cada dataset, ya sea día o noche, en formato *train*, *valid* y *test*, donde se utilizó una partición de 60 %, 20 % y 20 %, respectivamente. Dado lo anterior, es que para obtener mayor variabilidad se hizo uso de diferentes configuraciones de estas subdivisiones, dando origen a 3 experimentos mostrados en la tabla 4.2. En esta se detalla cada subconjunto utilizado para la etapa de entrenamiento y validación, así como la distribución de clases, free y busy, con la que se llevó a cabo cada experimento.

	Experimento 1	Experimento 2	Experimento 3
Train Dataset	Dataset Dia	Dataset Dia	Dataset Dia+Noche
Valid Dataset	Dataset Dia	Dataset Noche	Dataset Dia+Noche
Total Label Free	2742	3116	4144
Total Label Busy	547	654	874

Tabla 4.2: Resumen de los conjuntos utilizados en cada experimento realizado.

4.2.2. Implementación del modelo

Para la implementación del modelo, se utilizó como base el repositorio de GitHub de *Alexey*[9], que presenta el código necesario de Yolo basado en Darknet. Dado lo anterior, el primer paso consiste en clonar el repositorio a tu entorno de trabajo. Para este caso fue el entorno de *Google Colab*, que es un entorno de uso libre y permite ejecutar códigos de forma online. Además, durante el entrenamiento provee el uso de GPU, si así se requiere, que en este caso fue una *Tesla k80*.

En el formato actual del repositorio, YOLO ya permite detectar 80 clases, sin embargo, para que sea capaz de detectar plazas vacías u ocupadas de un estacionamiento, es necesario realizar nuevas configuraciones y un entrenamiento, utilizando como base un peso de red neural ya preentrenado, que en este caso será el *yolov4.conv.137*[22], de la versión 4 de YOLO.

El siguiente paso consiste en crear el archivo *Makefile*, el cual contiene todas las configuraciones para ejecutar el entrenamiento de la red, entre las que destacan el uso o no de GPU y CUDNN, además de la correspondiente configuración de acuerdo a cual tipo de GPU se va a utilizar. El archivo Makefile utilizado en este proyecto se encuentra adjunto en el anexo B.2.

Después de realizar lo anterior, deben copiarse los directorios que contienen el dataset, es decir, las carpetas *train*, *valid* y *test* al entorno de trabajo donde fue clonado el repositorio.

Cabe destacar, que es necesario modificar el formato del archivo *classes.txt* por *darknet.labels*, para que las clases sean reconocidas por la red. Posteriormente, las imágenes del directorio train y valid son copiadas en la ruta *data/obj/* y, además, se creará un documento .txt por cada conjunto, denominado train.txt y valid.txt, que contendrá una lista con las rutas a las imágenes de cada conjunto. Sumado a lo anterior, se copiará el archivo *darknet.labels* en el directorio *data/* y se renombrará como *obj.names*.

Finalmente, se creará un archivo denominado *obj.data* en el cual se especificarán el número de clases y las rutas a los archivos train.txt, valid.txt, obj.names y a la carpeta *backup/*, donde son guardados los pesos durante el entrenamiento, tal como se muestra a continuación:

```
classes = 2
train = data/train.txt
valid = data/valid.txt
names = data/obj.names
backup = backup/
```

Una vez configurados los directorios, se procede a realizar la configuración de la arquitectura de la red, cuyo código se puede visualizar en el anexo B.3. Mediante estas instrucciones es posible configurar el tamaño de la red, learning rate, tamaño del batch, función de activación, entre otros.

Finalmente, se procede a compilar el archivo makefile mediante el comando *./make* y posteriormente ejecutar el entrenamiento utilizando el comando *./darknet detector train data/obj.data cfg/custom-yolov4-detector.cfg yolov4.conv.137 −map −dont_show*, pasando como parámetro los archivos configurados anteriormente.

4.3. Interfaz Gráfica

La interfaz gráfica utilizada para mostrar al usuario las plazas disponibles fue realizada en python y *Qtdesigner*. Para ello, se hizo uso de una plantilla base disponible en el repositorio [23], donde se encuentran los requerimientos necesarios para su uso. Cabe destacar que esta interfaz puede ser utilizada con la librería *pyside2* o *pyqt5*, donde se optó por esta última.

Una vez instaladas las librerías, se procedió a modificar la plantilla (*template*) para agregar las funcionalidades requeridas para este proyecto. Para ello se hizo uso de *Qtdesigner*, que corresponde a una tool amigable para diseñar interfaces de usuario mediante el uso de widgets y bloques, lo cual posteriormente se transforma a líneas de código de python. Un ejemplo de la interfaz de este tool se puede apreciar en la figura 4.8, donde se editó el archivo GUI_BASE.ui del repositorio.

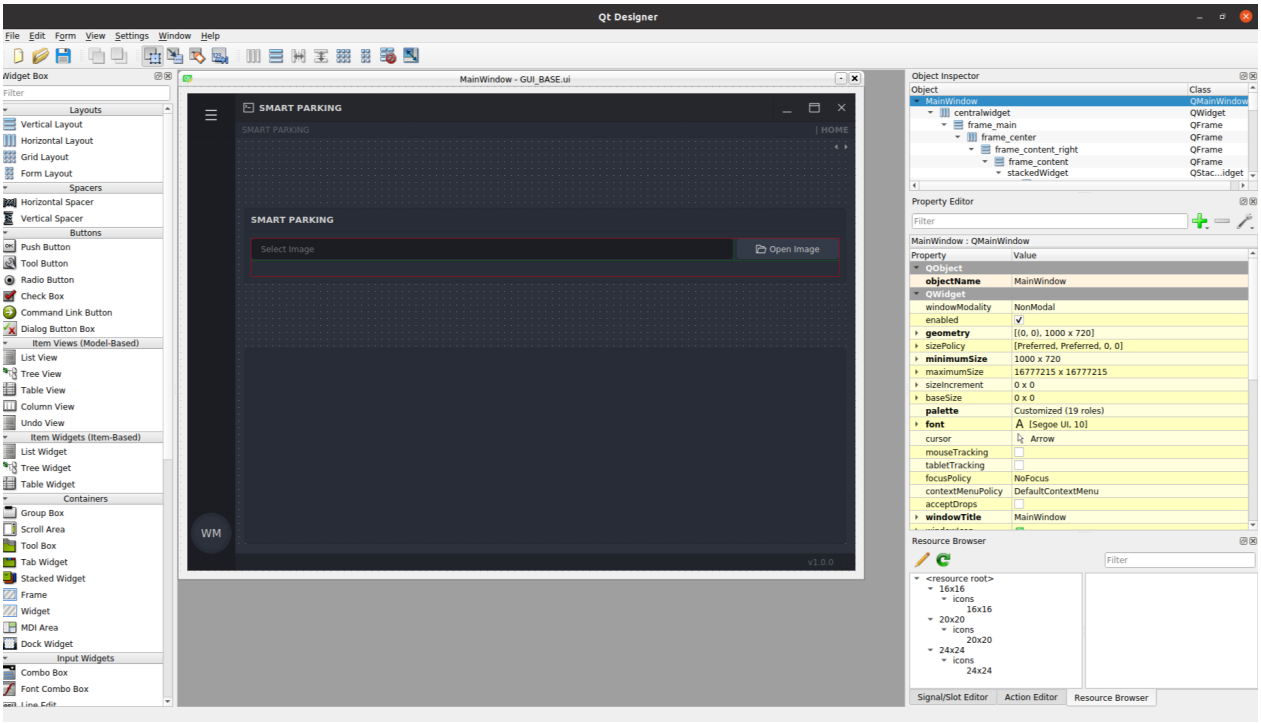


Figura 4.8: Ejemplo diseño de interfaz realizada con el tool Qt Designer.

El tool descrito permite interactuar, principalmente, de 3 formas; primeramente, como se muestra de la figura 4.8, se encuentra el componente central, que corresponde al entorno de trabajo para crear la interfaz. Por otro lado, en el sector izquierdo es posible notar lo que se conoce como *widget box*, lo cual es un menú que contiene todos los elementos necesarios a utilizar para interactuar con el usuario, desde cuadros de texto básico, botones, vistas, hasta herramientas más útiles que permiten desplegar gráficos y animaciones. Finalmente, el menú situado al lado derecho permite cambiar el estilo de cada widget, ya sea tamaño, color, forma, entre otros.

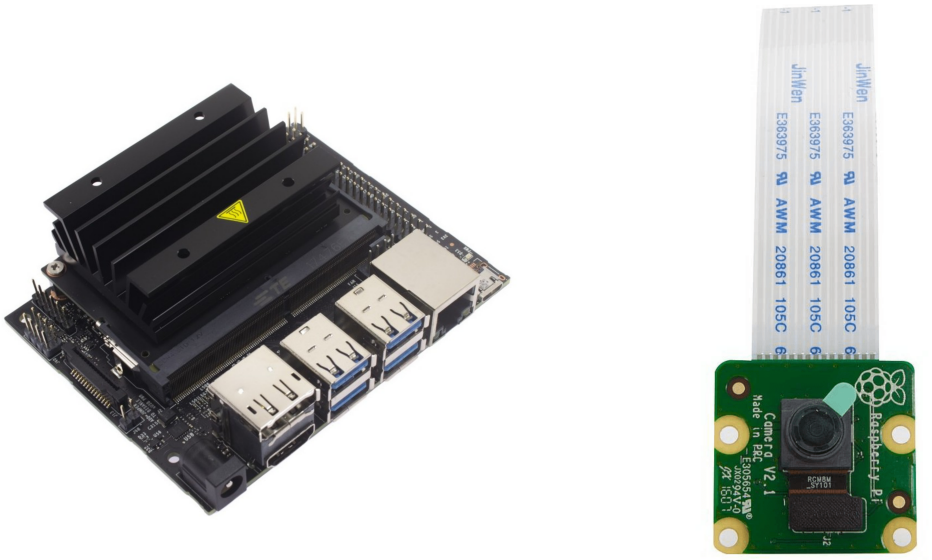
Para efectos de los objetivos de esta memoria se decidió solo crear una interfaz simple de carácter únicamente ilustrativo. Por ello, es que el diseño se basó en implementar una función que pudiese mostrar al usuario el estado de plaza, incorporando para ello un botón que permitiera cargar una imagen seleccionada por el usuario y aplicar el algoritmo de detección.

Cabe destacar que la función de selección de imagen por el usuario para aplicar el detector, no es el caso idóneo, ya que la imagen debería tomarse en tiempo real al estacionamiento, sin embargo, ya no se cuenta con acceso al recinto, por ello es que se optó por sustituir estas funciones para simular el proceso de detección.

Una vez que el diseño esta listo, se ejecuta el comando $<pyuic5\ -x\ GUI\_BASE.ui\ -o\ main.py>$, para convertir la interfaz gráfica a formato de líneas de código en python.

4.4. Implementación en Jetson Nano Nvidia

Para utilizar la Jetson Nano Nvidia es necesario, en primer lugar, realizar la instalación básica del sistema operativo disponible en [24]. En este caso se utilizó la versión *JetPack 4.6*, que incluye ya instaladas ciertas librerías útiles para el procesamiento de imágenes, entre las que destacan *cuDNN 8.2.1* y *CUDA 10.2*, los cuales son la base para utilizar el detector basado en YOLO. Dado que ya vienen instaladas las librerías base, el modelo puede ser utilizado con otras librerías básicas y utilizando el repositorio disponible en [9].



a) Jetson Nano Nvidia b) Cámara V2 8MP

Figura 4.9: Hardware Utilizado en la implementación final.

Capítulo 5

Resultados y Análisis

5.1. Resultados entrenamiento

En la figura 5.1 es posible apreciar como varía el mAP (en %), a medida que aumentan las iteraciones durante el entrenamiento. Cabe destacar que para este caso se consideró un cálculo inicial del mAP desde la iteración 200, hasta la iteración 1000, con un intervalo de 50 iteraciones, pudiéndose observar las curvas obtenidas según cada experimento descrito en la sección 4.2.1.

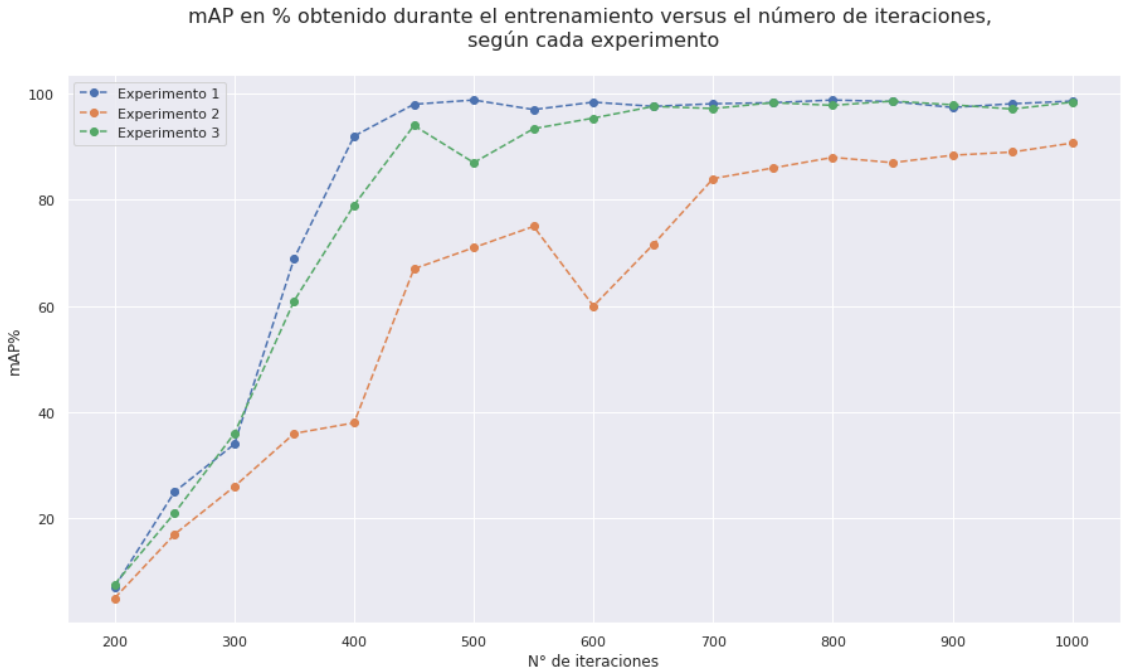


Figura 5.1: mAP versus número de iteración para cada experimento, durante entrenamiento.Best mAP experimento 1= 98.6 %, Best mAP experimento 2= 90.7 %, Best mAP experimento 3= 98.9 %.

De la figura 5.1, se observa que el experimento 1, cuya curva está representada por el

color azul, tiene un crecimiento más rápido del mAP respecto a los otros dos experimentos, convergiendo más rápido al mAP óptimo, ya que a partir de la iteración número 600, aproximadamente, ya se estabiliza su valor. Por otro lado, el experimento 2 presenta un crecimiento mucho más lento, y menos estabilización, presentado incluso peaks notorios de decaídas durante el entrenamiento y demorando más en alcanzar su valor de mAP óptimo. Lo anterior era esperable, dado que para este caso el dataset de validación utiliza solo imágenes capturadas durante la noche, por lo que las condiciones ambientales son completamente diferentes del dataset que se está utilizando en el conjunto de train, en cual solo se encuentran imágenes capturadas durante el día. Finalmente, el experimento 3 presenta características similares durante el entrenamiento al experimento 1, así como también en los valores óptimos del mAP. Aunque durante el crecimiento es un poco menos estable antes de la convergencia, presentando un peak de decaimiento del mAP, sin embargo, luego se estabiliza rápidamente. Lo anterior era lo esperado, ya que en ambos casos se utilizaron, tanto para train como valid, dataset bajos las mismas condiciones ambientales, ya sea en el experimento 1 donde el dataset utilizado eran capturas tomadas solamente durante el día, o en el experimento 3, donde los conjuntos tenían imágenes de capturas de día y noche.

Cabe destacar, que si bien es cierto desde un principio se podría solo haber realizado el experimento 3, donde se utilizan las dos condiciones ambientales, para el conjunto train y valid, lo que se buscaba era encontrar la mínima variabilidad que se necesita a la hora de capturar imágenes para obtener un buen modelo. Es decir, con esto se buscaba comprobar si realmente la condiciones de luz ambiental, a la hora de entrenar y validar un modelo, causaba un efecto significativo en las predicciones. Esto, ya que en muchas ocasiones no se dispone de un dataset con toda la variabilidad deseada. Así, dado lo anterior se puede apreciar que claramente los modelos que poseían los mismo tipos de imágenes en ambos conjuntos presentaron mejores resultados en la evaluación del entrenamiento, sin embargo, también resulta interesante que aun cuando el experimento 2 presentaba un conjunto de dataset con diferentes tipos de capturas para train y valid, este arrojó un resultado aceptable.

5.2. Resultados en conjunto test

Para evaluar el modelo se procedió a utilizar los conjuntos Test, previamente obtenidos de la división de la sección 4.2.1. Los tres subconjuntos fueron denominados con los nombres *test A*, *test B* y *test C*, los cuales se componen de:

- Test A: Sub-Dataset con imágenes capturadas durante el día.
- Test B: Sub-Dataset con imágenes capturadas durante la noche.
- Test C: Sub-Dataset con imágenes capturadas durante el día y noche.

Además, en la tabla 5.1 se puede apreciar la distribución de las clases *free* y *busy* en cada sub-dataset.

	Test A	Test B	Test C
Cantidad de etiquetas Free	668	324	992
Cantidad de etiquetas Busy	125	84	209
Total Etiquetas	793	408	1201

Tabla 5.1: Distribución de clases busy y free para los sub-conjuntos test.

Una vez obtenido los subconjuntos, se procedió a evaluar cada uno de estos en los modelos de cada experimento, utilizando los respectivos pesos obtenidos durante el entrenamiento, con lo cual resultaron 9 combinaciones posibles. En la tabla 5.2 es posible apreciar un resumen de cada métrica obtenida, según cada subconjunto test_i, evaluado en el modelo del $\text{experimento}_\text{i}$. Además, también se agrega el resultado del mAP de cada entrenamiento.

		Experimento 1	Experimento 2	Experimento 3
Best Train	mAP %	98.6	90.7	98.9
Test A	mAP %	98.76	98.99	99.30
	Precision %	97	97	98
	Recall %	99	99	99
	F1-score %	98	99	98
	Average IoU %	75.70	75.74	74.86
Test B	mAP %	84.64	90.61	99.22
	Precision %	91	90	96
	Recall %	73	91	99
	F1-score %	81	91	97
	Average IoU %	67.16	67.47	75.17
Test C	mAP %	94.14	96.26	99.31
	Precision %	95	95	96
	Recall %	90	96	99
	F1-score %	93	96	97
	Average IoU %	73.25	72.94	74.96

Tabla 5.2: Resumen resultados evaluación sub-conjuntos test, para cada experimento.

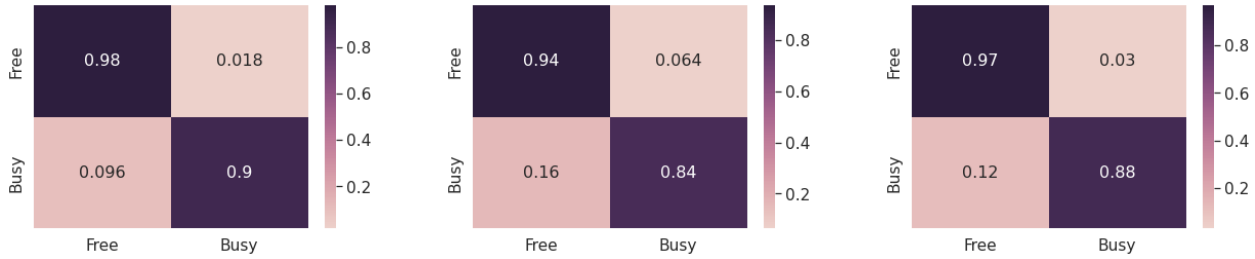
De la tabla anterior es posible notar que, tanto en el entrenamiento como en la evaluación de los conjuntos test, el mAP arroja valores sobre el 90 %, destacándose la mejor combinación para el experimento, con un valor de mAP de 98.9 % para el conjunto de entrenamiento y un mAP de 99.3 %, 99.22 % y 99.21 % para los conjuntos *Test A*, *Test B* y *Test C*, respectivamente. Por otro lado, también se obtienen las mejores métricas de precisión, recall, f1 score e IOU para este experimento. Lo anterior era lo esperado, ya que durante el entrenamiento de este modelo se utilizaron muestras de imágenes con luz ambiental diurna y nocturna, por lo que era de esperarse que todos los sub conjuntos de test arrojaran los mejores resultados para dicho modelo. Por otro lado, resulta interesante que aun cuando los otros modelos y

combinaciones poseen menor rendimiento, en cuanto a las métricas obtenidas, estos siguen siendo buenos, arrojando en la mayoría de las métricas sobre el 90 %. Por lo tanto, se puede apreciar que aun cuando el factor de luz ambiental es relevante a la hora de entrenar, de todas formas se obtienen modelos aceptables, aunque no se tenga la variabilidad deseada, lo cual resulta de vital importancia, ya que en escenarios reales muchas veces no se tienen datos con todas las variabilidad ambiental que se desearía.

Resulta interesante destacar, que para el caso de la métrica IoU, ésta no alcanzó un valor tan alto como las demás, sin embargo, es lo esperado ya que como se explicó en secciones anteriores las plazas de estacionamiento no poseen una marca separadora tan distintiva, lo que hace difícil la detección de las plazas, sobretodo las vacías, lo cual provoca que el delimitador utilizado como etiqueta no sea exactamente igual al encontrado por el detector, pero sí dentro de rangos aceptables. Cabe destacar que el valor de esta métrica alcanza sus valores mínimos para la evaluación del sub-conjunto Test B que, como se explicó en el apartado anterior, correspondía solo a imágenes capturadas durante la noche. Lo anterior es lo esperado, ya que, si durante el día ya es difícil distinguir el delimitador entre cada plaza, por la noche esta visibilidad es aún menor, incluso para el ojo humano. Es por ello que se espera que la métrica arroje un menor valor para estas condiciones ambientales.

Por otro lado, como se observó en la sección 4.1.3, las clases se encuentran desbalanceadas, existiendo una mayor cantidad de etiquetas para la clase "free", por lo que resulta importante observar el desempeño obtenido por cada una de ellas. Para ello, se realizó una matriz de confusión por cada test evaluado y para cada experimento, resultando así una cantidad de 9 matrices de confusión, las cuales pueden observarse en las figuras 5.2, 5.3 y 5.4.

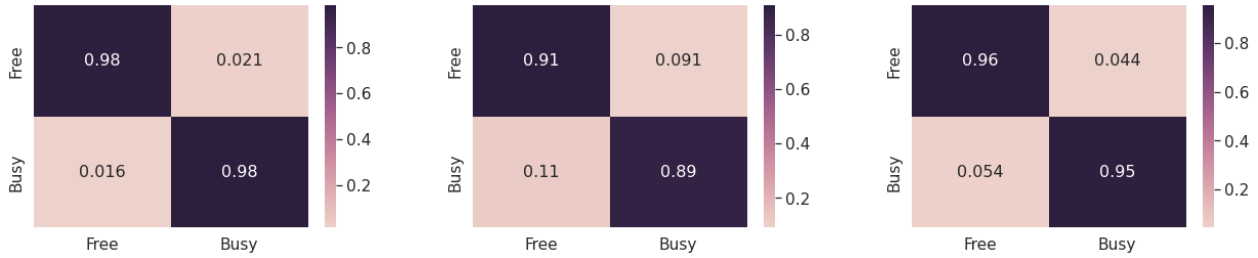
En primer lugar, se tienen los resultados correspondientes a las matrices de confusión utilizando los pesos correspondientes al entrenamiento del experimento 1, mostrados en la figura 5.2. En ellos se puede observar que en todos los subconjutnos el mejor resultado en la detección es para la clase "free", lo cual es esperado en sentido que a la red se le otorgaron más muestras de esta clase. Cabe destacar que, al comparar los resultados obtenidos por los subconjuntos, la menor precisión se obtuvo en los sub-dataset *Test B* y *Test C*, que coincide con lo esperado, ya que esta red solo se entrenó para imágenes capturadas durante el día, y los subconjuntos *Test B* y *Test C* presentaban imágenes capturadas durante la noche, por lo cual se disminuye su precisión al evaluar esas muestras.



(a) Sub-conjunto Test A (b) Sub-conjunto Test B (c) Sub-conjunto Test C

Figura 5.2: Matriz de confusión para cada sub-conjunto test, utilizando los pesos de entrenamiento correspondientes al experimento 1.

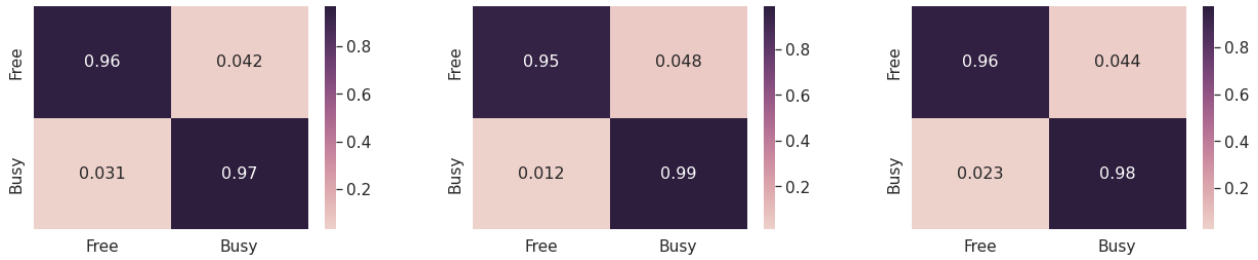
En el caso de los resultados del experimento 2, mostrado en la figura 5.3, se obtienen resultados similares en cuanto a un mejor resultado para la clase "free", aunque con una leve mejora para los sub-conjuntos *Test B* y *Test C*, lo cual tiene relación con el hecho de que se utilizó un conjunto de imágenes con condiciones ambientales nocturnas para el conjunto de validación durante el entrenamiento, mejorando la precisión del modelo.



(a) Sub-conjunto Test A (b) Sub-conjunto Test B (c) Sub-conjunto Test C

Figura 5.3: Matriz de confusión para cada sub-conjunto test, utilizando los pesos de entrenamiento correspondientes al experimento 2.

Finalmente, los resultados arrojados para el experimento 3 se pueden observar en la figura 5.4. En este caso, a diferencia de los resultados anteriores, se presenta una mayor precisión para la clase busy en cada uno de los subconjuntos de test evaluados. Lo anterior puede deberse a que, para este experimento, al incluir tanto como para el conjunto train como para el de validación imágenes capturadas durante la noche, la visualización de los delimitadores entre cada plaza es menos distintiva, lo que pudo afectar un poco el entrenamiento. A pesar de ello, el porcentaje de precisión de la clase busy por sobre la clase free es prácticamente despreciable. Cabe destacar, además, que en este caso ambas clases alcanzaron métricas bastantes altas, lo cual es esperado ya que los conjuntos durante el entrenamiento presentaban alta variabilidad.



(a) Sub-conjunto Test A (b) Sub-conjunto Test B (c) Sub-conjunto Test C

Figura 5.4: Matriz de confusión para cada sub-conjunto test, utilizando los pesos de entrenamiento correspondientes al experimento 3.

Es resumen, el experimento 3 obtuvo el mejor desempeño, lo que era de esperarse dado que, tanto como para el conjunto de entrenamiento y validación, se utilizó variabilidad en las muestras utilizadas, con capturas tomadas durante el día y la noche. Sin embargo, resulta interesante destacar que, aunque los experimentos 1 y 2 no presentaban este tipo de variabilidad en sus conjuntos, de todas formas, alcanzaron un desempeño aceptable, lo cual

es importante, dado que en la realidad no se cuenta siempre con la variabilidad esperada. Por otro lado, es interesante observar que, aunque las clases estaban desbalanceadas, la clase "busy", que era la que contaba con menor cantidad de muestras, obtuvo un buen desempeño, lo que puede deberse a que para el entrenamiento se utilizó un peso pre-entrenado de YOLO, que ya reconocía dentro de sus 80 clases a los autos. Por esta razón, es que aun cuando existían menos muestras para esta clase, el modelo ya se encontraba pre-entrenado con muestras de este tipo. El hecho anterior, también podría ser la causa de que el modelo convergiera más rápido durante el entrenamiento, ya que se logró estabilizar en todos los experimentos antes de las 1000 iteraciones, cuando empíricamente es común que los modelos que se entrenan para diferenciar dos clases, se estabilicen alrededor de las 2000 iteraciones.

Ahora bien, con respecto a la clase "free", era de vital importancia contar con más muestras, ya que era una clase totalmente nueva para la red, además de difícil diferenciar a simple vista, ya que no se contaba con delimitadores distintivos, como es habitual en estos estacionamientos que son demarcados con líneas blancas o amarillas. Sin embargo, pese a que no se contaba con tantas muestras en general, el detector logró un buen desempeño.

5.2.1. Resultados cualitativos

En las figuras 5.5, 5.6, 5.7 y 5.8 se pueden observar ejemplos de los resultados obtenidos al aplicar el algoritmo de detección a capturas de imágenes sobre el estacionamiento, ya sea con luz ambiental diurna o nocturna. Cabe destacar que en cada uno de los resultados expuestos se utilizó el peso correspondiente al entrenamiento del experimento 3, donde se obtuvieron los mejores resultados en las métricas. En cada una de las imágenes, se destacan en color morado los recuadros delimitadores que indican que la plaza está ocupada y, en color verde, los cuadros delimitadores que indican que la plaza se encuentra disponible. Además, en la parte superior derecha de cada imagen se agrega un recuadro con el número de total de plazas ocupadas con la etiqueta "free" y el total de plazas ocupadas con la etiqueta "busy".

En las figuras 5.5 y 5.6 se puede apreciar el resultado del algoritmo cuando arroja el resultado esperado, para el caso día y noche, respectivamente, detectando las 24 plazas de estacionamiento mostradas en las capturas. Cabe destacar que, en ambos casos, cuando algún vehículo presenta una dimensión muy extensa, debido a la perspectiva con que fue tomada la imagen, el vehículo abarca más de una plaza de estacionamiento. Lo anterior, ocasiona que partes del vehículo aparezcan en los delimitadores vecinos, sin embargo, pese a esto, el algoritmo marca de forma correcta el estado de estas plazas.

Figura 5.5: Ejemplo del algoritmo de detección aplicado a una captura de imagen de estacionamiento durante el día.

Figura 5.6: Ejemplo del algoritmo de detección aplicado a una captura de imagen de estacionamiento durante la noche.

En la figura 5.7 y 5.8 se pueden observar ejemplos en capturas donde el algoritmo arrojó falsos positivos a la hora de detectar plazas libres. En el caso de la figura 5.7 los falsos positivos son dos y se encuentran destacados con una circunferencia punteada de color rojo, donde se observa que para las dos últimas plazas, en vez de detectar las dos plazas, se detectaron 4. En el caso de la figura 5.7, correspondiente a un ejemplo de captura durante la noche, se puede observar un falso positivo, al igual que en el caso anterior, con respecto a la clase "free", el cual se encuentra destacado con una circunferencia punteada color naranjo. Se puede apreciar como la plaza es detectada correctamente cómo ocupada, sin embargo, a su lado izquierdo aparece un falso positivo con respecto a la clase "free".

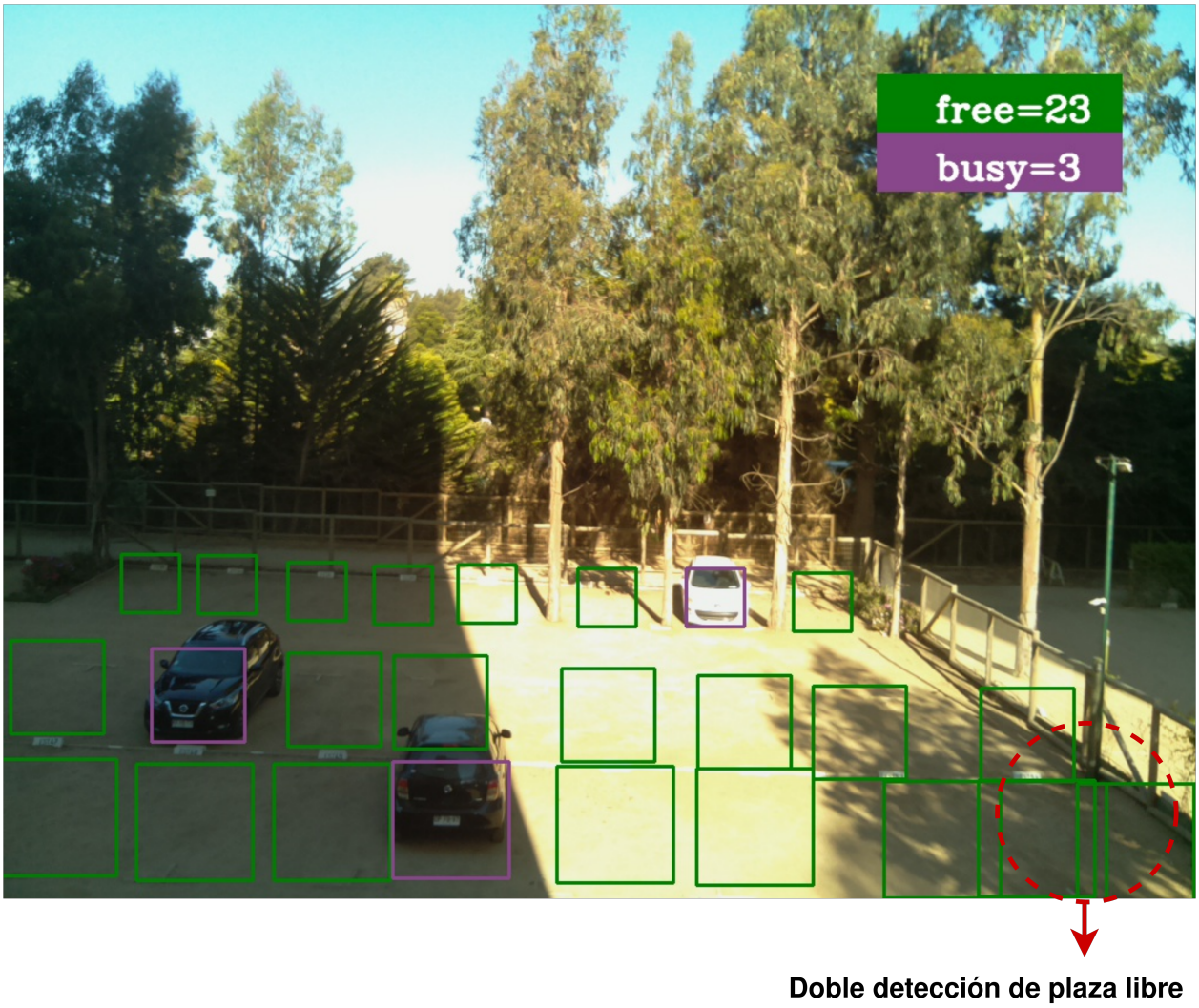
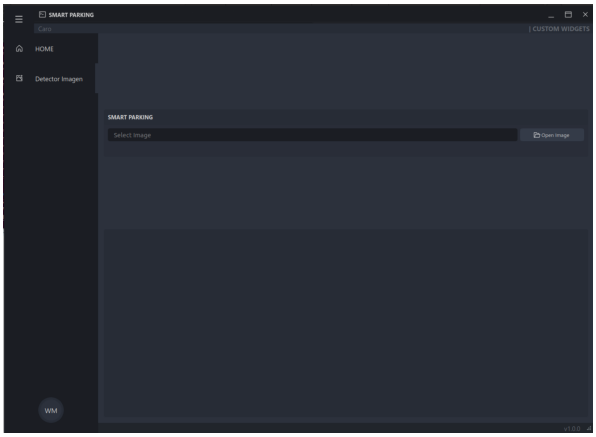


Figura 5.7: Ejemplo del algoritmo de detección aplicado a una captura de imagen de estacionamiento durante el día, con la presencia de dos falsos positivos.

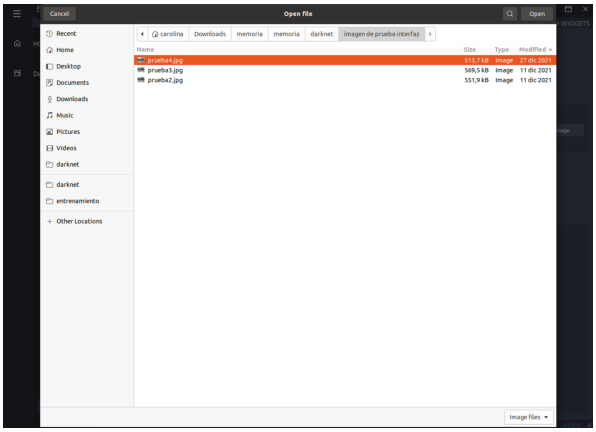
Figura 5.8: Ejemplo del algoritmo de detección aplicado a una captura de imagen de estacionamiento durante la noche, con la presencia de un falso positivo.

5.3. Visualización en interfaz

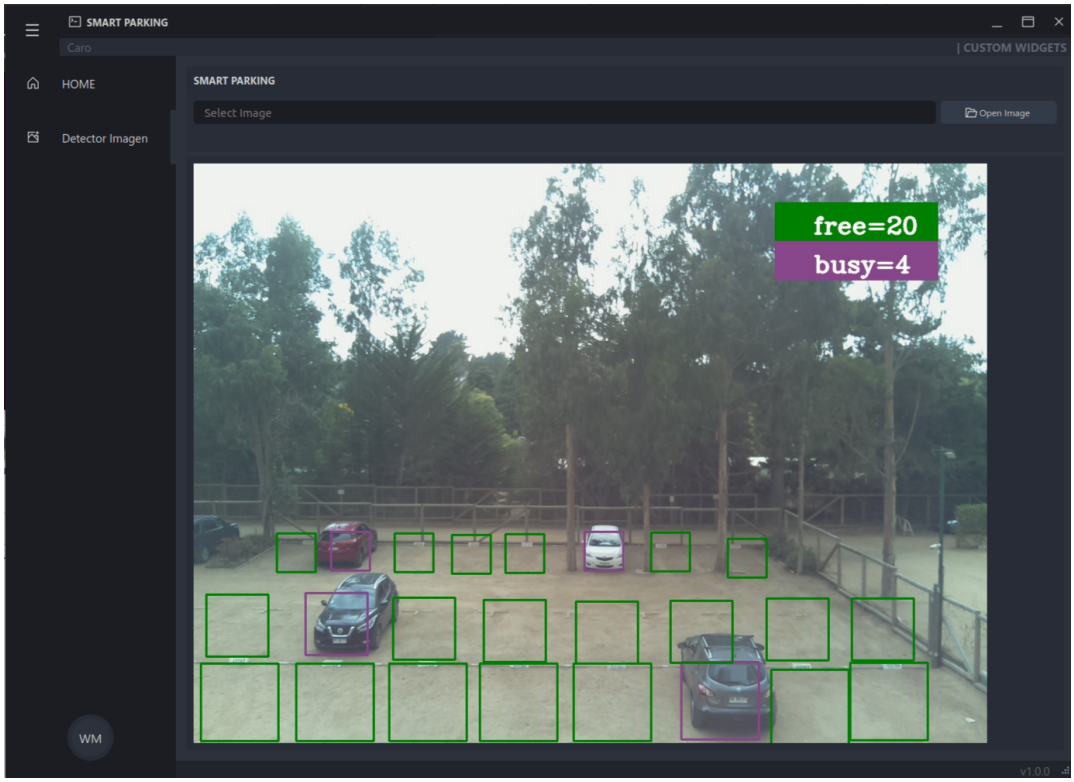
La figura 5.9 muestra el funcionamiento de la interfaz gráfica diseñada para la visualización de los resultados obtenidos por el detector. En primer lugar, en la subfigura 5.9a se puede observar un menú lateral al lado izquierdo, en el cual se encuentra un botón para la funcionalidad del detector de imagen. En el centro es posible encontrar otro botón con la etiqueta "open image" para seleccionar la imagen que será procesada. Luego en la subfigura 5.9b se puede apreciar la ventana que se despliega para seleccionar cualquier imagen en los posibles directorios. Finalmente, una vez seleccionada la imagen, el algoritmo la procesa, entregando como resultado la visualización mostrada en la subfigura 5.9c, donde se aprecia la detección de las plazas libres y ocupadas, con sus respectivos cuadros delimitadores.



(a) Menú para el detector.



(b) Selección de la imagen.



(c) Resultado de la detección de plazas ocupadas/desocupadas.

Figura 5.9: Ejemplo funcionamiento de interfaz.

5.4. Contribuciones y posibles aplicaciones

El trabajo presentado en los capítulos anteriores permite ser aplicado en un sin número de escenarios reales, desde un estacionamiento residencial hasta estacionamientos en lugares comerciales. Sin embargo, la propuesta de valor no solo se basa en la detección y conteo de plazas disponibles como tal, sino también en las posibles funcionalidades extras que pueden agregarse, logrando así una poderosa herramienta que permitirá, principalmente a los

conductores, mejorar los aspectos negativos que provoca la acción de estacionar.

Así, las funcionalidades dependerán de en qué tipo de estacionamiento será aplicada esta herramienta, las cuales se describirán a continuación:

5.4.1. Aplicaciones en sector residencial

En el caso de una zona residencial como, por ejemplo, departamentos que cuenten con su propio estacionamiento, el sistema podría contar con una aplicación en la cual los residentes accederían y tener principalmente tres funciones:

- Visualización en tiempo real del estacionamiento para acceder desde cualquier lugar.
- Un sistema de notificaciones que funcionaria a partir del algoritmo en cuestión, ya que si el usuario así lo desea, podría gestionar una inscripción que le permitiese avisar si su vehículo fue movido de su lugar como, por ejemplo a causa de un robo o cuando otro conductor hizo uso de su plaza asignada sin su consentimiento, que es un problema usual que se presenta en este tipo de estacionamientos.
- Un sistema de verificación del estado de las plazas, también mediante una aplicación, destinadas al estacionamientos para visitas.

5.4.2. Aplicaciones en espacios comerciales y públicos

En el caso de espacios comerciales y públicos, mediante el uso de una aplicación y a partir del algoritmo de detección, sería posible notificar a cualquier conductor las plazas disponibles de un estacionamiento que tenga implementado este sistema. De esta forma, el conductor tendría acceso a información que le permitiría encontrar un estacionamiento de forma rápida y eficiente, disminuyendo así aspectos negativos para el conductor, así como el estrés y la pérdida de tiempo, además de reducir los gases contaminantes innecesarios que se emanan al momento de estacionar.

Resulta interesante destacar, en el contexto de retorno a clases presenciales en la **FCFM**, que un posible lugar para el uso de este sistema sería en el estacionamiento ubicado en la calle *Tupper*, en el cual es común que estacionen los estudiantes. A su vez, también es común que se sufran robos en ese sector, por lo que un sistema que permitiera notificar cuando tu vehículo fue movido, mediante una inscripción en una aplicación, según la plaza ocupada, resultaría de gran utilidad para mejorar la confianza del conductor al dejar su vehículo estacionado.

Capítulo 6

Conclusiones y Trabajo Futuro

A partir del trabajo realizado se puede concluir que:

Los objetivos propuestos en la sección 1.3 fueron cumplidos, ya que se creó un sistema inteligente que permite la detección de 24 plazas de estacionamientos, así como como la clasificación del estado de la plaza (ocupadas/desocupadas). Lo anterior fue realizado a partir de una base de datos construida de un estacionamiento residencial, utilizando para ello un algoritmo de detección personalizado para esta problemática, cuyos resultados pueden ser visualizado en la interfaz diseñada.

Se destaca la construcción de la base datos, ya que no se había realizado en trabajos anteriores o en la literatura una base de datos que fuese hecha a partir de un estacionamiento residencial el cual, además, no presentaba marcas distintivas de separación entre cada plaza. Pese a lo anterior, el algoritmo destaca frente a otros, que ya resolvían una problemática similar, ya que permite detectar las plazas aun cuando no se presenta un delimitador destacado como en otros estacionamientos donde las separaciones son claramente demarcadas con líneas blancas o amarillas. En cambio, para este caso, aunque se presentaba un separador, este no era tan visible, sumado a que la cámara utilizada no era de tan alta resolución.

Al comparar este modelo con otros sistemas que se basan en el uso de sensores fijos, resulta necesario destacar que el modelo propuesto presenta una mayor ventaja. Esto debido a que el costo de utilizar sensores fijos es mas alto, sobretodo en estacionamientos grandes, donde se requiere la instalación de un sensor por cada plaza, así como también el costo de mantención y la vulnerabilidad que presentan frente a posibles robos.

Finalmente, a pesar de solo contar con una cámara de baja resolución, provocando una perspectiva complicada para la detección de las plazas, el algoritmo alcanzó un gran desempeño, obteniendo un alto valor en las métricas evaluadas, cuya falla más común era detectar una cantidad mínima de falsos positivos respecto a la clase “free”, cuando la imagen se veía afectada por el exceso de luz durante el día, o baja intensidad de luz durante la noche debido a la sombra provocada por los vehículos.

6.1. Trabajo futuro

Como continuación de este proyecto, y para mejorar el modelo obtenido, se propone:

- El uso de una cámara de mayor resolución y adaptable de forma automática a los niveles de luz del entorno. En caso de no existir esa posibilidad, se recomienda el uso de un sensor de luz, de modo tal de cambiar a nivel de software los parámetros de la cámara, según las luz capturada por el sensor y no con un horario fijo como se hizo en este proyecto. De esta forma la captura de imágenes se calibra de forma óptima según el día y la noche, según las estaciones del año.
- La captura de imagen para diferentes épocas del año, de modo de obtener mayor variabilidad al momento de realizar el entrenamiento, sobre todo en la época de invierno, donde es usual obtener imágenes borrosas producto del lente cubierto por el agua de lluvia.
- Finalmente, resulta necesario realizar una interfaz que sea de fácil acceso al usuario y presente una conexión online, de tal forma de poder visualizar el estacionamiento en tiempo real de forma remota. Además, implementar funciones tales como un sistema de notificaciones que avise al usuario cuando se ocupa/desocupa la plaza asignada a su departamento si el así lo desea. Lo anterior permite agregar una propuesta de valor para el caso de estacionamientos residenciales, donde es usual que otro vehículo ocupe tu plaza asignada sin autorización. De esta forma, este nuevo sistema permitiría alertar de forma más eficiente al residente si su plaza fue ocupada o alertar en caso de robo de su vehículo.

Bibliografía

[1] *Estudio realizado por IBM. Disponible en:* https://www-03.ibm.com/press/us/en/pressrelease/35515.wss.

[2] *J. Lin, S. Chen, C. Chang and G. Chen, "SPA: Smart Parking Algorithm Based on Driver Behavior and Parking Traffic Predictions,ïn IEEE Access, vol. 7, pp. 34275-34288, 2019, doi: 10.1109/ACCESS.2019.2904972.*

[3] *Parque de Vehículos en Chile, disponible en :* https://www.ine.cl/estadisticas/economia/transporte-y-comunicaciones/permiso-de-circulacion/parque-de-vehiculos.

[4] *Salman Khan, Syed Ali Shah, Hossein Rahmani, Gerard Medioni, Mohammed Bennamoun, .ª Guide to Convolutional Neural Networks for Computer Vision", Published by Morgan Claypool Publishers, 2018.*

[5] *Adam Gibson; Josh Patterson, "Deep Learning", Published by O'Reilly Media, Inc., 2017.*

[6] *Definición de detector de objetos y YOLO V4 disponible en:* https://medium.com/aiguys/yolo-v4-explained-in-full-detail-5200b77aa825.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[8] *Algoritmo YOLO:* https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246.

[9] *Repositorio YOLO disponible en:* https://github.com/AlexeyAB/darknet.

[10] *Definición métricas:* https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1.

[11] *C. Roman, R. Liao, P. Ball, S. Ou and M. de Heaver, "Detecting On-Street Parking Spaces in Smart Cities: Performance Evaluation of Fixed and Mobile Sensing Systems,ïn IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 7, pp. 2234-2245, July 2018, doi: 10.1109/TITS.2018.2804169.*

[12] *G. Amato, F. Carrara, F. Falchi, C. Gennaro and C. Vairo, Çar parking occu-*

pancy detection using smart camera networks and Deep Learning,"2016 IEEE Symposium on Computers and Communication (ISCC), Messina, 2016, pp. 1212-1217, doi: 10.1109/ISCC.2016.7543901.

[13] *A. Krizhevsky, I. Sutskever and G. E. Hinton, Ïmagenet classification with deep convolutional neural networks", Advances in neural information processing systems, pp. 1097-1105, 2012.*

[14] *Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.*

[15] *Base de Datos Find a Car Park. Disponible en:* https://www.kaggle.com/daggysheep/find-a-car-park.

[16] *Base de Datos CNRPark+EXT. Disponible en:* http://cnrpark.it/.

[17] *Base de Datos PKLot. Disponible en:* https://web.inf.ufpr.br/vri/databases/parking-lot-database/.

[18] *Empresa Meste. Pagina Web Disponible en :* http://www.meste.cl/Parking/Casos_de_Exito/Kidzania.

[19] *Empresa Parkassist. Pagina Web Disponible en :* https://www.parkassist.com/es/solutions/wayfinding-signage.

[20] *Empresa Urbiotica. Pagina Web Disponible en :* https://www.urbiotica.com/ejemplos-smart-cities/estacionamiento-inteligente-las-condes-chile/.

[21] *Programa para asignación de etiqueta LabelImg disponible en:* https://github.com/tzutalin/labelImg.

[22] *Pesos YOLO V4:* https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137.

[23] *Template interfaz disponible en:* https://github.com/Wanderson-Magalhaes/Simple_PySide_Base.

[24] *Sistema Operativo Jetson Nano Nvidia disponible en:* https://developer.nvidia.com/embedded/jetpack.

Anexo A

Detalle de Hardware Utilizado

Producto	Valor	Proveedor
Raspberry pi 4B	$69.990	raspberry
Camara Rev 1.3 MP	$7.990	camara
Jetson Nano Nvidia	$157.990	jetson nano nvidia
Camara V2 8MP	$39.990	camara v2
Tarjeta de Memoria 64GB	$12.590	tarjeta de memoria
Transformador Raspberry Pi	$7.990	transformador
Fuente de poder	$18.990	fuente de poder

Tabla A.1: Especificaciones y proveedores del hardware.

Anexo B

Códigos

B.1. Código en python para captura de imágenes

```
import time
import picamera
import datetime
from fractions import Fraction

while True:
    x = datetime.datetime.now()
    if 7<x.hour<19:
        with picamera.PiCamera() as camera:
            camera.start_preview()
            time.sleep(2)
            date_time = x.strftime("%m-%d-%Y-%H:%M:%S")
            camera.capture('./data_dia/img'+date_time+'.jpg')
            time.sleep(300)
    else:
        with picamera.PiCamera() as camera2:
            camera2.framerate = Fraction(1, 6)
            camera2.shutter_speed = 6000000
            camera2.exposure_mode = 'off'
            camera2.iso = 800
            time.sleep(10)
            date_time = x.strftime("%m-%d-%Y-%H:%M:%S")
            camera2.capture('./data_noche/img'+date_time+'.jpg')
            time.sleep(300)
```

B.2. Makefile

```
%%writefile Makefile
GPU=1
CUDNN=1
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=1
```

```
ZED_CAMERA=0
ZED_CAMERA_v2_8=0


USE_CPP=0
DEBUG=0

ARCH= -gencode arch=compute_35,code=sm_35 \
      -gencode arch=compute_50,code=[sm_50,compute_50] \
      -gencode arch=compute_52,code=[sm_52,compute_52] \
      -gencode arch=compute_61,code=[sm_61,compute_61] \
      -gencode arch=compute_37,code=sm_37

ARCH= -gencode arch=compute_60,code=sm_60

OS := $(shell uname)

VPATH=./src/
EXEC=darknet
OBJDIR=./obj/

ifeq ($(LIBSO), 1)
LIBNAMESO=libdarknet.so
APPNAMESO=uselib
endif

ifeq ($(USE_CPP), 1)
CC=g++
else
CC=gcc
endif

CPP=g++ -std=c++11
NVCC=nvcc
OPTS=-Ofast
LDFLAGS= -lm -pthread
COMMON= -Iinclude/ -I3rdparty/stb/include
CFLAGS=-Wall -Wfatal-errors -Wno-unused-result -Wno-unknown-pragmas -fPIC

ifeq ($(DEBUG), 1)
#OPTS= -O0 -g
#OPTS= -Og -g
COMMON+= -DDEBUG
CFLAGS+= -DDEBUG
else
ifeq ($(AVX), 1)
CFLAGS+= -ffp-contract=fast -mavx -mavx2 -msse3 -msse4.1 -msse4.2 -msse4a
endif
endif

CFLAGS+=$(OPTS)

ifneq (,$(findstring MSYS_NT,$(OS)))
LDFLAGS+=-lws2_32
endif

```

```
ifeq ($(OPENCV), 1)
COMMON+= -DOPENCV
CFLAGS+= -DOPENCV
LDFLAGS+= `pkg-config --libs opencv4 2> /dev/null || pkg-config --libs
    opencv`
COMMON+= `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags
    opencv`
endif

ifeq ($(OPENMP), 1)
CFLAGS+= -fopenmp
LDFLAGS+= -lgomp
endif

ifeq ($(GPU), 1)
COMMON+= -DGPU -I/usr/local/cuda/include/
CFLAGS+= -DGPU
ifeq ($(OS),Darwin) #MAC
LDFLAGS+= -L/usr/local/cuda/lib -lcuda -lcudart -lcublas -lcurand
else
LDFLAGS+= -L/usr/local/cuda/lib64 -lcuda -lcudart -lcublas -lcurand
endif
endif

ifeq ($(CUDNN), 1)
COMMON+= -DCUDNN
ifeq ($(OS),Darwin) #MAC
CFLAGS+= -DCUDNN -I/usr/local/cuda/include
LDFLAGS+= -L/usr/local/cuda/lib -lcudnn
else
CFLAGS+= -DCUDNN -I/usr/local/cudnn/include
LDFLAGS+= -L/usr/local/cudnn/lib64 -lcudnn
endif
endif

ifeq ($(CUDNN_HALF), 1)
COMMON+= -DCUDNN_HALF
CFLAGS+= -DCUDNN_HALF
ARCH+= -gencode arch=compute_70,code=[sm_70,compute_70]
endif

ifeq ($(ZED_CAMERA), 1)
CFLAGS+= -DZED_STEREO -I/usr/local/zed/include
ifeq ($(ZED_CAMERA_v2_8), 1)
LDFLAGS+= -L/usr/local/zed/lib -lsl_core -lsl_input -lsl_zed
#-lstdc++ -D_GLIBCXX_USE_CXX11_ABI=0
else
LDFLAGS+= -L/usr/local/zed/lib -lsl_zed
#-lstdc++ -D_GLIBCXX_USE_CXX11_ABI=0
endif
endif

OBJ=image_opencv.o http_stream.o gemm.o utils.o dark_cuda.o
    convolutional_layer.o list.o image.o activations.o im2col.o col2im.o
    blas.o crop_layer.o dropout_layer.o maxpool_layer.o softmax_layer.o
    data.o matrix.o network.o connected_layer.o cost_layer.o parser.o
```

```
    option_list.o darknet.o detection_layer.o captcha.o route_layer.o
    writing.o box.o nightmare.o normalization_layer.o avgpool_layer.o coco.
    o dice.o yolo.o detector.o layer.o compare.o classifier.o local_layer.o
     swag.o shortcut_layer.o activation_layer.o rnn_layer.o gru_layer.o rnn
    .o rnn_vid.o crnn_layer.o demo.o tag.o cifar.o go.o batchnorm_layer.o
    art.o region_layer.o reorg_layer.o reorg_old_layer.o super.o voxel.o
    tree.o yolo_layer.o gaussian_yolo_layer.o upsample_layer.o lstm_layer.o
     conv_lstm_layer.o scale_channels_layer.o sam_layer.o
ifeq ($(GPU), 1)
LDFLAGS+= -lstdc++
OBJ+=convolutional_kernels.o activation_kernels.o im2col_kernels.o
    col2im_kernels.o blas_kernels.o crop_layer_kernels.o
    dropout_layer_kernels.o maxpool_layer_kernels.o network_kernels.o
    avgpool_layer_kernels.o
endif

OBJS = $(addprefix $(OBJDIR), $(OBJ))
DEPS = $(wildcard src/*.h) Makefile include/darknet.h

all: $(OBJDIR) backup results setchmod $(EXEC) $(LIBNAMESO) $(APPNAMESO)

ifeq ($(LIBSO), 1)
CFLAGS+= -fPIC

$(LIBNAMESO): $(OBJDIR) $(OBJS) include/yolo_v2_class.hpp src/
    yolo_v2_class.cpp
  $(CPP) -shared -std=c++11 -fvisibility=hidden -DLIB_EXPORTS $(COMMON) $(
    CFLAGS) $(OBJS) src/yolo_v2_class.cpp -o $@ $(LDFLAGS)

$(APPNAMESO): $(LIBNAMESO) include/yolo_v2_class.hpp src/yolo_console_dll.
    cpp
  $(CPP) -std=c++11 $(COMMON) $(CFLAGS) -o $@ src/yolo_console_dll.cpp $(
    LDFLAGS) -L ./ -l:$(LIBNAMESO)
endif

$(EXEC): $(OBJS)
  $(CPP) -std=c++11 $(COMMON) $(CFLAGS) $^ -o $@ $(LDFLAGS)

$(OBJDIR)%.o: %.c $(DEPS)
  $(CC) $(COMMON) $(CFLAGS) -c $< -o $@

$(OBJDIR)%.o: %.cpp $(DEPS)
  $(CPP) -std=c++11 $(COMMON) $(CFLAGS) -c $< -o $@

$(OBJDIR)%.o: %.cu $(DEPS)
  $(NVCC) $(ARCH) $(COMMON) --compiler-options "$(CFLAGS)" -c $< -o $@

$(OBJDIR):
  mkdir -p $(OBJDIR)
backup:
  mkdir -p backup
results:
  mkdir -p results
setchmod:
  chmod +x *.sh

```

```
.PHONY: clean

clean:
	rm -rf $(OBJS) $(EXEC) $(LIBNAMESO) $(APPNAMESO)
```

B.3. Código configuración arquitectura de la red

```

def file_len(fname):
  with open(fname) as f:
    for i, l in enumerate(f):
      pass
  return i + 1

num_classes = file_len('train/_darknet.labels')
print("N mero de clases: " + str(num_classes))

if os.path.exists('./cfg/custom-yolov4-detector.cfg'): os.remove('./cfg/
    custom-yolov4-detector.cfg')


with open('./cfg/custom-yolov4-detector.cfg', 'a') as f:
  f.write('[net]' + '\n')
  f.write('batch=64' + '\n')
  f.write('subdivisions=24' + '\n')
  f.write('width=416' + '\n')
  f.write('height=416' + '\n')
  f.write('channels=3' + '\n')
  f.write('momentum=0.949' + '\n')
  f.write('decay=0.0005' + '\n')
  f.write('angle=0' + '\n')
  f.write('saturation = 1.5' + '\n')
  f.write('exposure = 1.5' + '\n')
  f.write('hue = .1' + '\n')
  f.write('\n')
  f.write('learning_rate=0.001' + '\n')
  f.write('burn_in=1000' + '\n')
  #f.write('burn_in=200' + '\n')
  max_batches = num_classes*2000
  f.write('max_batches=' + str(max_batches) + '\n')
  f.write('policy=steps' + '\n')
  steps1 = .8 * max_batches
  steps2 = .9 * max_batches
  f.write('steps='+str(steps1)+','+str(steps2) + '\n')



  with open('cfg/yolov4-custom2.cfg', 'r') as f2:
    content = f2.readlines()
    for line in content:
      f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
```

```
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 0,1,2' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146,
   142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

  with open('cfg/yolov4-custom3.cfg', 'r') as f3:
    content = f3.readlines()
    for line in content:
      f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 3,4,5' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146,
   142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

  with open('cfg/yolov4-custom4.cfg', 'r') as f4:
    content = f4.readlines()
    for line in content:
      f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 6,7,8' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146,
   142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

  with open('cfg/yolov4-custom5.cfg', 'r') as f5:
    content = f5.readlines()
    for line in content:
      f.write(line)
```