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

# IMPLEMENTACIÓN DE UN SISTEMA DE DETECCIÓN DE RESIDUOS RECICLABLES BASADO EN VISIÓN COMPUTACIONAL

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

**MATÍAS SEBASTIÁN SAAVEDRA MAYORGA**

PROFESOR GUÍA:
ANDRÉS CABA RUTTE

MIEMBROS DE LA COMISIÓN:
FRANCISCO CASADO CASTRO
PABLO GONZÁLEZ ORDÓÑEZ

Este trabajo ha sido parcialmente financiado por:
BEAUCHEF PROYECTA
ODD INDUSTRIES

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: **MATÍAS SEBASTIÁN SAAVEDRA MAYORGA**
FECHA: 2020
PROF. GUÍA: ANDRÉS CABA RUTTE

# IMPLEMENTACIÓN DE UN SISTEMA DE DETECCIÓN DE RESIDUOS RECICLABLES BASADO EN VISIÓN COMPUTACIONAL

Chile es el mayor productor de residuos sólidos municipales (RSM) de toda América Latina y el Caribe, produciendo 1,25 kg/día por habitante. Al 2016, la mayor parte de los RSM acabaron en rellenos sanitarios. Desde el 2015 existe en Chile la "ley de fomento al reciclaje" (ley 20.920) que establece metas de recolección y valorización incrementales para 6 productos prioritarios. Uno de ellos es "envases y embalajes", compuestos por: vidrios, plásticos, papeles y cartones, metales y cartón para bebidas. En Chile existe actualmente una industria recicladora para gran parte de estos productos y hay reciclaje del tipo *Single Stream* donde llegan materiales reciclables a las plantas y se separan con el uso de cintas transportadoras y uso intensivo de mano de obra.

En este trabajo se implementa un sistema de detección de residuos reciclables a través del uso de una red neuronal convolucional con el objetivo de ser la parte de identificación y localización de un separador automático de materiales reciclables en las plantas de reciclaje. Dada la aplicación, el sistema funciona en un dispositivo embebido u *on the edge* y es capaz de detectar cuatro tipos de materiales reciclables presentes en los residuos domiciliarios: vidrio, PET, tetrapak y aluminio a una tasa promedio de 9,47 *frames* por segundo.

La metodología permite resolver el problema planteado de principio a fin. Desde el diseño del montaje y la adquisición de imágenes, hasta la implementación en el dispositivo *on the edge*. Se escoge como dispositivo la NVIDIA Jetson Nano, debido a su bajo coste y compatibilidad con *frameworks* de *deep learning*. Dada la capacidad de la Jetson Nano y que la aplicación en una cinta transportadora implica que debe funcionar a una alta velocidad de inferencia, se escoge como red a entrenar SSD MobileNetV2.

Antes de entrenar la red es necesario a crear una base de datos de imágenes de materiales reciclables etiquetadas con *bounding boxes* pues no existe una base de datos específica para este problema. Las imágenes son etiquetadas o reetiquetadas en la plataforma online Labelbox. Para entrenar se utiliza el *framework* abierto *Tensorflow Object Detection API* y un modelo preentrenado en COCO. Para realizar el entrenamiento de se utiliza una máquina virtual de Google Cloud Platform.

Los resultados del trabajo muestran que se detectan los materiales en la simulación experimental del montaje con un *mean average precision* (mAP) de 0,53. Al revisar en profundidad los datos de entrenamiento se observa que hay errores en las marcas, tales como marcas falsas, objetos altamente ocluidos y marcas de grupos como un solo elemento. Estos resultados muestran que se puede realizar detección de materiales reciclables *on the edge* y que hay un amplio margen para seguir mejorando el modelo a través de la corrección y la obtención de más datos.

*Para mi abuela Amelia.*

# Tabla de Contenido

# Índice de Tablas

# Índice de Figuras

# Capítulo 1

# Introducción

## 1.1. Motivación y Antecedentes

Chile es el mayor productor de residuos sólidos municipales (RSM) de toda América Latina y el Caribe, produciendo 1,25 kg/día por habitante, siendo el promedio de la región 0,93 kg/día por habitante en el año 2010 [1].

Al 2016, la mayor parte de los residuos acabó en rellenos sanitarios, ya que solo el 23,6 % de los residuos no peligrosos se valorizaron, es decir, se reutilizaron, se reciclaron o se aprovechó su poder calorífico [2]. Para el caso de los RSM, que corresponden al 35,3 % de los residuos no peligrosos, la tasa de valorización es de tan solo un 1,5 % [3]. En la Figura 1.1 se muestra la valorización de residuos no peligrosos en Chile desglosada por región, dónde la que más valoriza es la Región Metropolitana, que concentra la mayor cantidad de habitantes.

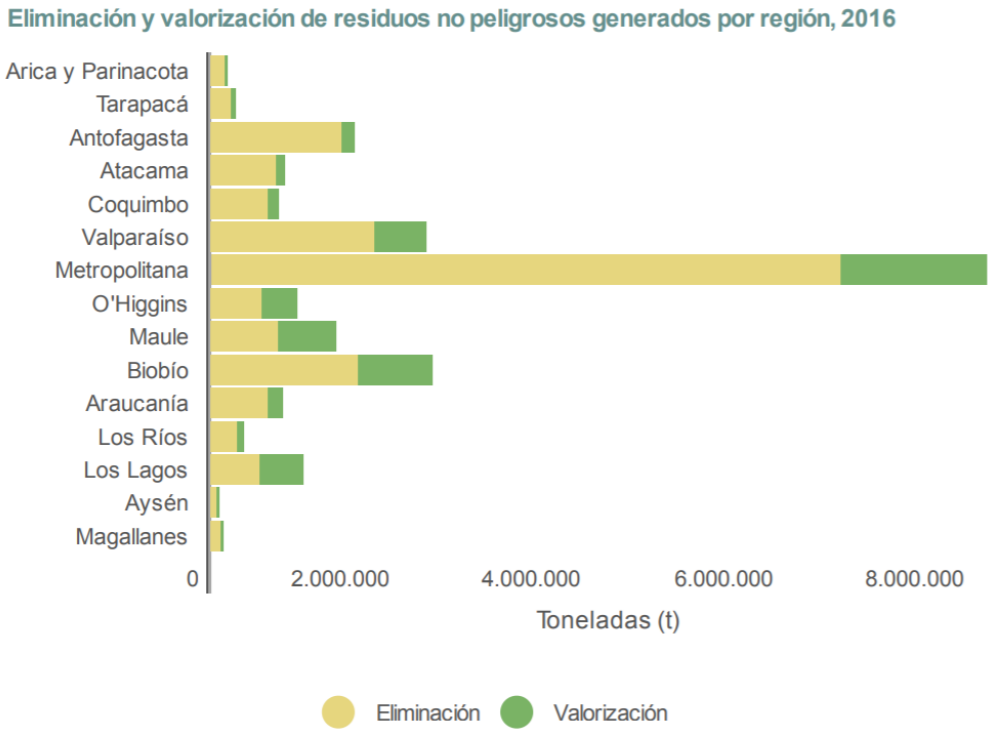


Figura 1.1: Valorización residuos no peligrosos por región [3]

En la Tabla 1.1 se muestra la composición de los residuos sólidos municipales que se produjeron en Chile el año 2009. De los RSM valorizados ese año, el 51 % se hizo a través de compostaje y el 43 % a través de reciclaje [4]. Este trabajo trata solo sobre los materiales recuperables que no pertenecen a materia orgánica ni a textiles, esto porque la manipulación de los primeros es más compleja pues son de rápida descomposición, y los últimos porque representan el menor porcentaje de composición.

Tabla 1.1: Composición de los residuos sólidos municipales generados en 2009.

| **Tipo de residuo** | **Composición [%]** |
|---|---|
| Materia orgánica | 53,3 |
| Papeles y cartones | 12,4 |
| Plásticos | 9,4 |
| Vidrios | 6,6 |
| Metales | 2,3 |
| Textiles | 2,0 |
| Otros | 14,0 |

Para gran parte de estos materiales recuperados existe una industria de procesadoras nacionales asociadas, y solo en el caso de las latas de aluminio, dado que no existe una, se exportan. Para los plásticos se encuentran empresas procesadoras como RECIPET, RECIPLAS, CAMBIASO, entre otras. Para el vidrio se encuentra CRISTALERÍAS TORO, CRISTALERÍAS CHILE y VERALLIA. Para papeles y cartones PAPELES CORDILLERA, PAPELERA DEL PACÍFICO, PAIMASA, FORESTAL Y PAPELERA CONCEPCIÓN y T-PAC [5].

La importancia del reciclaje en Chile ha ido en aumento, a tal punto que desde el 2015 existe una "ley de fomento al reciclaje" (ley 20.920) que establece metas de recolección y valorización para 6 productos prioritarios, estos son: aceites lubricantes, aparatos eléctricos y electrónicos, baterías, pilas, neumáticos y, envases y embalajes [2]. Sobre esta última categoría, el año 2019 se promulgó el anteproyecto del decreto que establece metas incrementales desde el año 2022 hasta el 2030. Estas metas se pueden ver en la infografía de la Figura 1.2 realizada por Pulso del diario La Tercera[1].

[1] La Tercera - "Ley REP obligará a las empresas a reciclar casi la mitad de los envases plásticos a 2030"

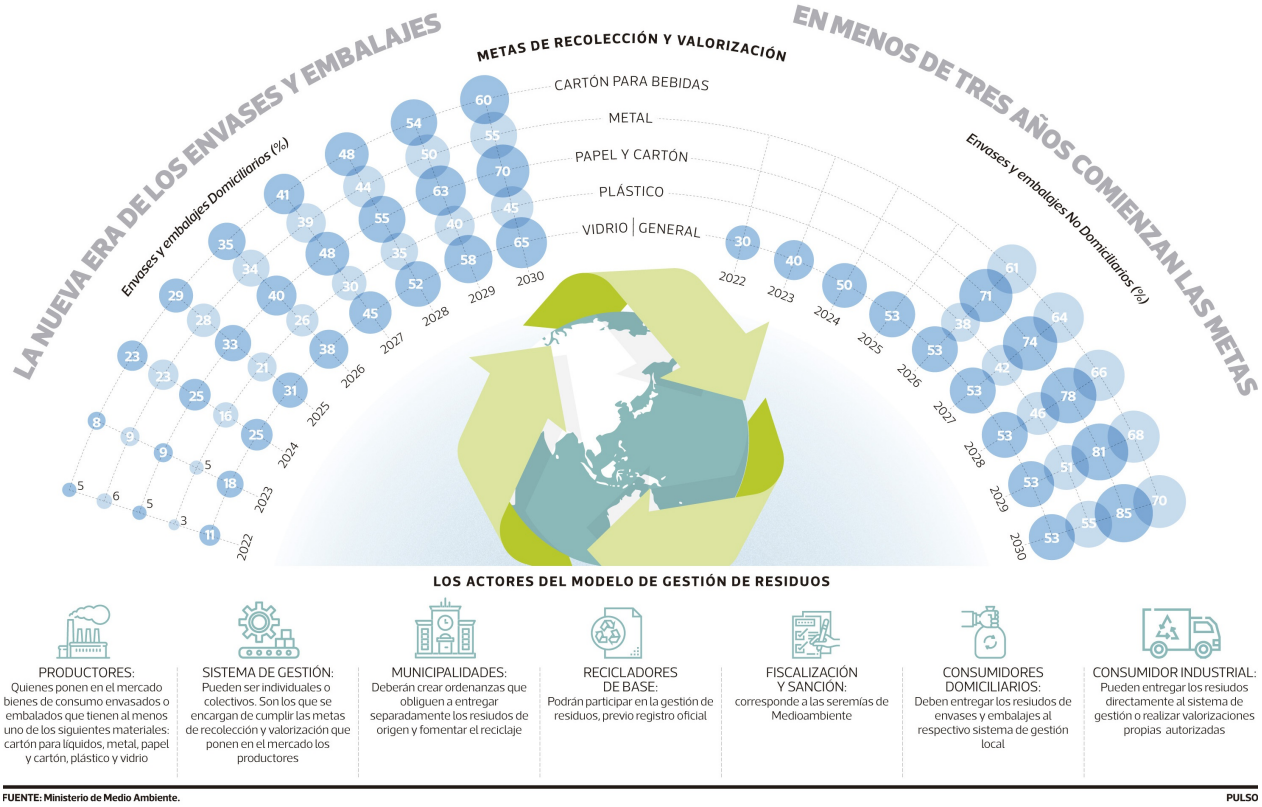


Figura 1.2: Infografía sobre las metas de envases y embalajes de la ley de fomento al reciclaje elaborado por el diario La Tercera. Al año 2022 se debe valorizar el 5 % de los cartones para bebidas (tetrapak), el 6 % de los metales como hojalata y aluminio, el 5 % de los papeles y cartones, el 3 % de los distintos tipos plásticos y el 11 % del vidrio. Estos porcentajes aumentan al año 2030 a 60 %, 55 %, 70 %, 45 % y 65 %, respectivamente.

Por otra parte, en distintas municipalidades de Estados Unidos existen programas de reciclaje llamados *Single Stream*, es decir, de un solo flujo, donde todos los materiales reciclables se mezclan en el camión recolector y llegan juntos a la planta de procesamiento, donde se separan a través de cintas transportadoras en base a distintas técnicas.

En Chile existe el caso de la planta de reciclaje de KDM Tratamiento, donde en su página web mencionan que la planta "combina un tratamiento de selección manual y automatizada" y "contempla un uso intensivo de mano de obra, principalmente en la etapa de recuperación manual de materiales reciclables". Además, existen otros puntos limpios o centros de reciclaje que se denominan *Multi Stream*, es decir, con múltiples flujos, donde los materiales ya llegan separados y simplemente se realiza un proceso de control de calidad, donde se revisa que los materiales sean efectivamente los correctos, para después acopiarlos, comprimirlos, si da el caso, y luego enviar un gran volumen a las plantas correspondientes.

## 1.2. Descripción del problema y diseño de solución

Las plantas de reciclaje o *material recovery facility* (MRF) suelen utilizar múltiples sensores para separar los materiales. Estos sensores pueden ser basados en propiedades físicas

del material, cómo imanes y sensores inductivos para separar materiales ferrosos y aluminio, o sensores cercanos al infrarrojo (NIR) para la separación de distintos tipos de plásticos. También pueden ser basados en el tamaño y forma del material, como los trómeles que separan cartones y elementos grandes de los más pequeños, o separadores balísticos que separan elementos 2D (papeles), finos (arena, restos de comida) y 3D (latas, botellas, etc). Sin embargo en las plantas también puede haber un uso intensivo de mano de obra tanto para la separación manual de residuos como para el control de calidad de los mismos [6].

Este uso de mano de obra se ha automatizado en los últimos años a través del uso de robots inteligentes de distintas empresas que detectan los materiales utilizando técnicas de visión computacional y luego lo separan con el uso de brazos con succión o agarre. Algunas de estos son: *AMP Cortex* de la empresa *AMP Robotics*, *Max-AI* de las empresas *Bulk Handling Systems (BHS)* con *NRT*, *Fast Picker* de *Zen Robotics*, y *SamurAI* de *Machinex* [7].

Sin embargo, los datos que usan estos robots no son libres, sino que de uso privado de cada empresa. Además, puesto que existen pocas bases de datos de imágenes para la detección de materiales reciclables, o estas son muy pequeñas, se hace difícil crear una aplicación abierta o de menor costo que permita a ciudades o comunas pequeñas valorizar sus residuos.

Se plantea como solución de la falta de automatización, el uso de una máquina que a través de visión computacional pueda identificar y separar automáticamente los distintos materiales que llegan a las plantas de reciclaje. Este trabajo soluciona la parte de identificación y localización de los materiales. Para esto es necesario desarrollar un sistema que pueda detectar (o segmentar) los materiales que pasan por la cinta en tiempo real, puesto que las cintas están en continuo movimiento y la separación ocurre inmediatamente después de la detección.

Esto limita la solución del problema a sistemas cuyo procesamiento sea local o *on the edge*, ya que el procesamiento en la nube presenta latencias más altas, lo que no permite un ciclo de inferencia en tiempo real [8]. La tasa a la que debe funcionar el sistema dependerá de la velocidad a la que funciona la cinta, mientras mayor es la velocidad de la cinta, más rápida debe ser la inferencia.

Respecto a bases de datos abiertas de imágenes de elementos reciclables, lo más cercano son imágenes para la clasificación de objetos, que es un problema más sencillo que la detección, ya que solo se identifica que objetos hay en una imagen y no en que parte de la imagen están. Uno de estos *datasets* es *Trashnet* [9], que contiene 2.527 imágenes de seis clases: 501 imágenes de vidrio, 594 de papel, 403 de cartón, 482 de plástico, 410 de metal, y 137 de otros residuos. También se encuentra disponible de manera abierta *Waste Classification Data* de *Kaggle* [2] que consta de más de 25.000 imágenes separadas en dos clases: orgánicos y reciclables. Debido a lo anterior, es necesario crear una base de datos de detección para llenar el vacío que existe. Para que una base de datos pueda servir para un problema de detección, los elementos que se quieren detectar deben estar rodeados por una caja que se conoce como *bounding box*, otra alternativa es rodear al objeto con un polígono.

Para localizar y clasificar los elementos se decide utilizar la detección de objetos por sobre

[2] https://www.kaggle.com/techsash/waste-classification-data/

la segmentación de instancias por, principalmente, dos razones. Primero, la segmentación de instancias es una tarea más compleja que la detección de objetos, lo que implica que el tiempo de inferencia es más alto y pocas redes logran inferencias en tiempo real [10]. Segundo, la creación de bases de datos de segmentación también toma un tiempo mucho mayor, llegando a demorar 315 segundos por cada elemento versus los 34,5 segundos que toma marcar un objeto para la detección, incluyendo el tiempo de revisión [11].

## 1.3. Objetivos

### 1.3.1. Objetivo General

Detectar materiales reciclables a través de la implementación de una red neuronal convolucional para su uso en una máquina que separe elementos reciclables en una cinta transportadora.

### 1.3.2. Objetivos Específicos

Los objetivos específicos que conforman el objetivo general son:

1. Crear una base de datos de imágenes de materiales reciclables, etiquetándolas con su respectiva clase y *bounding box* para poder entrenar redes neuronales de detección de objetos.
2. Entrenar una red neuronal con la base de datos generada.
3. Implementar una red neuronal en un sistema *edge*.
4. Diseñar un montaje que simule las cintas transportadoras para la evaluación del desempeño del sistema.

## 1.4. Estructura de la Memoria

Este trabajo está dividido en 6 capítulos, a continuación se describen brevemente.

El capítulo 2 presenta el marco teórico y se divide en dos grandes secciones, "Redes neuronales y detección de objetos" y "Bases de datos de detección de objetos". En la primera se explica el funcionamiento de las redes neuronales convolucionales, la detección de objetos en imágenes y la arquitectura de la red neuronal que se utiliza para resolver el problema. En la segunda sección, se muestran las bases de datos y métricas más utilizadas para la detección de objetos.

Luego, en el capítulo 3, se describe la metodología utilizada en el desarrollo del trabajo, detallando en cada paso los costos de estos, las herramientas empleadas y los problemas y soluciones encontradas.

En el capítulo 4 se muestran los resultados de la red neuronal implementada en base a

distintas métricas. En el capítulo 5 se analizan esos resultados y se realiza una exploración más profunda de los datos.

Finalmente, en el capítulo 6, se presentan las conclusiones de la memoria y se plantean posibles trabajos futuros que surgen de este trabajo.

# Capítulo 2

# Marco Teórico y Estado del Arte

## 2.1. Redes neuronales y detección de objetos

### 2.1.1. Conceptos básicos de redes neuronales

#### 2.1.1.1. Definición

Una red neuronal es un algoritmo de aprendizaje automático, es decir, que es capaz de aprender de los datos. Estas redes, a las cuales se les llama de diversas formas como *fully-connected*, *feedforward* o *multilayer perceptron (MLP)*, consisten de una colección de unidades llamadas neuronas organizadas en capas.

El objetivo de estas redes es aproximar una función $f^*$. Por ejemplo, un clasificador funciona asignando una clase $y$ a una entrada $x$ a través de la función $f^*$, tal que $y = f^*(x)$. La red define un mapa $y = f(x, \theta)$ y aprende el valor de los parámetros $\theta$, también llamados pesos, tal que resulte en la mejor aproximación de la función $f^*$ [12].

Existen tres tipos de capas: la de entrada, que corresponde a la primera capa y es la información de entrada a la red; siguen las capas ocultas, que son todas las que se encuentran entre la primera y última, y reciben la información de las neuronas de la capa anterior; finalmente, la de salida que genera la salida de la red. Para un clasificador binario, por ejemplo, la capa de salida genera una clase. En la Figura 2.1 se muestra una red neuronal de dos capas ocultas.

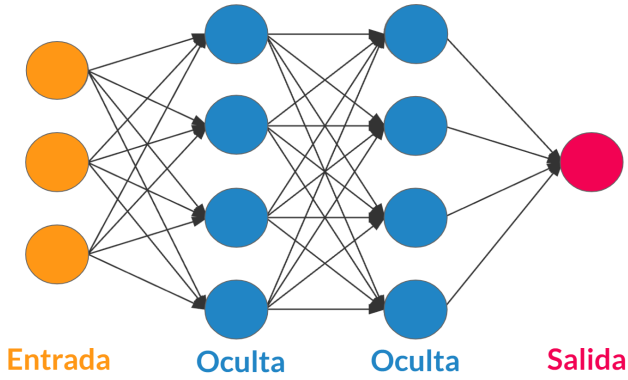


Figura 2.1: Ejemplo de MLP con dos capas ocultas.

#### 2.1.1.2. Neurona

Como se menciona, las unidades básicas de las redes neuronales son las neuronas, que son, básicamente, una función. Una neurona recibe los parámetros de entrada $x_1, x_2, ..., x_n$ y genera una salida $y$. Esto lo hace multiplicando dichas entradas por pesos $w_1, w_2, ..., w_n$, respectivamente. Luego se suman tal que $F = w_1x_1 + w_2x_2 + ... + w_nx_n + b$, sea la entrada de una función de activación no lineal, que genera la salida $y$. En la Figura 2.2 se muestra un ejemplo de una neurona o perceptrón.

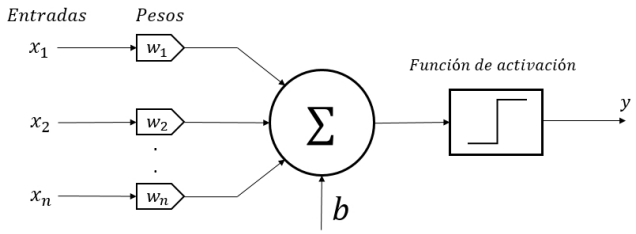


Figura 2.2: Ejemplo de neurona o perceptrón.

#### 2.1.1.3. Entrenamiento

A partir de lo anterior, una capa oculta $k$ con $m$ neuronas y que es precedida por la capa $k-1$ con $n$ neuronas, tendrá $n * m$ pesos, ya que cada neurona de la capa recibe $n$ entradas, que corresponden a las salidas de las neuronas de la capa anterior. Es a partir del ajuste de estos pesos que la red aprende.

Para aprender la red debe aproximar lo mejor posible la función $f^*$. Durante la fase de entrenamiento se actualiza el valor de los pesos con cada ejemplo que la red ve. Para hacer esto posible es necesario una función de pérdida, un método de optimización y la propagación hacia atrás del error *backpropagation*.

La actualización de pesos se hace siguiendo el método de optimización llamado descenso del gradiente, que se muestra en la ecuación (2.1). Donde $w_i$ corresponde al peso a actualizar, $\alpha$ a la tasa de aprendizaje y $L$ a la función de pérdida. A partir del descenso del gradiente nacen otros métodos de optimización ampliamente utilizados como *momentum* [13], *Adagrad* [14], *RMSProp* [15] y *Adam* [16], entre otros.

$$w_j = w_j - \alpha \frac{\partial L}{\partial w_j} \tag{2.1}$$

La función de pérdida o de costos pueden ser muchas, pero hay dos muy conocidas. Primero, el error cuadrático medio (MSE por sus sigla en inglés), que se muestra en la ecuación (2.2) para N variables y *cross-entropy*, que se muestra en la ecuación (2.3). La primera suele utilizarse para regresiones, mientras que la segunda para problemas de clasificación. En las ecuaciones $y$ es el valor o clase e $\hat{y}$ es la predicción.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{j=1}^{N} (y_j - \hat{y}_j)^2 \tag{2.2}$$

$$L(y, \hat{y}) = \sum_{j=1}^{N} y_j \log(\hat{y}_j) \tag{2.3}$$

Por último, *backpropagation*. Este método es el utilizado para actualizar los pesos de la red neuronal. Recibe ese nombre porque, al contrario del uso natural de la red, la actualización se realiza de la última capa a la primera siguiendo la ecuación (2.1) junto con las ecuaciones (2.4) y (2.5), siendo $\ell$ la función de activación no lineal, $i$ el índice de la neurona de la capa anterior y $j$ la neurona de la capa actual.

$$\frac{\partial L}{\partial w_{ij}} = x_i \delta_j \tag{2.4}$$

$$\delta_j = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial F_j} = \begin{cases} \frac{\partial L(y_j,t)}{\partial y_j}\frac{d\varphi(F_j)}{dF_j} & \text{si } j \text{ es una neurona de capa de salida.} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)\frac{d\varphi(F_j)}{dF_j} & \text{si } j \text{ es una neurona de otra capa.} \end{cases} \tag{2.5}$$

#### 2.1.1.4. Otros conceptos

Típicamente, en los problemas de *machine learning* en que hay datos etiquetados estos se dividen en tres conjuntos. El primero es de entrenamiento y contiene el mayor porcentaje de los datos, generalmente 60 %, y son los datos con los que el modelo entrena. El siguiente conjunto es el de validación que contiene un 20 % de los datos y en estos datos se valida que el modelo esté bien ajustado. Finalmente el conjunto de prueba contiene el 20 % restante y es la validación final del modelo. Esta división se hace ya que el desafío central al que se enfrentan las redes neuronales y los algoritmos de *machine learning* es que deben desempeñarse correctamente en datos nuevos, no utilizados en el entrenamiento, es decir debe aprender a generalizar. Es aquí donde surgen dos conceptos importantes: *overfitting* y *underfitting* [12]

De manera simple, el *ovefitting* es cuando la red se sobre ajusta a los ejemplos del entrenamiento, para los que tiene un desempeño excelente mientras que para otros ejemplos tiene un rendimiento paupérrimo, es decir, la red no es capaz de generalizar los ejemplos de la vida real. Esto puede deberse a falta de ejemplos de entrenamiento o a un sobreentrenamiento.

Por otro lado, *underfitting* es lo contrario, cuando el modelo se desempeña mal con los ejemplos de entrenamiento, lo que implica que el modelo es muy simple para describir los datos. En la Figura 2.3 se observa como el modelo pasa de *undefit*, a estar bien ajustado a los datos y a pasar a estar sobre ajustado o *overfit*. Esto se puede ver en el error de prueba, que son datos no utilizados, que comienza a subir mientras el error de entrenamiento sigue bajando.

Una manera para disminuir el *overfit* es utilizar técnicas de regularización. Algunas de ellas son: *dropout* y regularización $L_1$ y $L_2$. *Dropout* consiste en desactivar neuronas, siguiendo una probabilidad, durante el entrenamiento. Esto permite evitar la sobre especialización de una neurona, haciendo a las neuronas menos sensibles a pesos específicos [17].

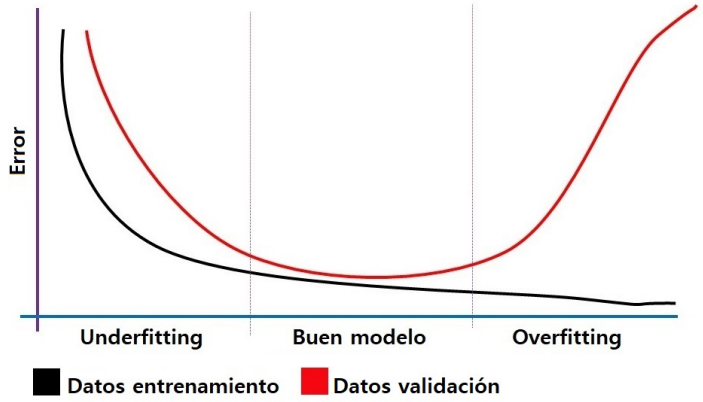


Figura 2.3: Evolución del error del modelo y zonas de *underfitting*, buen desempeño y *overfitting* [18].

Por otro lado, $L_1$ y $L_2$ consiste en agregar $\sum |w_i|$ y $\sum w_i^2$, respectivamente, a la función de perdida, reduciendo el *overfit* dado que la optimización nunca tiene acceso al valor real y, además, los pesos se ven forzados a tener valores más pequeños.

Finalmente, otro concepto importante son los *outliers*. Un *outlier* es "una observación que se desvía tanto de otras observaciones que levanta sospechas de que fue generada por un mecanismo diferente" [19], estos elementos pueden ser tanto perjudiciales y tratados como ruido para ciertas aplicaciones y algoritmos, por ejemplo en una regresión, o bien el objeto de estudio para otras, como detección de fraude [20].

### 2.1.2. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales, o CNN, son un tipo de redes neuronales ampliamente utilizadas para procesar imágenes. Como su nombre lo indica, este tipo de redes utiliza, en al menos una de sus capas, la operación matemática lineal llamada convolución en vez de la multiplicación matricial. Este tipo de redes fueron introducidas por LeCun en 1998 [21] y han sido tremendamente exitosas en aplicaciones de procesamiento de imágenes, posicionándose por sobre métodos de extracción de características clásicos como HOG, LBP o SIFT [22]. En la Figura 2.4 se muestra la CNN LeNet-5 introducida por LeCun [21] para reconocer dígitos.

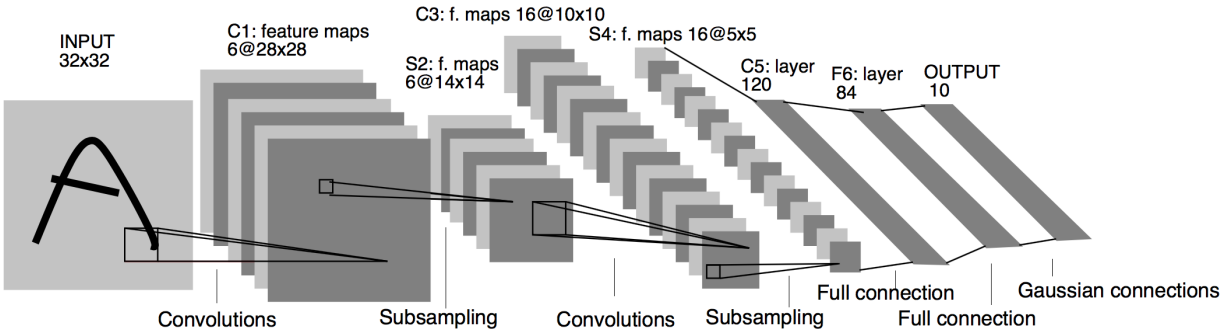


Figura 2.4: Arquitectura de LeNet-5, CNN utilizada para reconocimientos de dígitos [21].

Para profundizar sobre las redes convolucionales y explicar sus ventajas sobre las redes *fully-connected* es necesario explicar el funcionamiento de sus capas.

#### 2.1.2.1. Capa convolucional

La capa convolucional implementa la convolución 2D que se puede ver en la Figura 2.5, en la que se convoluciona una imagen $I$ de $5 \times 5$ píxeles con un kernel o filtro $k$ de $3 \times 3$. Matemáticamente, la convolución 2D sigue la ecuación (2.6). De manera intuitiva esto se puede ver como "poner" el filtro sobre la imagen y multiplicar los elementos de la imagen con los del filtro, luego sumar y mover el filtro a otra posición, realizando esto hasta cubrir todas las posiciones. La convolución 2D también se puede aplicar con un filtro de 3 dimensiones pero su profundidad debe ser la misma que la de la imagen, como se muestra en la Figura 2.6. En ambos ejemplos la dimensión de la salida es 1 puesto que tanto la imagen como el filtro tienen la misma profundidad.

$$y[\mathrm{i}, j] = \sum_{u=0}^{L_x} \sum_{v=0}^{L_x} k[v, u] \cdot I[\mathrm{i} - u, j - v] \tag{2.6}$$

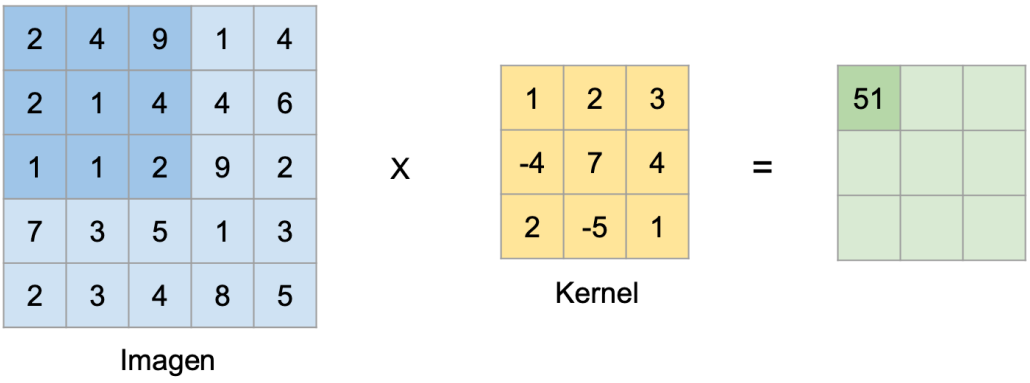


Figura 2.5: Ejemplo de convolución 2D entre una imagen y un filtro de profundidad 1 [23].

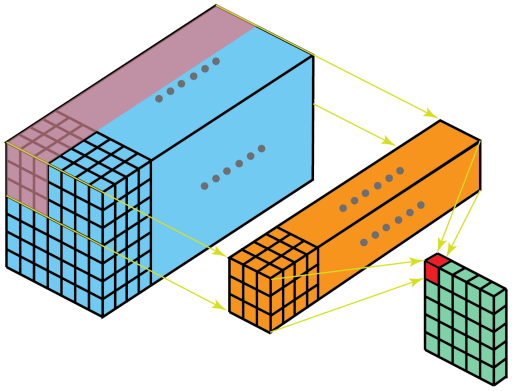

Figura 2.6: Ejemplo de convolución 2D entre una imagen y un filtro de profundidad $N$ [24].

El tamaño de salida, alto y ancho, depende de dos hiperparámetros, el *stride* y el *zero padding*. El *stride* define cuántos píxeles avanza el filtro sobre la imagen, tomando la Figura 2.5, el *stride* es de 1, puesto que la salida es de $3 \times 3$. El otro hiperparámetro, el *zero padding*, define cuantos ceros se agregan al borde la imagen, lo que permite controlar el tamaño de salida modificando el tamaño de entrada. Si no hay *zero padding*, la salida siempre será menor que la entrada. La capa convolucional puede tener tantos filtros como se requieran, esta cantidad de filtros definen la tercera dimensión de la salida.

#### 2.1.2.2. Capa *max pooling*

Otra de las capas que se utilizan en las redes convolucionales es la capa de *max pooling*. Esta capa se utiliza para reducir el tamaño de la imagen, reduciendo así el número de parámetros, controlando el *overfit* y disminuyendo el costo computacional. Como se muestra en la Figura 2.7, el *max pooling* reduce la dimensionalidad eligiendo el máximo de una zona de la imagen, en la Figura se hace con un filtro de $2 \times 2$ y un *stride* de 2, es decir, saltos de 2 píxeles, definiendo las zonas de colores.

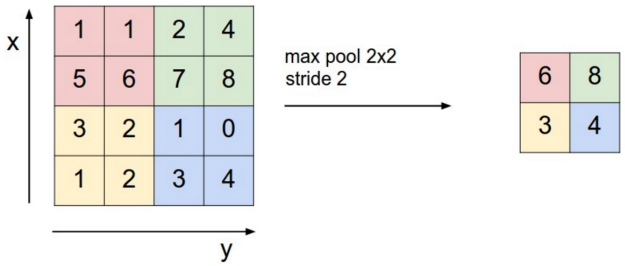


Figura 2.7: Ejemplo de *maxpooling* [25].

#### 2.1.2.3. Capa *fully connected*

La capa *fully connected* corresponde a un MLP. Esta capa se utiliza para realizar la clasificación de la imagen, por lo que, comúnmente, se encuentra al final de la red [21][22] [26]. Para ingresar los parámetros a esta capa se realiza una operación llamada *flatten*, es

decir, se cambia la dimensionalidad de 3D a 1D, manteniendo el número de parámetros. Por ejemplo, una entrada de $3 \times 3 \times 3$ pasa a ser una entrada de $27 \times 1 \times 1$.

### 2.1.3. Detección de objetos

La detección de objetos, a diferencia de la clasificación, corresponde a la tarea de encontrar todas las instancias de objetos en una imagen que correspondan a una o más clases específicas. Lo que significa, encontrar el objeto, clasificarlo y entregar su posición en la imagen a través de un *bounding box* [27] [28] [29] [30]. En la Figura 2.8 se observan 4 tareas distintas con visión computacional, la primera corresponde a la clasificación, es decir, que clase está presente en la imagen, la segunda corresponde a la clasificación y localización de un objeto, la tercera aplicación y la que interesa para el desarrollo de este trabajo es la detección de objetos, que clasifica y localiza con *bounding boxes* a múltiples objetos presentes en una imagen, y finalmente la segmentación que clasifica los píxeles de los objetos presentes en la imagen y no sólo encerrarlos en *bounding boxes* sino en polígonos.

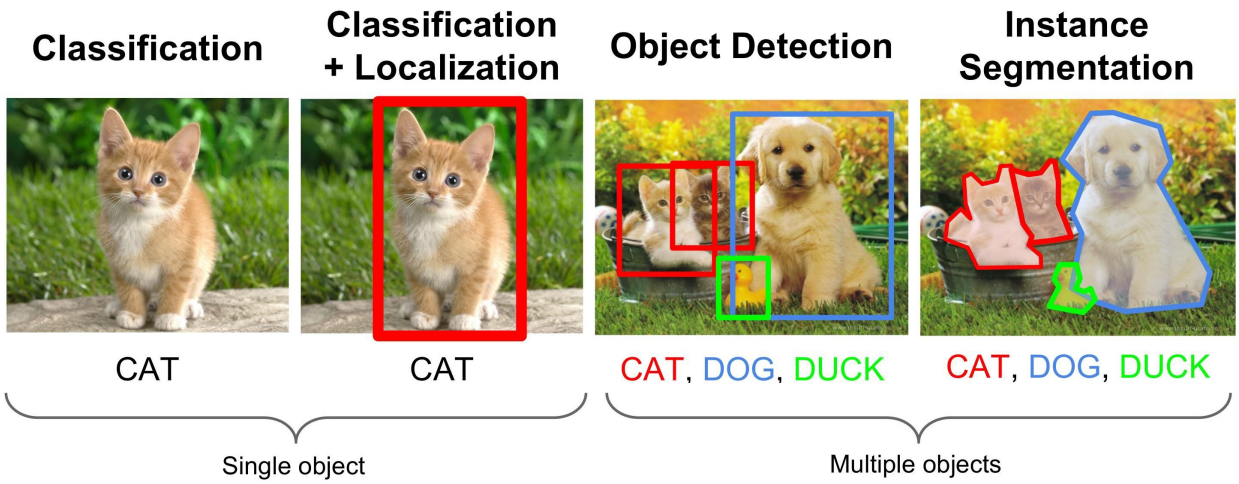


Figura 2.8: Diferentes tipos de tareas con visión computacional. De izquierda a derecha: Clasificación, clasificación y localización, detección de objetos y segmentación [31].

Existen, principalmente, dos métodos para detectar objetos. El primero es la detección de una etapa (o *one-stage*) [27] [30], en que la imagen pasa una sola vez por la red. El segundo corresponde a la detección en dos etapas (o *two-stage*) [28] [29] en que pasa dos veces por la red, en la primera pasada se proponen regiones de interés, donde pueden haber objetos y en la segunda se clasifican estos objetos. En este trabajo se utilizará el primer método, específicamente a partir de SSD, puesto que, por lo general, funciona más rápido que otros algoritmos [32], lo que es importante para esta aplicación ya que debe ser capaz de detectar los objetos mientras van pasando por la cinta transportadora como se mencionó en el capítulo 1.2.

### 2.1.4. Benchmarks

A continuación se muestra la comparación de velocidad de los modelos presentes en *Tensorflow Object Detection API*, uno de los *frameworks* de *Deep Learning* de código abierto más

populares [33][34], y que se utiliza en este trabajo por la experiencia del autor. Por otra parte, se muestra la comparación de velocidades de distintas redes implementadas en el dispositivo embebido utilizado.

#### 2.1.4.1. *Tensorflow Object Detection API* V1

En la Tabla 2.1 [1] se muestran los 25 modelos preentrenados en COCO reportados en *Tensorflow Object Detection API* [35]. Para cada uno de ellos se muestra el tiempo de procesamiento de una imagen de 600x600 píxeles y en una GPU NVIDIA GeForce GTX TITAN X. En negrita se destaca el modelo escogido y entrenado en este trabajo, debido a su velocidad y porque otros modelos tienen problemas abierto para su entrenamiento [2].

Tabla 2.1: Modelos disponibles en Tensorflow OD API preentrenados en COCO

| **Nombre del modelo** | **Tiempo de inferencia [ms]** | **COCO mAP** | **Tipo de salida** |
|---|---|---|---|
| SSD MobileNetV1 | 30 | 21 | *Boxes* |
| SSD MobileNetV1 0.75 depth | 26 | 18 | *Boxes* |
| SSD MobileNetV1 quantized | 29 | 18 | *Boxes* |
| SSD MobileNetV1 0.75 depth quantized | 29 | 16 | *Boxes* |
| SSD MobileNetV1 PPN | 26 | 20 | *Boxes* |
| SSD MobileNetV1 FPN | 56 | 32 | *Boxes* |
| SSD ResNet 50 FPN | 76 | 35 | *Boxes* |
| **SSD MobileNetV2** | **31** | **22** | ***Boxes*** |
| SSD MobileNetV2 quantized | 29 | 22 | *Boxes* |
| SSDlite MobileNetV2 | 27 | 22 | *Boxes* |
| SSD Inception v2 | 42 | 24 | *Boxes* |
| Faster RCNN Inception v2 | 58 | 28 | *Boxes* |
| Faster RCNN ResNet50 | 89 | 30 | *Boxes* |
| Faster RCNN ResNet50 lowproposals | 64 | - | *Boxes* |
| RFCN ResNet101 | 92 | 30 | *Boxes* |
| Faster RCNN ResNet101 | 106 | 32 | *Boxes* |
| Faster RCNN ResNet101 lowproposals | 82 | - | *Boxes* |
| Faster RCNN Inception ResNet v2 atrous | 620 | 37 | *Boxes* |
| Faster RCNN Inception ResNet v2 atrous lowproposals | 241 | - | *Boxes* |
| Faster RCNN NAS | 1833 | 43 | *Boxes* |
| Faster RCNN NAS lowproposals | 540 | - | *Boxes* |
| Mask RCNN Inception ResNet v2 atrous | 771 | 36 | *Masks* |
| Mask RCNN Inception v2 | 79 | 25 | *Masks* |
| Mask RCNN ResNet101 atrous | 470 | 33 | *Masks* |
| Mask RCNN ResNet50 atrous | 343 | 29 | *Masks* |

[1] Tabla obtenida de la página oficial de Tensorflow: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

[2] https://github.com/tensorflow/models/issues/5028#issuecomment-689671915

#### 2.1.4.2. Edge

En la Figura 2.9[3] se muestran los desempeños de varios modelos de redes convolucionales sobre una NVIDIA Jetson Nano, que será el sistema *on the edge* que se utilizará, estos modelos desempeñan tareas de clasificación (ResNet-50, Inception V4, VGG-19), detección de pose (OpenPose), segmentación (U-Net), aumento de resolución de imagen (Super Resolution) y detección de objetos (SSD y Tiny YOLO V3). Este *benchmark* fue realizado utilizando la librería de aceleración de NVIDIA: TensorRT.

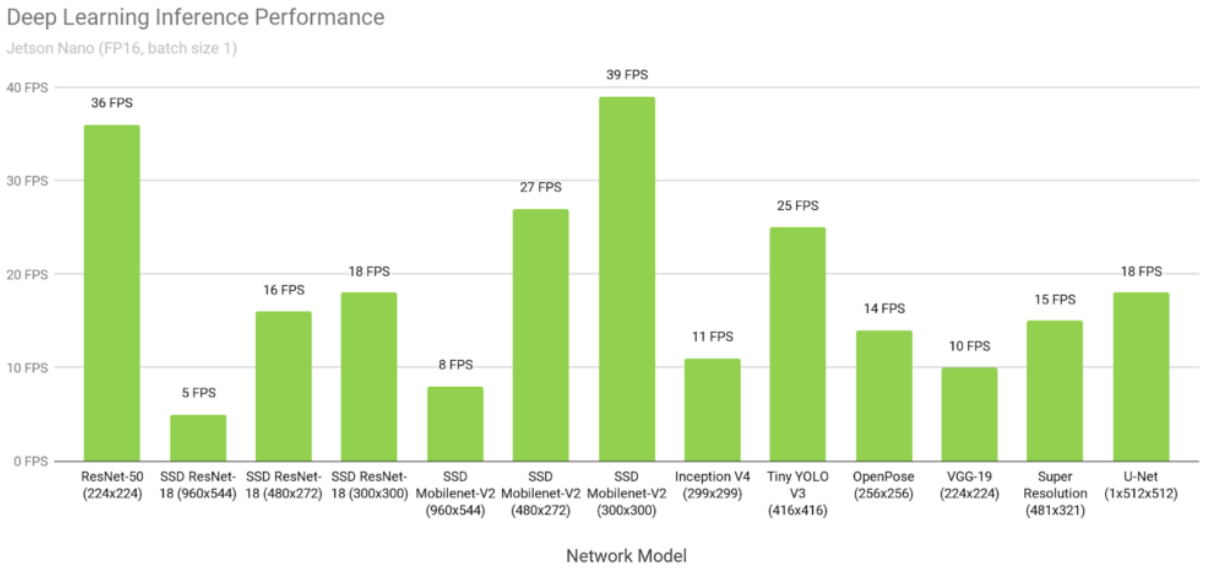


Figura 2.9: Desempeño de varios algoritmos utilizando una NVIDIA Jetson Nano.

### 2.1.5. *Single Shot Detector* (SSD)

La red *Single Shot Detector* es introducida el año 2016 como una red de detección de objetos de una sola pasada. A continuación se explicará el funcionamiento de esta red.

#### 2.1.5.1. Modelo

La arquitectura se puede dividir en dos partes, una de extracción de características llamada *backbone* y otra de detección de objetos, donde se hacen las predicciones. Para la parte de extracción de características el diseño original de la red utiliza una red de clasificación VGG-16 [26] truncada en la capa "conv4_3" de tamaño $38 \times 38 \times 512$ [32], pero cualquier otra red convolucional puede ser utilizada.

Siguiendo el trabajo original, a la capa "conv4_3" se le aplica una convolución con un filtro de $3 \times 3 \times 512 \times (k \times (c+4))$, donde $k$ es el número de *bounding boxes* a predecir, con $k = 4$ en el trabajo original, y $c$ el número total de clases. Es decir, se realizan $38 \times 38x4 = 5776$ predicciones de *bounding boxes* en esa capa, y cada una de esas predicciones contiene $c + 4$

[3] Imagen obtenida de la página oficial de NVIDIA: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks

parámetros, uno por cada clase y 4 del *offset* relativos al tamaño de la *bounding box* original, estos son $\Delta cx$, $\Delta cy$, $\Delta w$ y $\Delta h$.

Además, se añaden 5 capas de detección de objetos donde se realizan predicciones, cada capa tiene un tamaño menor que la anterior y en cada una de ellas ocurre el mismo proceso, en el que el número de predicciones es $m \times n \times k$. En tres de estas capas $k = 6$, mientras que en las otras dos $k = 4$. En total se realizan 8732 predicciones de detecciones. En la Figura 2.10 se aprecia la arquitectura de la red SSD, con un *backbone* de VGG-16 [26] y sus capas de detección de objetos.

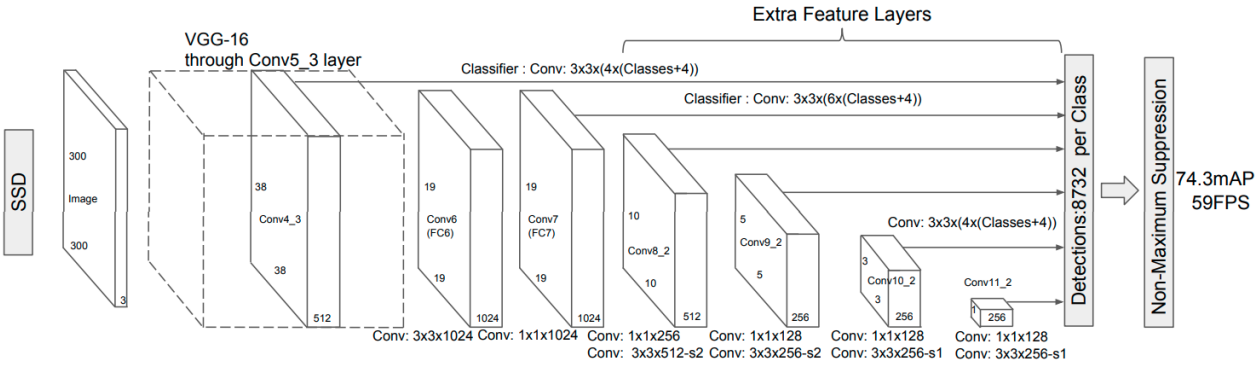


Figura 2.10: Arquitectura de la red SSD con tamaño de entrada de imagen de $300 \times 300$. El modelo añade varias capas de extracción de características al final de la red *backbone* [32].

Una característica interesante del modelo es que se utilizan *default boxes* para las predicciones de las cajas. Estas son cajas de tamaño predefinido y tienen relación de aspecto $a_r$ tal que $a_r \in \left\{1, 2, 3, \frac{1}{2}, \frac{1}{3}\right\}$. Además, el tamaño de las cajas también depende de una escala $s_k$ definida para capa de detección según la ecuación (2.7), donde $m$ es el número de capas usadas para realizar predicciones, $s_{\min}$ es la escala mínima definida para la primera capa (más a la izquierda) y $s_{\max}$ es la escala máxima definida para la última capa (más a la derecha). Se define $s_{\min} = 0.2$ y $s_{\max} = 0.9$. También, se añade un último $a_r$, tal que $a_r = \sqrt{s_k s_{k+1}}$. Así la altura de una caja está definida como $h_k^a = s_k/\sqrt{a_r}$ y en ancho como $w_k^a = s_k\sqrt{a_r}$ [32].

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1}(k - 1), \quad k \in [1, m] \tag{2.7}$$

#### 2.1.5.2. Entrenamiento

Para explicar el entrenamiento es importante definir un par de conceptos.

- ***Groundtruth box***

  La caja de verdad absoluta o *groundtruth box* es la *bounding box* etiquetada manualmente que establece dónde está el objeto.

- **IoU: *Intersection over Union***

  La intersección sobre la unión es una medida estándar utilizada para evaluar el solapamiento entre dos *bounding boxes*. Sigue la ecuación (2.8), donde simplemente se divide la intersección por la unión, en este caso las *bounding boxes* a medir son una de *groundtruth*

y otra de detección. La Figura 2.11 muestra gráficamente como se calcula la intersección sobre la unión.

$$\text{IoU} = \frac{\text{area}\left(B_p \cap B_{gt}\right)}{\text{area}\left(B_p \cup B_{gt}\right)} \tag{2.8}$$

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$

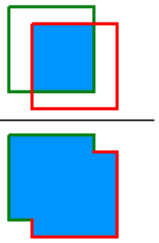

Figura 2.11: Cálculo de la intersección sobre la unión. En verde una detección y en rojo una *groundtruth*. Imagen obtenida de [36].

Las predicciones de la red pueden ser clasificadas como predicciones positivas y negativas. Para calcular el error de localización solo se toman en cuenta las predicciones positivas, que son aquellas en que el IoU de la *default box* con la caja *groundtruth* es mayor a 0,5. Así, se tiene una mejor comprensión de qué parte del IoU es debido a la predicción y qué parte por la *default box*.

Para entrenar se busca minimizar la función de pérdida de la ecuación (2.9), que está dividida en dos partes, una función de pérdida de confianza $L_{conf}$ y una función de pérdida de localización $L_{loc}$. Por su parte, $\alpha$ corresponde a un coeficiente para asignar un peso mayor o menor a $L_{loc}$, $N$ corresponde al número de predicciones positivas (si $N = 0$, entonces $L = 0$), $x$ es un indicador tal que $x_{ij}^p = \{1, 0\}$, es 1 si es una predicción positiva para la *default box* $i$ y el *groundtruth* $j$ de la clase $p$.

$$L(x, c, l, g) = \frac{1}{N}\left(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)\right) \tag{2.9}$$

La función de pérdida de localización se define como lo muestra la ecuación (2.10) y el error de confianza según la ecuación (2.11), donde para las predicciones negativas se considera la clase 0 o *background*.

$$\begin{array}{rr} L_{loc}(x, l, g) = \sum_{i \in \text{Pos}}^{N} \sum_{m \in \{cx, cy, w, h\}} & x_{ij}^k \, \text{smooth}_{\text{L1}}\left(l_i^m - \hat{g}_j^m\right) \\ \hat{g}_j^{cx} = \left(g_j^{cx} - d_i^{cx}\right)/d_i^w & \hat{g}_j^{cy} = \left(g_j^{cy} - d_i^{cy}\right)/d_i^h \\ \hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) & \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right) \end{array} \tag{2.10}$$

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p \log\left(\hat{c}_i^p\right) - \sum_{i \in Neg} \log\left(\hat{c}_i^0\right) \quad \text{donde} \quad \hat{c}_i^p = \frac{\exp\left(c_i^p\right)}{\sum_p \exp\left(c_i^p\right)} \tag{2.11}$$

Finalmente, para el entrenamiento se ocupan otras dos técnicas *hard negative mining* y *data augmentation*. La primera técnica elige las predicciones negativos que tuvieron mayor error de localización de tal forma que la relación entre predicciones negativas y predicciones

positiva sea como máximo 3:1, así el entrenamiento es más estable y rápido. La segunda, se utiliza para aumentar los datos en el entrenamiento para que la red pueda generalizar de mejor manera, para esto se voltean las imágenes horizontal y verticalmente, se recortan teniendo un IoU con los objetos $\in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, también se aplican distintas distorsiones a las imágenes [32].

#### 2.1.5.3. *Backbone*

Como se mencionó anteriormente, SSD también puede utilizar otros *backbones* como extractores de características. Uno de los más comunes es MobileNet en sus versiones MobileNetV1 [37] y MobileNetV2 [38], esto ya que son redes optimizadas para un funcionamiento en dispositivos móviles, por lo que son rápidas, al igual que SSD. Otros extractores utilizados son las redes ResNet [39] en sus versiones ResNet50, ResNet101 y ResNet152. El *backbone* escogido dependerá del foco de la tarea a realizar, en este trabajo se prioriza la velocidad por sobre la precisión del modelo, por lo que se escoge MobileNetV2.

## 2.2. Bases de datos y métricas de detección de objetos

Las bases de datos moldean los problemas técnicos que los investigadores estudian. Es gracias a estas bases de datos que se han desarrollado distintos algoritmos en el área de visión computacional [40]. Esto ocurre ya que los científicos comparten un "problema común", donde se documentan mejor los resultados y, como los datos son de acceso público, los resultados son reproducibles. Además, para gran parte de estos algoritmos no solo se comparten los resultados, sino que también el código del algoritmo mismo [41].

A continuación, se describirán tres de las bases de datos más usadas en la detección de objetos, *Pascal VOC* [42], *COCO* [43] y *Open Images* [44]. Estas se ocupan como *benchmarks* para comparar los distintos algoritmos de detección. También se describirá una base de datos sobre "anotaciones basura en su contexto", o TACO por sus siglas en inglés *Trash Annotations in Context Dataset* [45], que es el dataset más cercano a la tarea que se busca solucionar en este trabajo.

### 2.2.1. Pascal VOC

*Pascal VOC* (*Visual Object Classes*) es una base de datos que contiene 11.000 imágenes y 20 categorías. Hay más de 27.000 objetos con sus respectivas clases y *bounding boxes*. Esta base de datos se encuentra disponible desde el año 2007 y, al igual que COCO, se establecía una competencia anual, *The PASCAL Visual Object Classes (VOC) Challenge*, hasta el año 2012.

### 2.2.2. COCO

*COCO* es una de las bases de datos más utilizadas para establecer métricas de detección de objetos. Contiene más de 200.000 imágenes y 80 categorías [43]. Con esta base de datos, desde el 2015, se establece una competencia anual en la Conferencia Internacional sobre Visión Computacional (*ICCV: International Conference on Computer Vision*. De esta base

de datos se utilizará una clase que contiene imágenes relevantes para la aplicación del trabajo, esta es *bottle*, con 8.880 imágenes.

En la Figura 2.12 se pueden ver las distintas clases de *COCO* y *Pascal VOC*, se observa que la segunda tiene menos categorías e instancias, por lo que *COCO* es una base de datos más completa. De *Pascal VOC* la única clase que sirve es *bottle*. El problema para este trabajo con ambas bases de datos (*COCO* y *Pascal VOC*) es que ninguna de las etiquetas es un material reciclable, por ejemplo la clase *bottle* puede ser tanto vidrio, PET u otro, por lo que las imágenes deben ser filtradas y reetiquetadas con el material correspondiente.

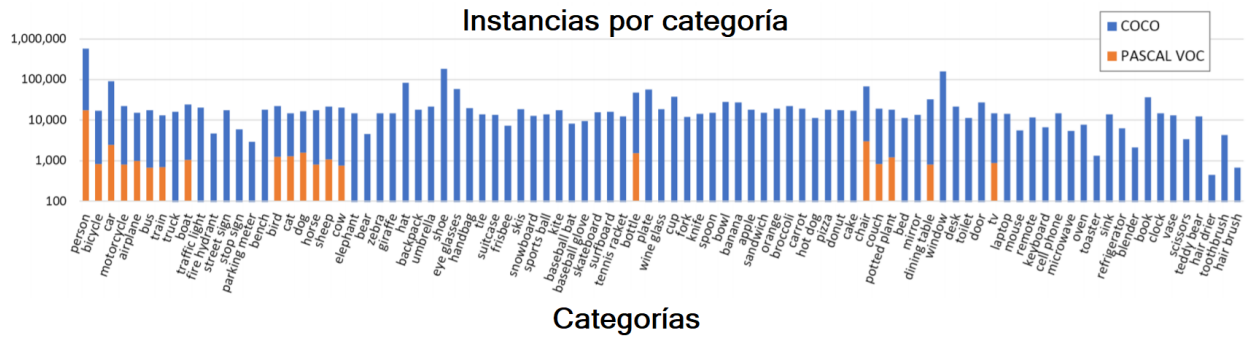


Figura 2.12: Instancias por categorías de *COCO* y *Pascal VOC*.

### 2.2.3. *Open Images*

*Open Images Dataset V4 (OIDV4)* es un enorme *dataset* desarrollado por Google AI con 9,2 millones de imágenes y contiene marcas de clasificación, detección y relaciones visuales. En específico para la tarea de detección contiene 15,4 millones de *bounding boxes* separados en 600 categorías [44]. Para esta tarea existen tres clases que son de interés: *bottle*, *tin can* y *beer*, pero esta última está contenida por las primeras dos, por lo que solo se utilizarán las primeras. La clase *bottle* contiene 11.456 imágenes y 25.545 marcas, mientras que *tin can* contiene 1.307 imágenes y 2.585 marcas.

### 2.2.4. TACO

*TACO* es una base de datos de basura en diversos ambientes como en la naturaleza, agua, pavimento, entre otros. Fue creada el 2019 y aún es una base de datos en crecimiento [45]. Consta de 60 subclases que son parte de 28 macro clases. Al 10 de mayo de 2020 la base de datos consta de 1.500 imágenes etiquetadas con 4.784 objetos etiquetados y 2.831 imágenes por etiquetar [4]. Algunas de las clases que sirven para este trabajo son *glass bottle*, *clear plastic bottle*, *broken glass* y *drink can*.

[4] http://tacodataset.org/stats

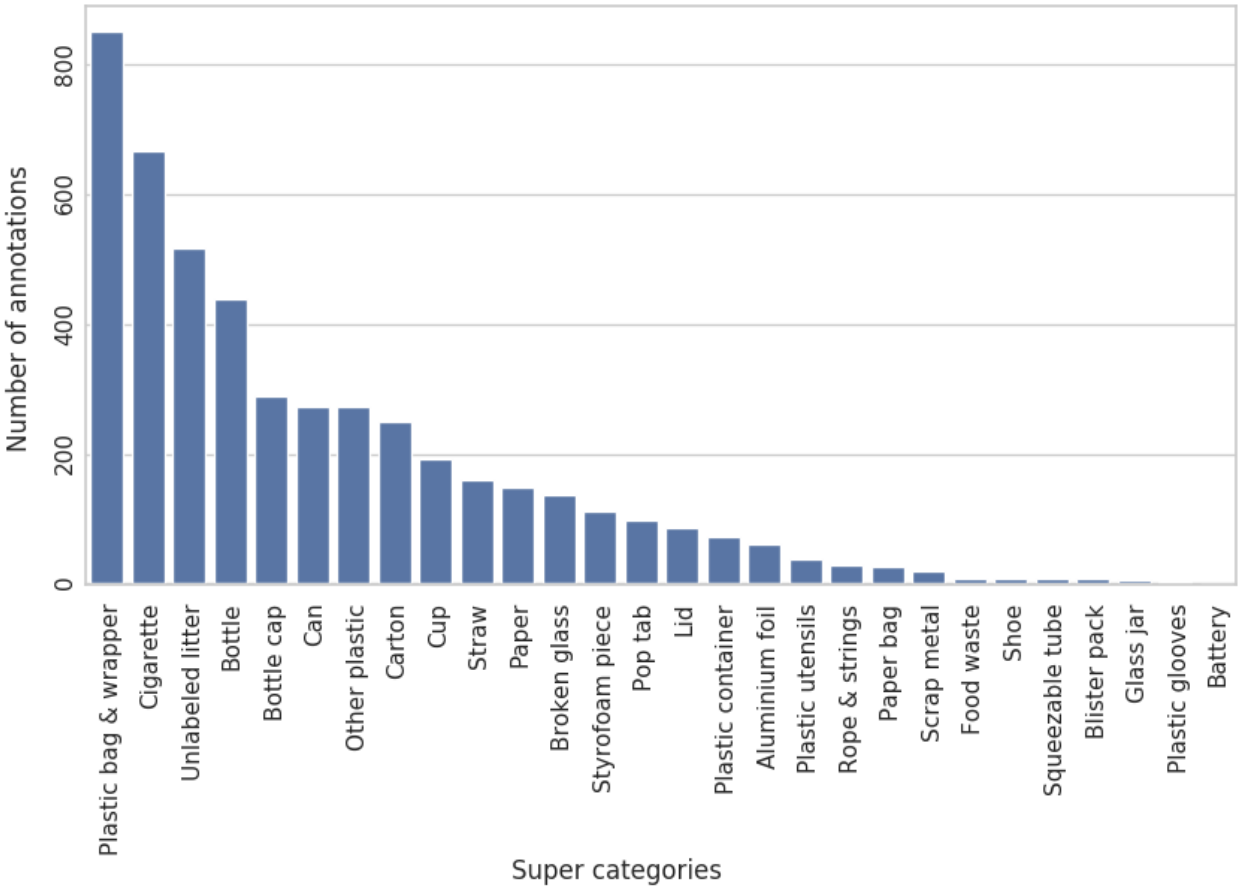


Figura 2.13: Instancias por macro clase en *TACO*.

### 2.2.5. Métricas de Desempeño

A continuación, se describirán las métricas de desempeño que se suelen utilizar para comparar algoritmos de detección de objetos en competencias como *COCO*[43] y *Pascal VOC*[42], y que se utilizarán en esta memoria para medir los resultados de la red a implementar.

#### 2.2.5.1. *Precision* y *recall*

Para definir *precision* y *recall* hay que entender lo que son los verdaderos positivos, falsos positivos y falsos negativos. Estos se determinan según un umbral de IoU, que se denominará $IoU_t$, tal que:

- **Verdadero Positivo (TP):** ocurre cuando el IoU entre la detección y el *groundtruth* es mayor a $IoU_t$ y las clases concuerdan.
- **Falso Positivo (FP):** ocurre cuando el IoU entre la detección y el *groundtruth* es menor a $IoU_t$ o bien, es mayor a $IoU_t$ pero ya existe un TP con una confianza mayor.
- **Falso Negativo (FN):** ocurre cuando un *groundtruth* no tiene TP, es decir, o no hay detección con IoU mayor a $IoU_t$ o se equivoca en la clase.

*Precision* corresponde a la cantidad de detecciones correctas sobre el total de detecciones, se calcula como lo muestra la ecuación (2.12). Por su parte el *recall* corresponde a la cantidad

de detecciones correctas sobre el total de *groundtruth* presentes y se calcula como lo muestra la ecuación (2.13). Donde TP, FP y FN son las siglas en inglés de verdaderos positivos, falsos positivos y falsos negativos, respectivamente.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{total de detecciones}} \tag{2.12}$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{total de } groundtruth} \tag{2.13}$$

#### 2.2.5.2. Curva *precision-recall* y *Average Precision (AP)*

La curva de *precision-recall* se calcula siguiendo el siguiente algoritmo para cada clase del *dataset* a medir.

1. Se ordenan las detecciones según su *confidence*.
2. Según el $IoU_t$ escogido se determina si la detección corresponde a un TP o FP.
3. Se calcula el *recall* en cada posición según el número de TP acumulados hasta esa posición y la *precision* con el número de TP y FP acumulados hasta esa posición.

Una forma fácil de comparar detectores es a través de un único valor numérico, esto se logra con el AP, que representa el área bajo la curva de la curva *precision-recall* [36]. Las ecuaciones 2.14 y 2.15 muestran cómo calcular el AP. Se realiza una interpolación entre todos los puntos de *recall* $r_n$ en la curva, tomando el máximo hasta el punto $r_n$, luego, para calcular el AP basta con sumar las áreas como muestra la Figura 2.14.

$$AP = \sum_{n=0} (r_{n+1} - r_n)\, \rho_{\text{interp}}\,(r_{n+1}) \tag{2.14}$$

$$\rho_{\text{interp}}\,(r_{n+1}) = \max_{\tilde{r}:\tilde{r} \geq r_{n+1}} \rho(\tilde{r}) \tag{2.15}$$

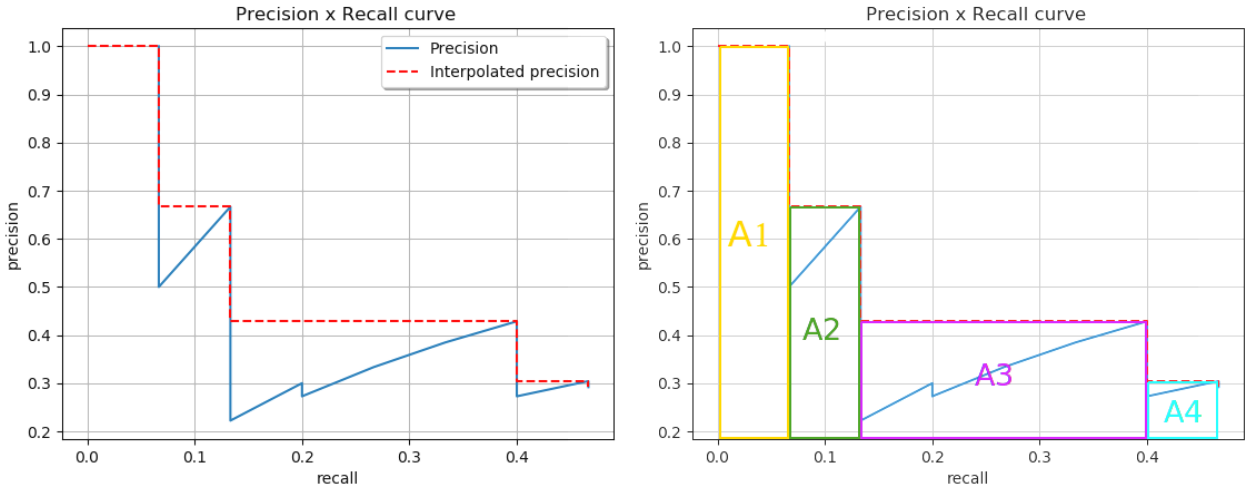


Figura 2.14: Cálculo del AP, se realiza una interpolación de la precisión y luego se suma el área. Imagen obtenida de [36].

En *COCO* [43] se utilizan valores de $IoU_t$ entre 0,5 y 0,95 con un paso de 0,05 para obtener los AP, luego son promediados obteniendo un mAP o *mean average precision*. En este trabajo se utilizarán los términos $AP^{50}$ y $AP^{75}$ para referirse a AP con $IoU_t$ de 0,5 y 0,75, respectivamente, mientras que el término AP se utilizará como promedio de los AP entre 0,5 y 0,95 de una sola clase y, finalmente, mAP se utilizará como promedio de los AP entre 0,5 y 0,95 de todas las clases.

# Capítulo 3

# Metodología y Aportes del Trabajo de Memoria

El ciclo de trabajo de esta memoria se puede dividir en dos partes, una de planificación y otra de desarrollo. En la parte de planificación, que se realiza una única vez, se lleva a cabo el diseño teórico de la solución (descrito en la sección 1.2), la selección de componentes y, el presupuesto y compras. La segunda parte, de desarrollo, se efectúa de manera iterativa, y se realizan las tareas de captura o adquisición de imágenes, el etiquetado de estas, el entrenamiento de las redes neuronales y la implementación en el sistema *on the edge*. En la Figura 3.1 se aprecia el ciclo de trabajo iterativo, con los pasos mencionados anteriormente y que se describen en esta sección.

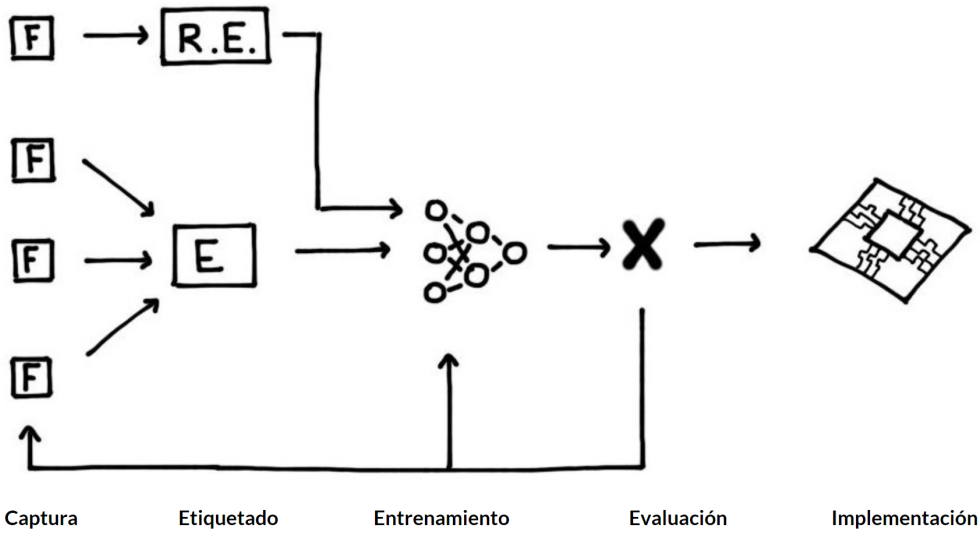


Figura 3.1: Ciclo de trabajo iterativo.

## 3.1. Selección de componentes

Se escoge como dispositivo *edge* la NVIDIA Jetson Nano en su versión Kit de Desarrollo (*Developer Kit*), pues es de bajo coste, está pensada para aplicaciones de *deep learning* y soporta distintos *frameworks* como Tensorflow y Pytorch. Luego, se eligen componentes que sean compatibles con el dispositivo, como la cámara RGB, la fuente de alimentación

y la tarjeta de almacenamiento, estos últimos dos son sumamente importantes para que la Jetson Nano pueda funcionar, ya que no vienen incluidos en el Kit de Desarrollo. Así, los componentes seleccionados son:

- NVIDIA Jetson Nano Developer Kit, para realizar el procesamiento.
- Raspberry Pi Camera Module V2, para capturar las imágenes.
- MicroSD UHS-1 64 GB, para instalar el sistema operativo y las librerías necesarias.
- Fuente de alimentación 5V 4A, para alimentar la Jetson Nano y sus periféricos.

También se utilizan mouse, teclado y pantalla para controlar el equipo, pero debido a que sólo se utilizan una vez para configurar la Jetson Nano, no se mencionan dentro de los componentes seleccionados.

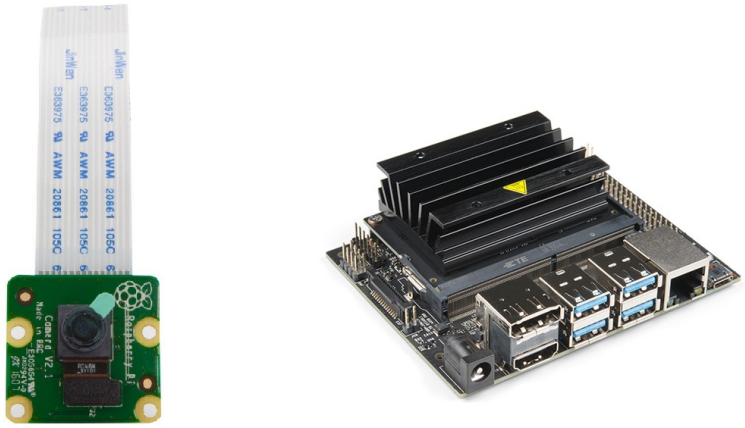

Figura 3.2: A la izquierda: Raspberry Pi Camera Module V2. A la derecha: NVIDIA Jetson Nano Developer Kit.

El costo total es de $ 223.260 [CLP] sin incluir costos de envío y el detalle se encuentra en el Anexo A. Se adquieren todos los componentes en el mercado nacional por la facilidad de compra y los tiempos de despacho.

## 3.2. Adquisición de imágenes

Se obtienen imágenes de diversas fuentes de internet, en particular, de las bases de datos mencionadas en el marco teórico y descripción del problema: *COCO*, *Open Images*, *Trashnet* y *Waste Classification Data*. Además, se realiza una recolección propia a partir de imágenes capturadas por alumnos de la Universidad de Chile (*Cubelli*) y una recolección pagada a la empresa Tagias (*Tagias*)[1].

A continuación, en la Tabla 3.1 se muestran las bases de datos, la cantidad de imágenes disponibles en ellas relacionadas con este trabajo y la tarea que se debe realizar sobre dichas imágenes para poder ser utilizadas. Hay dos tareas a realizar: cambiar etiqueta y etiquetar.

[1] https://tagias.com/

La primera corresponde a cambiar la etiqueta de los *bounding boxes* que contienen elementos reciclables, como las botellas de *COCO* y *Open Images* que deben ser reetiquetadas como vidrio, PET u otro. La segunda tarea es marcar las *bounding boxes* de los objetos.

Tabla 3.1: Cantidad de imágenes utilizadas por base de datos.

| **Base de datos** | **Cantidad de imágenes** | **Tarea de marcado** |
|---|---|---|
| COCO | 8.880 | Cambiar etiqueta |
| Open Images | 12.763 | Cambiar etiqueta |
| Trashnet | 935 | Etiquetado |
| Waste Classification | 1.742 | Etiquetado |
| Cubelli | 849 | Etiquetado |
| Tagias | 538 | Captura y etiquetado |
| La Marina | 253 | Etiquetado |

Cabe destacar que todas las imágenes son formato PNG y JPG (o JPEG) pero cualquier formato que sea soportado por PIL[2] puede servir. Además, para evitar problemas de rotación con las plataformas de marcado y librerías de lectura de imágenes en Python, se eliminan los datos Exif que contienen algunas de las imágenes de las bases de datos *Trashnet* y *La Marina*.

Por otra parte, no todas las imágenes son representativas del problema a resolver ya que el contexto en el que están los objetos es distinto al de la aplicación real, hay muchas imágenes con fondo blanco y pocos elementos por imagen. Los mejores datos son los de la base de datos Tagias pues fue generada pensando específicamente en la aplicación de la planta de reciclaje. Las figuras 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, y 3.9 muestran ejemplos de las imágenes contenidas en las bases de datos *COCO*, *Open Images*, *Trashnet*, *Waste Classification*, *Cubelli*, *Tagias* y *La Marina*, respectivamente.


Figura 3.3: Imágenes de muestra de COCO.

[2] $https://pillow.readthedocs.io/en/4.1.x/handbook/image-file-formats.html#fully-supported-formats$


Figura 3.4: Imágenes de muestra de Open Images.

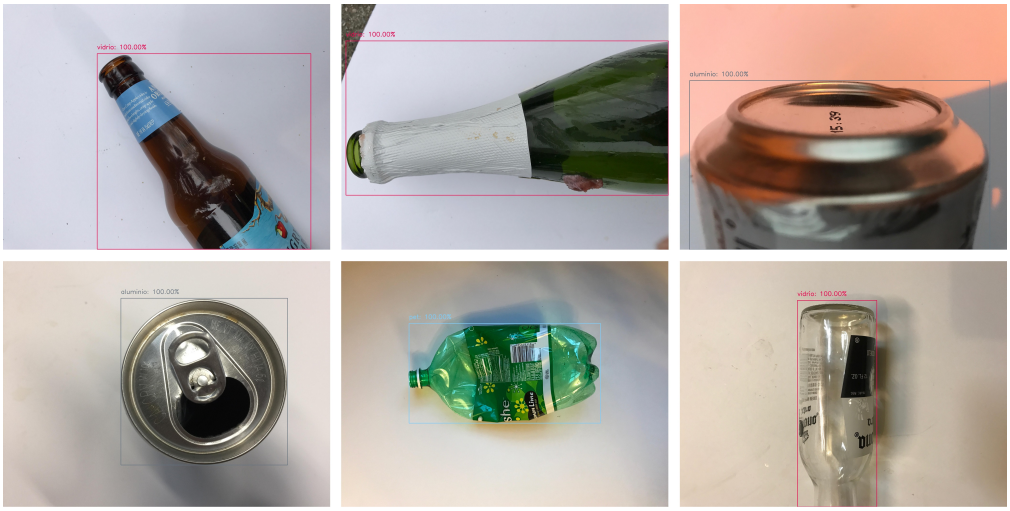

Figura 3.5: Imágenes de muestra de Trashnet.


Figura 3.6: Imágenes de muestra de *Waste Classification*.

Figura 3.7: Imágenes de muestra de Cubelli.

Figura 3.8: Imágenes de muestra de Tagias.


Figura 3.9: Imágenes de muestra de La Marina.

## 3.3. Etiquetado de las imágenes

Para etiquetar las imágenes se utiliza la plataforma online *Labelbox* [46] que permite subir bases de datos en línea, crear proyectos de marcado de detección de objetos con *bounding boxes* y las clases correspondientes, y exportar las marcas. También permite crear proyectos de clasificación para seleccionar la o las clases de una imagen. Para la tarea de etiquetado se suben las imágenes completas y se marcan todos los objetos presentes, mientras que para la tarea de reetiquetado se cortan las secciones de la imagen, tal como se muestra en la Figura 3.10, que contienen al objeto y a ese objeto se le asigna una clase.

Figura 3.10: Instancias de vidrio recortadas para una tarea de reetiquetado.

Se utilizan estos dos tipos de proyectos para marcar las bases de datos. Se escoge esta plataforma porque es fácil de usar, no requiere implementar nada extra y ofrecen licencia educacional gratis. En la Figura 3.11 se aprecia la interfaz de la plataforma *Labelbox* con distintas tareas de etiquetado creadas. En las figuras 3.12 y 3.13 se aprecia la interfaz de *Labelbox* en las tareas de clasificación y marcado de *bounding boxes*, respectivamente.

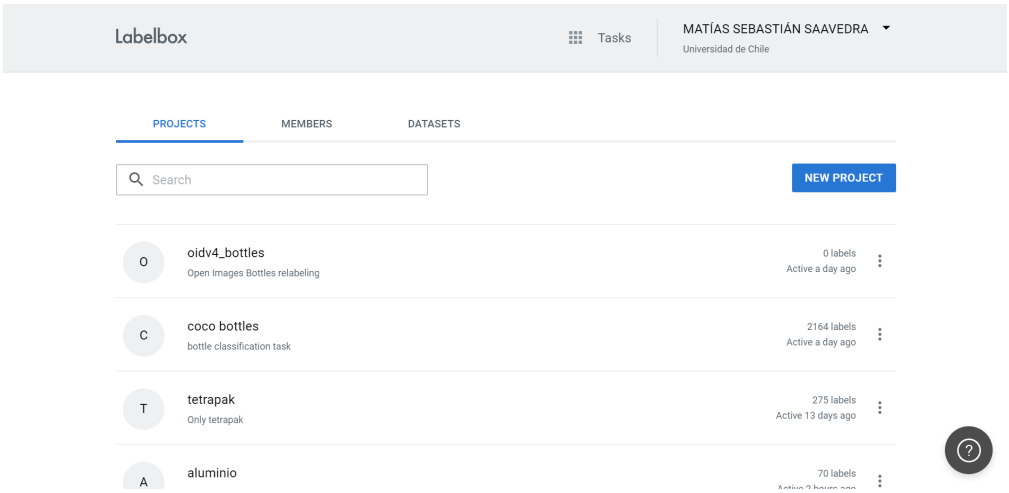


Figura 3.11: Interfaz de Labelbox.

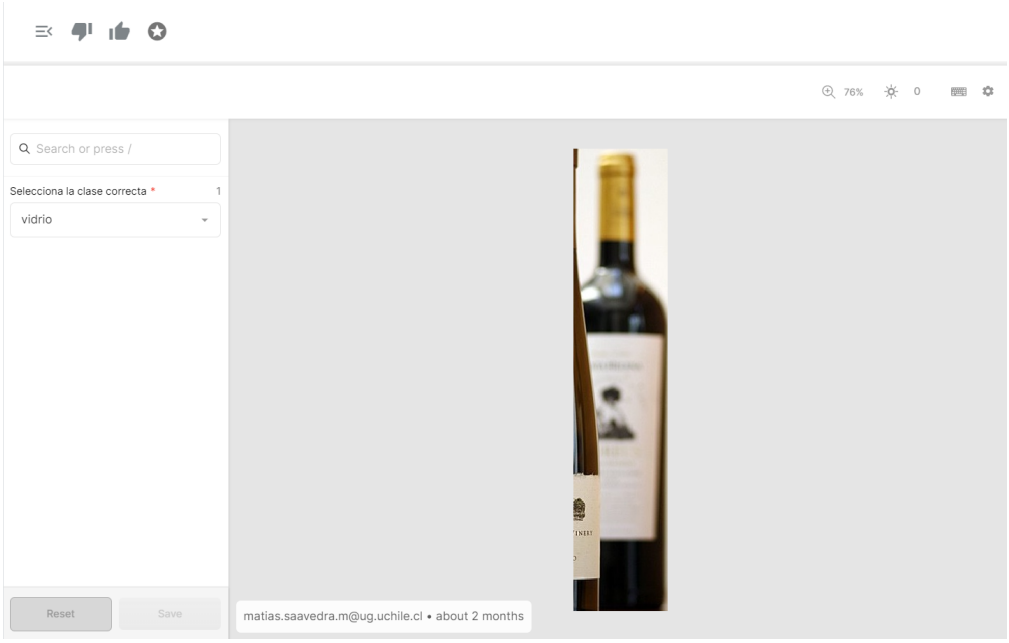


Figura 3.12: Tarea de clasificación en *Labelbox* utilizada para reetiquetar las bases de datos.

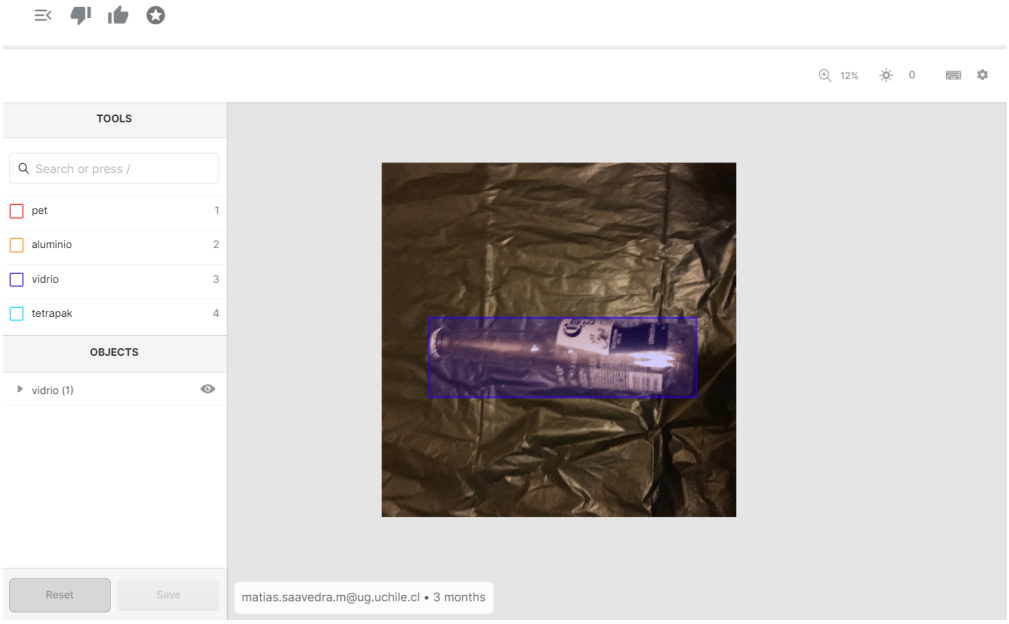


Figura 3.13: Tarea de marcado de *bounding boxes* utilizada para etiquetar las bases de datos.

Se realiza un filtrado de las imágenes disponibles en el proceso de etiquetado, debido a que hay imágenes de dibujos que no corresponden a materiales reciclables, como también en el proceso de reetiquetado en el cual habían presentes otros tipos de plásticos. En específico, para los *datasets* de etiquetado *Waste Classification* y Trashnet se descartan 531 y 24 imágenes, respectivamente.

En las tareas de reetiquetado de *COCO* se descartan 6.155 imágenes. Para el caso de *OIDV4* no se logra reetiquetar todo el *dataset* sino que sólo el 7 % de las marcas de vidrio y PET (2.141 de un total de 27.545). Tampoco se logra etiquetar el aluminio en *OIDV4*, por lo que se dejan esas 2.988 marcas del dataset intactas. En las figuras 3.14 y 3.15 se muestran algunas de las imágenes que fueron descartadas, tanto por ser dibujos o por ser un grupo muy grande de un mismo elemento.


Figura 3.14: Cuatro imágenes que fueron descartadas de *Waste Classification* por ser un grupo denso y difícil de separar de elementos reciclables.


Figura 3.15: Cuatro imágenes que fueron descartadas de *Waste Classification* por no ser materiales reciclables reales sino que dibujos.

En la Tabla 3.2 se muestra la descripción de las bases de datos utilizadas para entrenar, es decir, la cantidad de imágenes, la cantidad de marcas presentes, la cantidad de marcas por imagen de cada base de datos y el precio equivalente de cada *dataset* utilizando como referencia la Tabla de precios de Tagias que se encuentra en el Anexo B. Además, se considera un valor de dólar de $ 788,80 [CLP]. El costo total de las bases de datos es de $ 137.370 [CLP] pero la única base de datos que efectivamente se paga es Tagias con un costo de $ 52.557 [CLP], el resto fue etiquetado personalmente en *Labelbox*.

Tabla 3.2: Descripción de las bases de datos usadas para el entrenamiento y test de la red neuronal.

| **Base de datos** | **Cantidad de imágenes** | **Cantidad de marcas** | **Cantidad de marcas por imagen** | **Precio equivalente [USD]** |
|---|---|---|---|---|
| Waste Classification | 1.211 | 3.065 | 2,53 | $ 42,91 |
| Tagias | 538 | 1.214 | 2,26 | $ 66,63 |
| OIDV4 | 2.428 | 4.729 | 1,95 | $ 18,91 |
| COCO | 2.725 | 4.330 | 1,59 | $ 17,32 |
| La Marina | 253 | 264 | 1,04 | $ 3,69 |
| Trashnet | 911 | 915 | 1,00 | $ 12,81 |
| Cubelli | 849 | 849 | 1,00 | $ 11,88 |

La Figura 3.16 muestra la cantidad de instancias de cada clase presente en todas las imágenes utilizadas para entrenar y la Tabla 3.3 muestra la cantidad de instancias de cada clase desglosada en cada base de datos. Se observa que hay un claro desbalance en las clases ya que hay tan solo 411 ejemplos de tetrapak, mientras que hay 6.481 de vidrio, 4.516 de aluminio y 3.958 de PET.

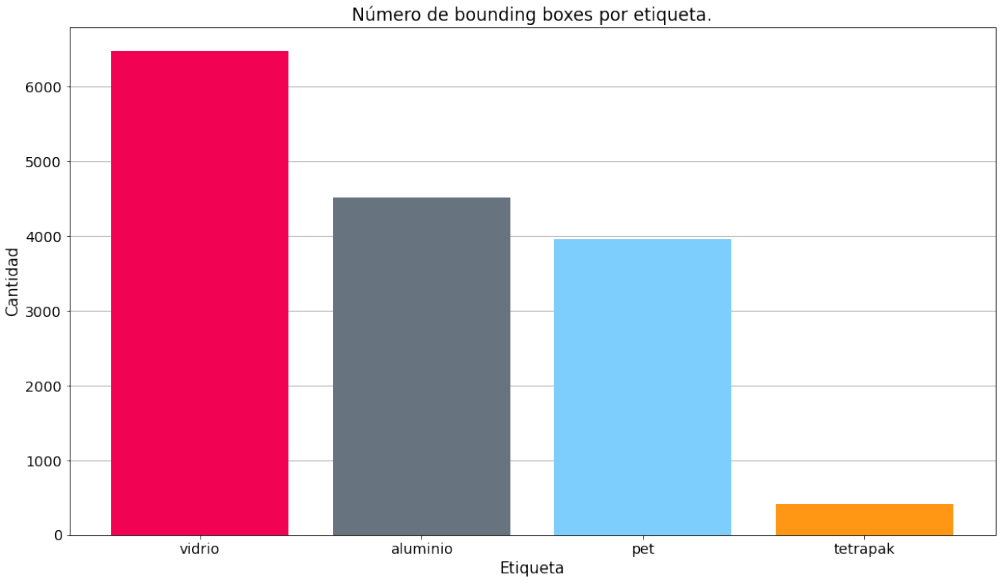


Figura 3.16: Cantidad de instancias por clases en todas las imágenes.

Tabla 3.3: Composición de las bases de datos por instancia de cada clase.

| Base de datos | Vidrio | Aluminio | PET | Tetrapak |
|---|---|---|---|---|
| Waste Classification | 1.028 | 870 | 1.167 | 0 |
| Tagias | 129 | 246 | 733 | 106 |
| OIDV4 | 1.454 | 2.988 | 287 | 0 |
| COCO | 3.259 | 0 | 1.071 | 0 |
| La Marina | 124 | 6 | 104 | 30 |
| Trashnet | 487 | 148 | 280 | 0 |
| Cubelli | 0 | 258 | 316 | 275 |

## 3.4. Entrenamiento de redes neuronales

Para entrenar se utilizan todos los *datasets* a excepción de La Marina que se utiliza para la evaluación, los resultados mostrados son respecto a este *dataset*. Además, el entrenamiento se realiza en una máquina virtual en *Google Cloud Platform* con el hardware descrito en la Tabla 3.4, utilizando el sistema operativo Ubuntu 18.04 y la librería *Tensorflow Object Detection API* con Tensorflow 1.15 [35].

Tabla 3.4: Hardware de la máquina virtual utilizado para el entrenamiento.

| Componente | Entrenamiento | Precio por hora [USD] |
|---|---|---|
| GPU | NVIDIA Tesla K80 | $ 0,45[a] |
| Máquina | n1-standard-8 (8 núcleos y 30GB RAM) | $ 0,38[b] |
| Almacenamiento | 200 GB | $ 0,01 |

[a] https://cloud.google.com/compute/gpus-pricing?hl=es-419#gpus
[b] https://cloud.google.com/compute/vm-instance-pricing?hl=es-419#n1_standard_machine_types

Se realiza un total de 6 entrenamientos a la arquitectura SSD con *backbone* MobileNetV2. Para todos los entrenamientos se utilizan las siguientes técnicas de aumento de datos: voltear las imágenes horizontal y verticalmente, cortar la imagen y cambiar el brillo de las imágenes, pues se ha demostrado que ayuda a mejorar el desempeño de los algoritmos [32][47].

Para los primeros dos entrenamientos se utilizan los *datasets Cubelli*, *Waste Classification* y *Trashnet*, con un total de 4.829 marcas en 2.971 imágenes y se entrena durante 20.000 y 7.000 iteraciones, respectivamente, con un tamaño de *batch* (imágenes por iteración) de 24, lo que hace que se demore aproximadamente 16 minutos cada 1.000 iteraciones. Luego, los siguientes entrenamientos se realizan con todas las bases de datos y de manera iterativa, esto quiere decir que se entrena el mismo modelo anterior pero por un mayor número de iteraciones, primero por 20.000 iteraciones, luego 10.000 más, luego 20.000 más y finalmente 10.000 más, es decir, se entrenó un modelo por 60.000 iteraciones generando 3 modelos intermedios a los cuales se les llamará por el número de iteraciones acumuladas, es decir, 20.000, 30.000, 50.000 y 60.000. Se detiene el entrenamiento en esta iteración pues el mAP en el conjunto de validación comienza a saturarse como muestra la Figura 3.17.

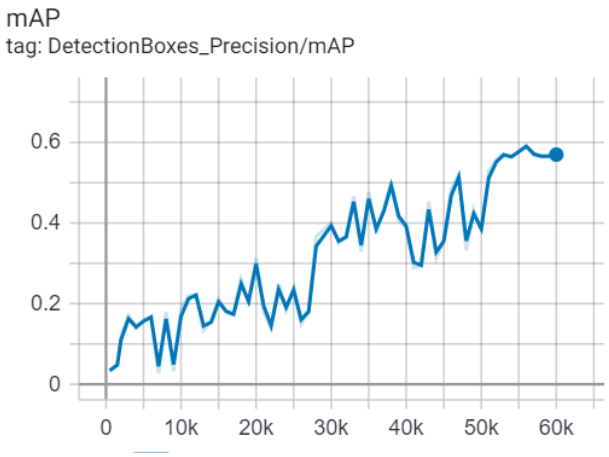


Figura 3.17: mAP en conjunto de validación a lo largo del entrenamiento.

Teniendo en cuenta el precio por hora de la máquina con los componentes descritos en la Tabla 3.4, el tiempo que toma entrenar una iteración y el número total de iteraciones, se puede calcular el costo que toma obtener el modelo. Estos costos se describen en la Tabla 3.5, donde para los modelos entrenados con todas las bases de datos se toma el precio relativo al modelo anterior. El costo total de entrenar todos estos modelos es de $15.370 [CLP].

Tabla 3.5: Costo para obtener el modelo.

| **Arquitectura de Red** | **Cantidad de imágenes** | **Cantidad de marcas** | **Número de iteraciones** | **Precio [USD]** |
|---|---|---|---|---|
| SSD MobileNetV2 | 2.971 | 4.829 | 7.000 | $ 1,56 |
| SSD MobileNetV2 | 2.971 | 4.829 | 20.000 | $ 4,48 |
| SSD MobileNetV2 | 8.662 | 15.102 | 20.000 | $ 4,48 |
| SSD MobileNetV2 | 8.662 | 15.102 | 30.000 | $ 2,24 |
| SSD MobileNetV2 | 8.662 | 15.102 | 50.000 | $ 4,48 |
| SSD MobileNetV2 | 8.662 | 15.102 | 60.000 | $ 2,24 |

## 3.5. Red en sistema *on the edge*

Para implementar la red en la NVIDIA Jetson Nano es necesario configurarla e instalar todos las librerías a utilizar, esto se hace siguiendo los tutoriales de Adrian Rosebrock [48] y Alasdair Allan [49]. Se instala el sistema operativo JetPack 4.2 y las librerías de Python Tensorflow, TensorRT y OpenCV. También se instala VNC para poder controlar la Jetson a distancia. Una de las dificultades que surgen al implementar la red en el dispositivo es la incompatibilidad de las versiones de Tensorflow. Al momento de entrenar y compilar la red se utiliza Tensorflow 1.15 pero la versión de la Jetson Nano es Tensorflow 1.13. Para solucionar este problema se vuelve a compilar la red en la versión 1.13, siguiendo el tutorial de Chengwei Zhang [50].

Una vez configurado el dispositivo se monta simulando la cinta transportadora tal como muestra la Figura 3.18, donde la Jetson Nano se encuentra en altura sobre una superficie

de vidrio transparente y con la cámara apuntando verticalmente hacia abajo. Debajo se encuentra una bolsa de basura negra de 70 centímetros de ancho por 9 metros de largo sobre la que se ponen los materiales y que se mueve manualmente. En la Figura 3.19 se muestra una de las imágenes capturadas por el dispositivo mientras los materiales se transportaban por la cinta.

Figura 3.18: Montaje real de la solución.

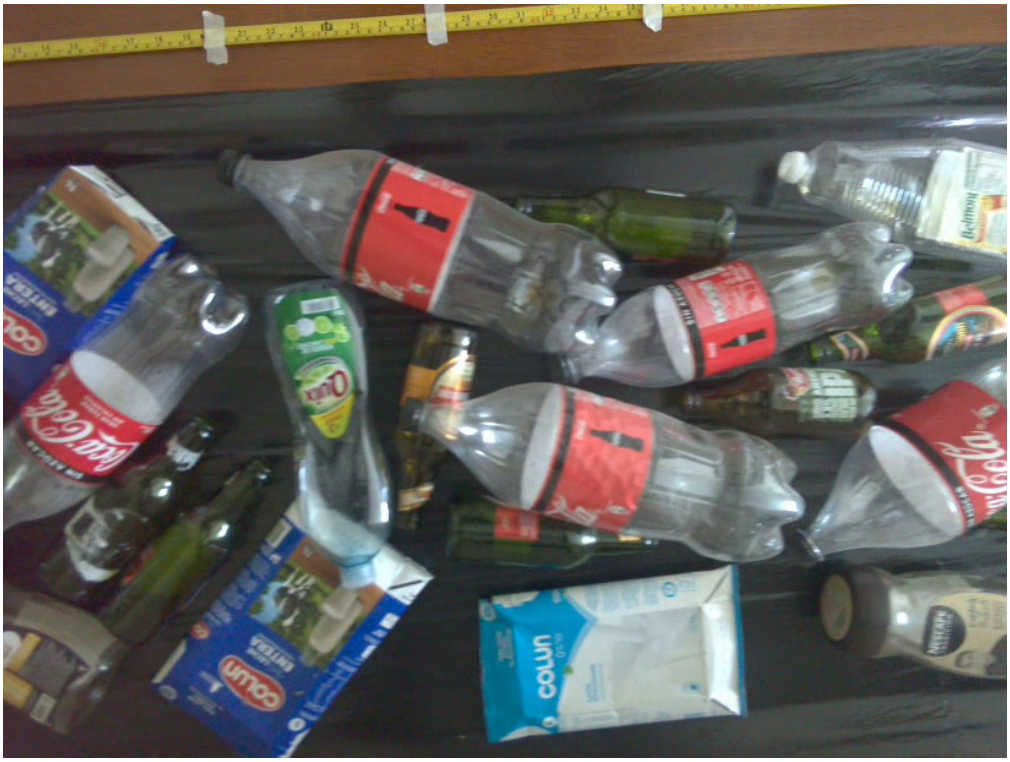


Figura 3.19: Imágenes capturadas por el dispositivo montado.

# Capítulo 4

# Resultados

## 4.1. MobileNetV2 SSD

A continuación se muestran los resultados del modelo SSD con *backbone* MobileNetV2 sobre la base de datos *La Marina*, que no fue utiliza para entrenar y presenta los elementos reciclables en un fondo negro y con vista superior. Se evalúa en base a métricas de AP por clase, mAP y una matriz de confusión para un IoU de 0,5.

### 4.1.1. Tiempo de inferencia

Los *frames* por segundo (fps) de la red en el sistema *on the edge* $9,47 \pm 0,19$ basado en 419 *frames* grabados y considerando tiempo de inferencia y filtrado de detecciones con baja confidencia y el uso de memoria en la interfaz gráfica del sistema operativo. La zona de análisis de la cámara es de 70 centímetros. La Figura 4.1 presenta un histograma de los *frames* por segundo en la NVIDIA Jetson Nano.

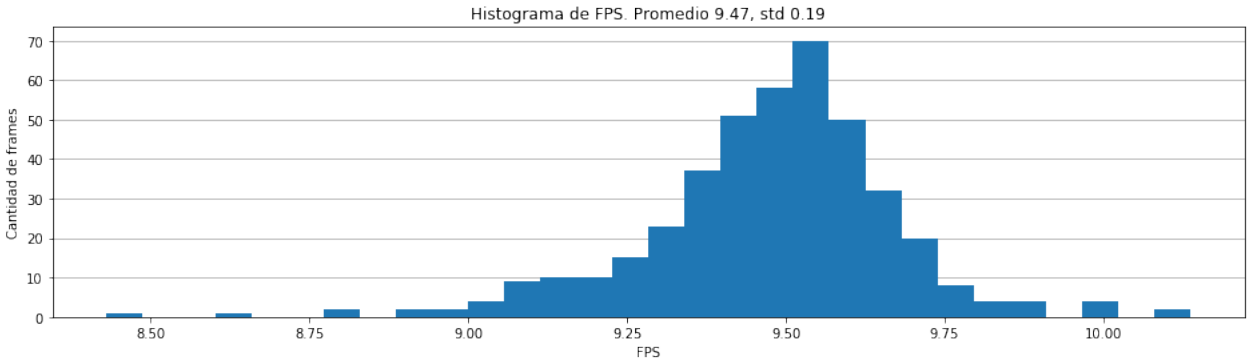


Figura 4.1: Histograma de *frames* por segundos (FPS) en NVIDIA Jetson Nano.

### 4.1.2. mAP

Se observa en el mAP de la Tabla 4.1 que a medida que hay más iteraciones mejor desempeño obtiene el modelo sobre el conjunto de test. En la Figura 4.2 y la Tabla 4.2 y se observa que para cada clase el mayor AP se obtiene en el modelo con 60.000 iteraciones, superando los 0,5 en todas las clases.

Tabla 4.1: AP y mAP por cada modelo. En negrita se destaca el modelo que obtiene el mejor resultado.

| **Modelo** | $AP_{50}$ | $AP_{75}$ | $mAP$ |
|---|---|---|---|
| SSD MobileNetV2 Parcial 7.000 | 0,067 | 0,007 | 0,028 |
| SSD MobileNetV2 Parcial 20.000 | 0,025 | 0,013 | 0,011 |
| SSD MobileNetV2 Total 20.000 | 0,414 | 0,343 | 0,279 |
| SSD MobileNetV2 Total 30.000 | 0,510 | 0,471 | 0,339 |
| SSD MobileNetV2 Total 50.000 | 0,450 | 0,390 | 0,341 |
| SSD MobileNetV2 Total 60.000 | **0,675** | **0,645** | **0,530** |

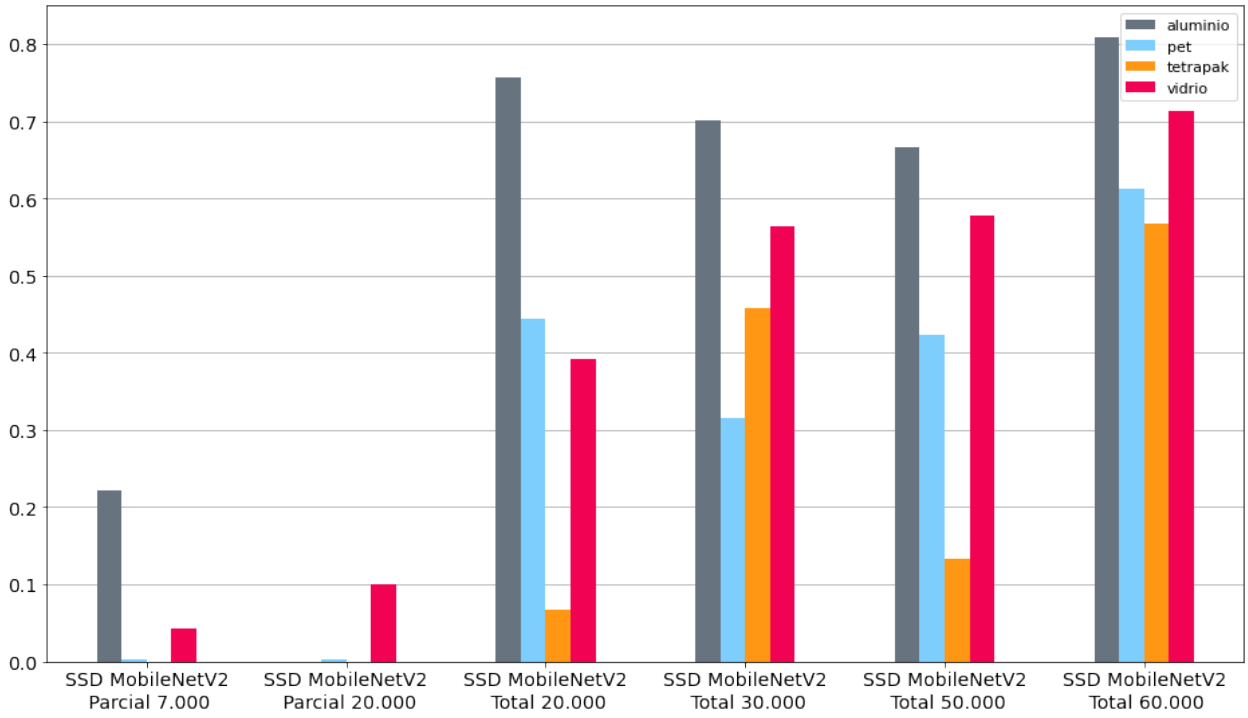


Figura 4.2: AP por clase para cada modelo.

Tabla 4.2: AP por cada clase. En negrita se destaca el modelo que obtiene mejor resultado.

| | **7000 Parcial** | | | **20000 Parcial** | | | **20000 Total** | | |
|---|---|---|---|---|---|---|---|---|---|
| **Clase** | **AP** | **AP50** | **AP75** | **AP** | **AP50** | **AP75** | **AP** | **AP50** | **AP75** |
| Aluminio | 0,088 | 0,222 | 0,000 | 0,000 | 0,000 | 0,000 | 0,557 | 0,757 | 0,757 |
| PET | 0,000 | 0,003 | 0,000 | 0,000 | 0,001 | 0,000 | 0,262 | 0,443 | 0,280 |
| Tetrapak | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,055 | 0,066 | 0,066 |
| Vidrio | 0,025 | 0,040 | 0,030 | 0,046 | 0,099 | 0,054 | 0,242 | 0,391 | 0,245 |
| | **30000 Total** | | | **50000 Total** | | | **60000 Total** | | |
| **Clase** | **AP** | **AP50** | **AP75** | **AP** | **AP50** | **AP75** | **AP** | **AP50** | **AP75** |
| Aluminio | 0,477 | 0,702 | 0,702 | 0,577 | 0,666 | 0,666 | **0,640** | **0,809** | **0,809** |
| PET | 0,193 | 0,316 | 0,234 | 0,294 | 0,423 | 0,333 | **0,454** | **0,612** | **0,527** |
| Tetrapak | 0,320 | 0,458 | 0,386 | 0,110 | 0,133 | 0,133 | **0,462** | **0,566** | **0,566** |
| Vidrio | 0,366 | 0,563 | 0,464 | 0,383 | 0,577 | 0,429 | **0,563** | **0,713** | **0,675** |

### 4.1.3. Curva *precision-recall*

Las curvas de *precision-recall* de la Figura 4.3 muestran como las clases vidrio y PET decaen lentamente y el aluminio decae rápidamente al final de la curva. Esto ocurre porque el aluminio tiene tan solo 6 ejemplos.

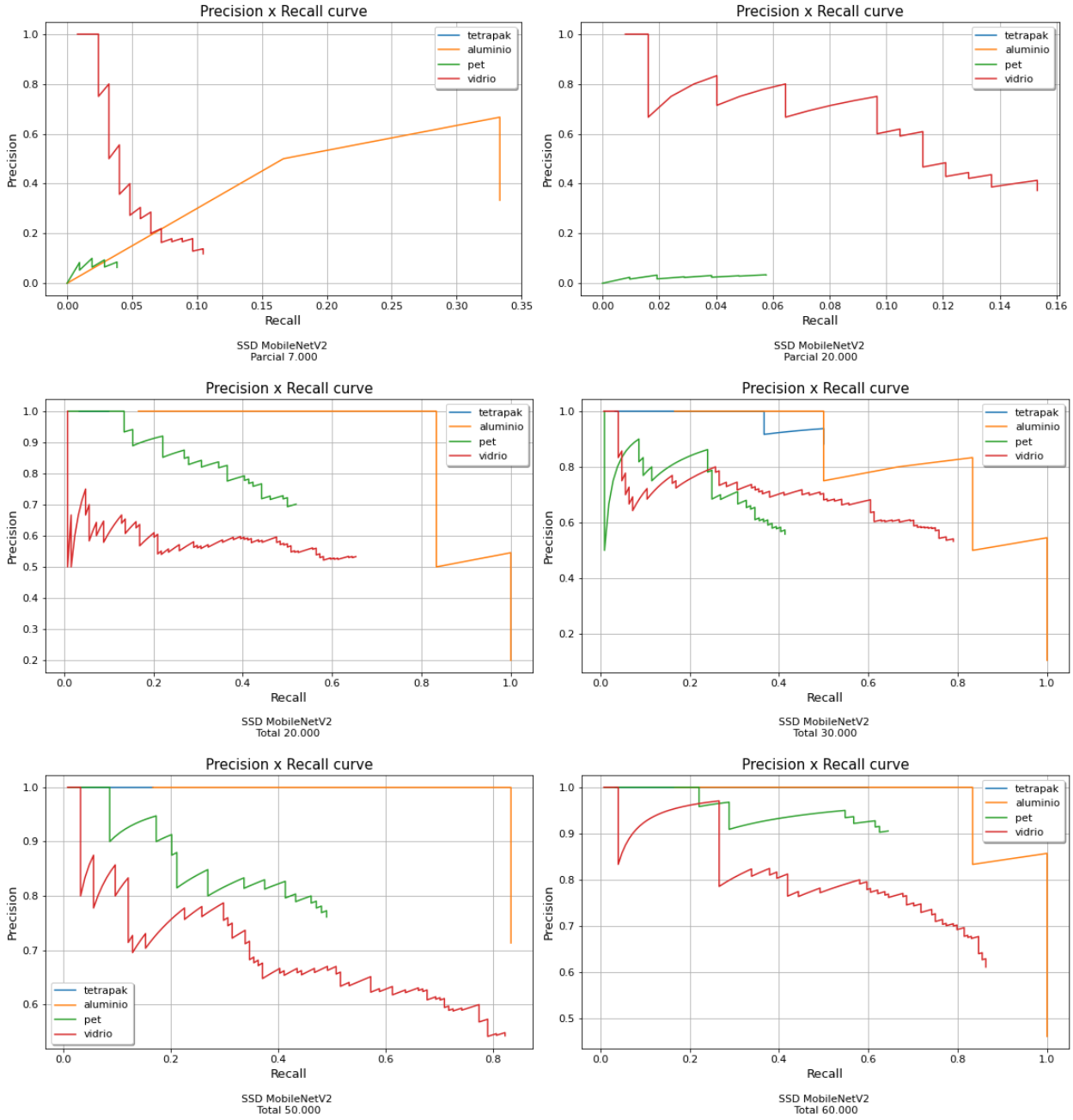


Figura 4.3: Curvas de *precision-recall* por clase para cada modelo.

### 4.1.4. Matriz de confusión

En las matrices de confusión de la Figura 4.4 se observa que a medida que aumentan las iteraciones en el modelo se van resolviendo los errores entre clases, a excepción de PET y vidrio.

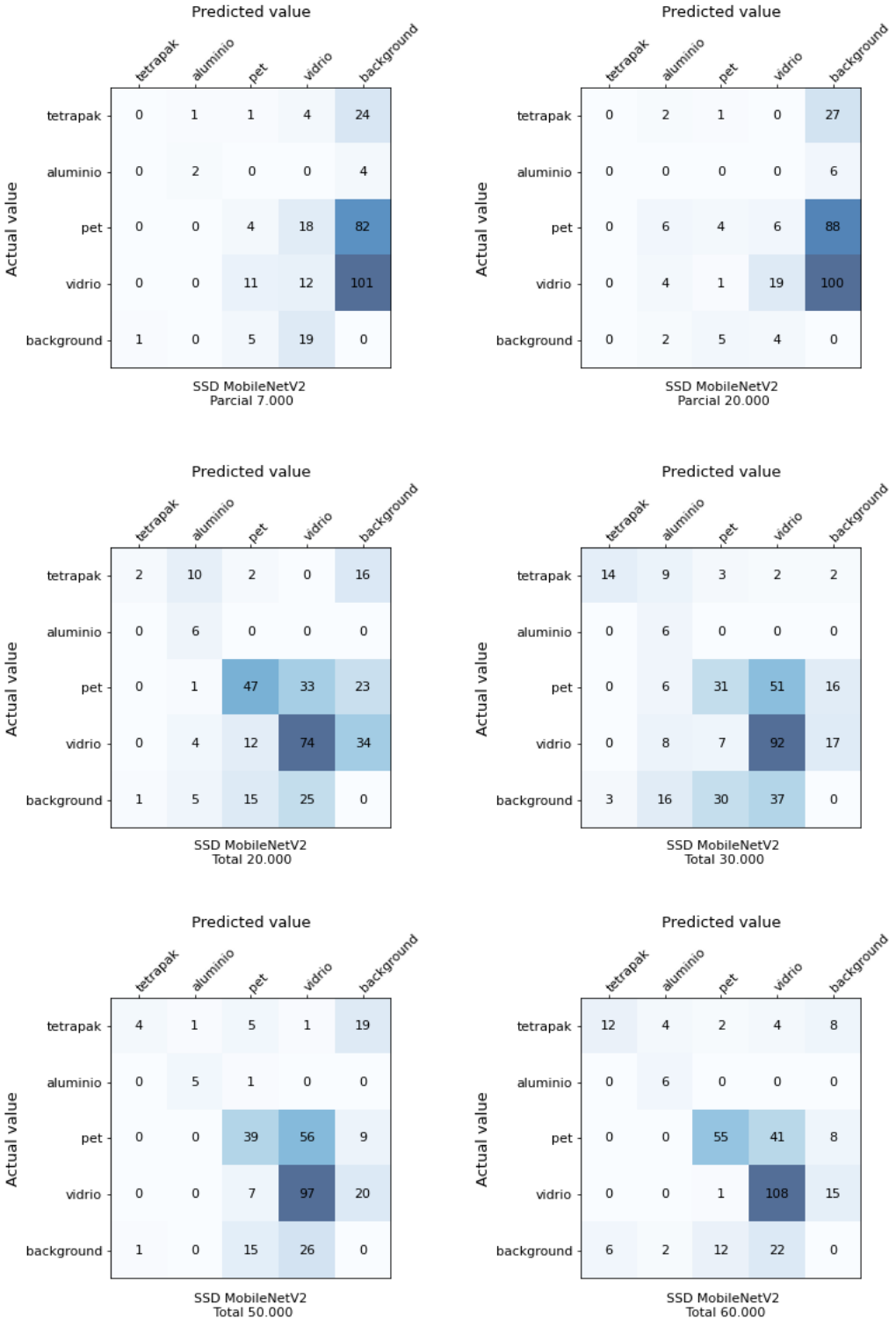


Figura 4.4: Matriz de confusión para cada modelo.

# Capítulo 5

# Análisis de resultados

A partir de los resultados de velocidad se establece que el sistema funciona a una velocidad superior a la necesaria para la aplicación, soportando una velocidad de cinta de hasta 23 kilómetros por hora (6,38 metros por segundo), siendo las velocidades recomendadas para la separación manual cercano a los 0,6 kilómetros por hora [1].

Observando el resto de los resultados se establece que los modelos que no fueron entrenados con todos los datos, es decir, *SSD MobileNetV2 Parcial 7.000* y *SSD MobileNetV2 Parcial 20.000*, obtienen resultados realmente pobres, con un máximo de 0.028 mAP. En la Figura 4.4 se observa que para estos modelos hay una alta cantidad de falsos negativos (elementos no detectados). Esto indica que la cantidad y calidad de datos son muy distintos a los de *La Marina*.

Analizando la base de datos *Cubelli* se observa en la Figura 5.1 que las imágenes son muy similares, con una pequeña rotación de los elementos, por lo que el modelo puede estar aprendiendo a memorizar estos elementos más que aprender características que permitan distinguir entre una clase y otra. Por otro lado tanto *Waste Classification* como *Trashnet* están compuestas de imágenes con fondo blanco, lo que puede estar afectando la detección en fondo negro. En las figuras 5.2 y 5.3 se observan ejemplos de estos dos datasets, respectivamente.

Figura 5.1: Imágenes similares del *dataset Cubelli*.

[1] https://the-mkgroup.com/portfolio/picking-belts/


Figura 5.2: Imágenes similares del *dataset Waste Classification*.

Figura 5.3: Imágenes con fondo blanco del *dataset Trashnet*.

Para el resto de los modelos se puede ver que a medida que aumentan las iteraciones de entrenamiento aumenta el mAP. Cabe destacar que todos se desempeñan significativamente mejor que los modelos *Parcial*, esta mejora esta asociada a la mayor cantidad de datos de *OIDV4*, *COCO* y *Tagias*. El mejor modelo, es decir, el que tiene mayor mAP, es *SSD MobileNetV2 Total 60.000*, obteniendo también el mejor AP por clase.

A partir de las matrices de confusión se puede establecer que para el mejor modelo se confunden los plásticos PET con vidrios en 41 ocasiones, el caso inverso, confundir vidrios con PET, ocurre sólo una vez. Esto puede deberse, principalmente, al desbalance de clases presente en el dataset completo, donde los vidrios predominan con 6.481 etiquetas versus las 3.958 de PET y el hecho de que ambos materiales son transparentes. También se observa que el tetrapak se confunde con todo el resto de las clases, lo que puede ocurrir dado que es la clase que cuenta con menos ejemplos. Además, la matriz de confusión muestra que hay 42 falsos positivos y 31 falsos negativos.

En la Figura 5.4 se muestran algunas de las detecciones erróneas mencionadas anteriormente. Por otro lado, la curvas de *precision* y *recall* de la Figura 4.3 muestran como, a medida que aumenta el *recall*, disminuye la *precision*, por sobretodo en las clases vidrio y PET. Esto quiere decir que existen varias detecciones con valores bajos de confianza, ya que al aumentar el umbral de confianza hay más falsos positivos.

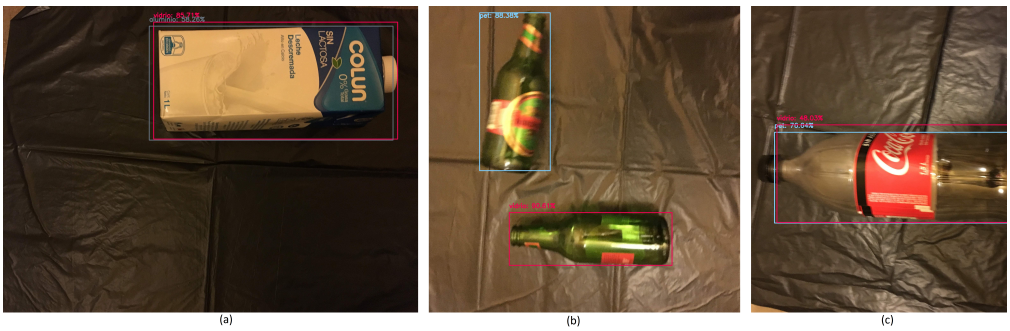

Figura 5.4: Detecciones erróneas. (a) Detecciones de vidrio y aluminio sobre una caja de tetrapak, (b) detección de PET sobre una botella de vidrio y (c) detección de vidrio y PET sobre una botella de PET.

Para evaluar de mejor manera se puede ver la Figura 5.5 que muestra los valores de confianza de las detecciones separados por clase. Se observa que de las 280 detecciones totales hay 43 con valores de confianza por debajo de 0,5: 20 son de vidrio, 14 de PET, 7 de tetrapak y 2 de aluminio. Llama la atención el tetrapak pues estas 7 detecciones representan un 38,9 % del total de detecciones de tetrapak. Los valores bajos de confianza se deben, fundamentalmente, a la baja cantidad de ejemplos de tetrapak en el entrenamiento. Por otro lado, el vidrio es la clase que posee el mayor porcentaje de detecciones con un valor superior a 0,9 de *confidence*, con un 61,7 % de las detecciones, al contrario del tetrapak, esto ocurre porque hay una alta cantidad de ejemplos de vidrio.

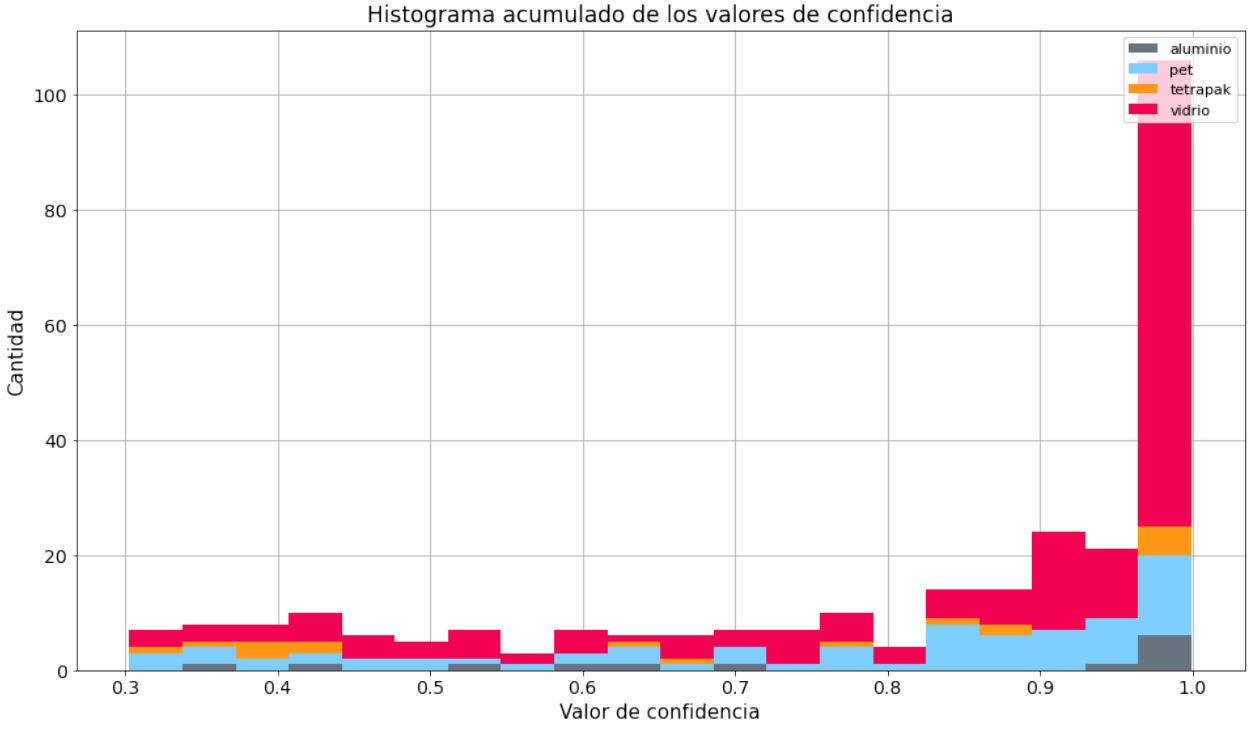


Figura 5.5: Histograma de los valores de confidencia de las detecciones sobre las imágenes de *prueba*.

En la Figura 5.6 se observa el efecto de aumentar el umbral de confianza para considerar una detección como efectiva. A medida que el umbral aumenta, disminuyen los falsos positivos

pero aumentan a su vez los falsos negativos, es decir, deja de detectar elementos, tanto equivocados como correctos. Aún con un umbral de 0,9 hay errores entre PET y vidrio, esto ocurre tanto para botellas transparentes como verdes, tal como muestra la Figura 5.7. Estas clases son conflictivas debido a su parecido y se deben añadir más ejemplos para que la red pueda distinguirlas.

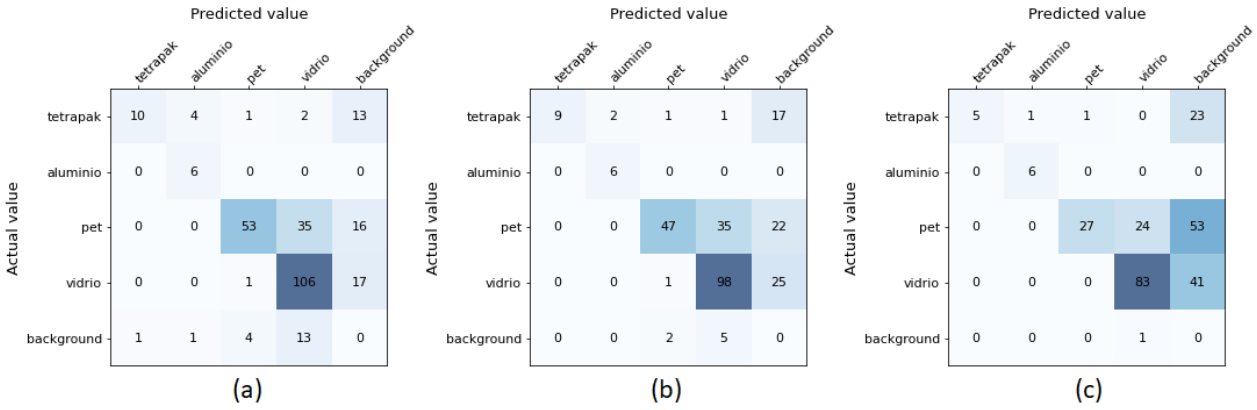


Figura 5.6: Matrices de confusión a distintos umbrales. (a) umbral de confidencia de 0,5, (b) umbral de confidencia de 0,7 (c) umbral de confidencia de 0,9.


Figura 5.7: Ejemplos de errores en *La Marina* con confianza mayor a 0,9.

Se analizó la base de datos utilizada para entrenar respecto a los tamaños de *bounding boxes* y la relación de aspecto de estas. En la Figura 5.8 se pueden ver los tamaños de las *bounding boxes* en altura y ancho relativo a la altura y ancho total de la imagen, separados por etiqueta y por base de datos. Se puede apreciar que los *datasets* que tienen *bounding boxes* con tamaños relativos en el cuadrante más pequeño (menor al 0.2) en altura y ancho son *OIDV4*, *COCO*, *Waste Classification*, *Tagias* y Trashnet ,y representan a todas las clases. Por otro lado, las bases de datos que tienen *bounding boxes* con tamaños relativos en el cuadrante más

grande (mayor a 0.8) son *Cubelli*, *OIDV4*, *Waste Classification* y *Trashnet*, y representan a todas las clases menos tetrapak.

$$RS = \sqrt{\frac{w_{boundingbox} * h_{boundingbox}}{w_{imagen} * h_{imagen}}} \tag{5.1}$$

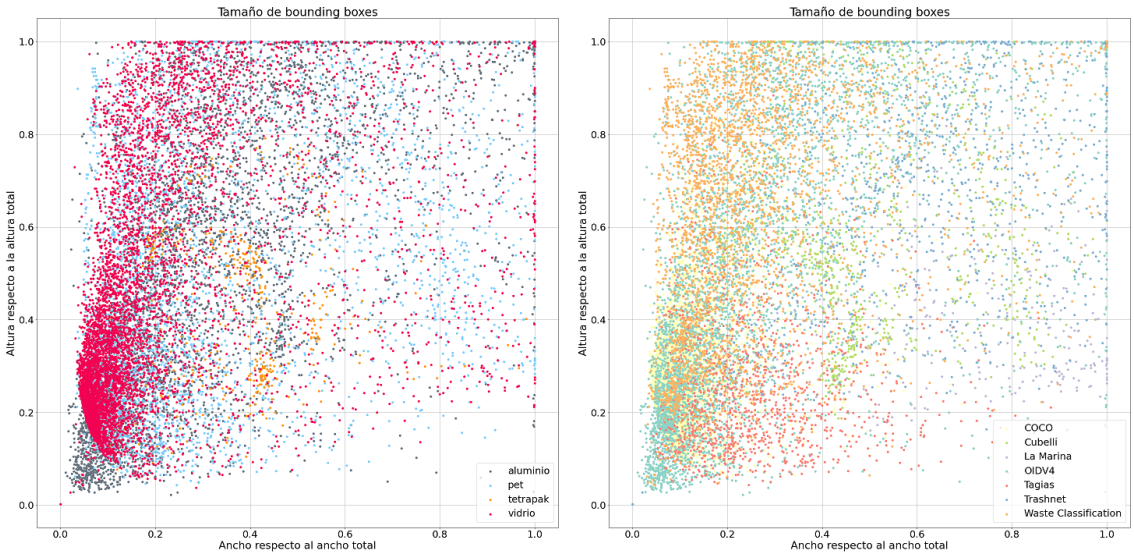


Figura 5.8: Tamaño de las *bounding boxes* en el *dataset* de entrenamiento, separadas por etiqueta (izquierda) y fuente de datos (derecha).

Estos tamaños también se encuentran representados en la Figura 5.9, donde se muestra un histograma y gráfico de caja con el tamaño relativo calculado como lo representa la ecuación (5.1). Se observa que los tamaños del aluminio son por lo general mayor, los del vidrio presentan gran cantidad de *outliers* de gran tamaño, el PET también presenta *outliers* de gran tamaño y el tetrapak tiene una dispersión menor con unos pocos *outliers* pequeños y grandes. Al observar en las imágenes los tamaños que se escapan de la norma, tanto grandes como pequeños, se observa que hay marcas erróneas. En la Figura 5.10 se muestran cuatro ejemplos de marcas pequeñas que no corresponden a ningún material. Corregir estas marcas puede mejorar el desempeño de la red.

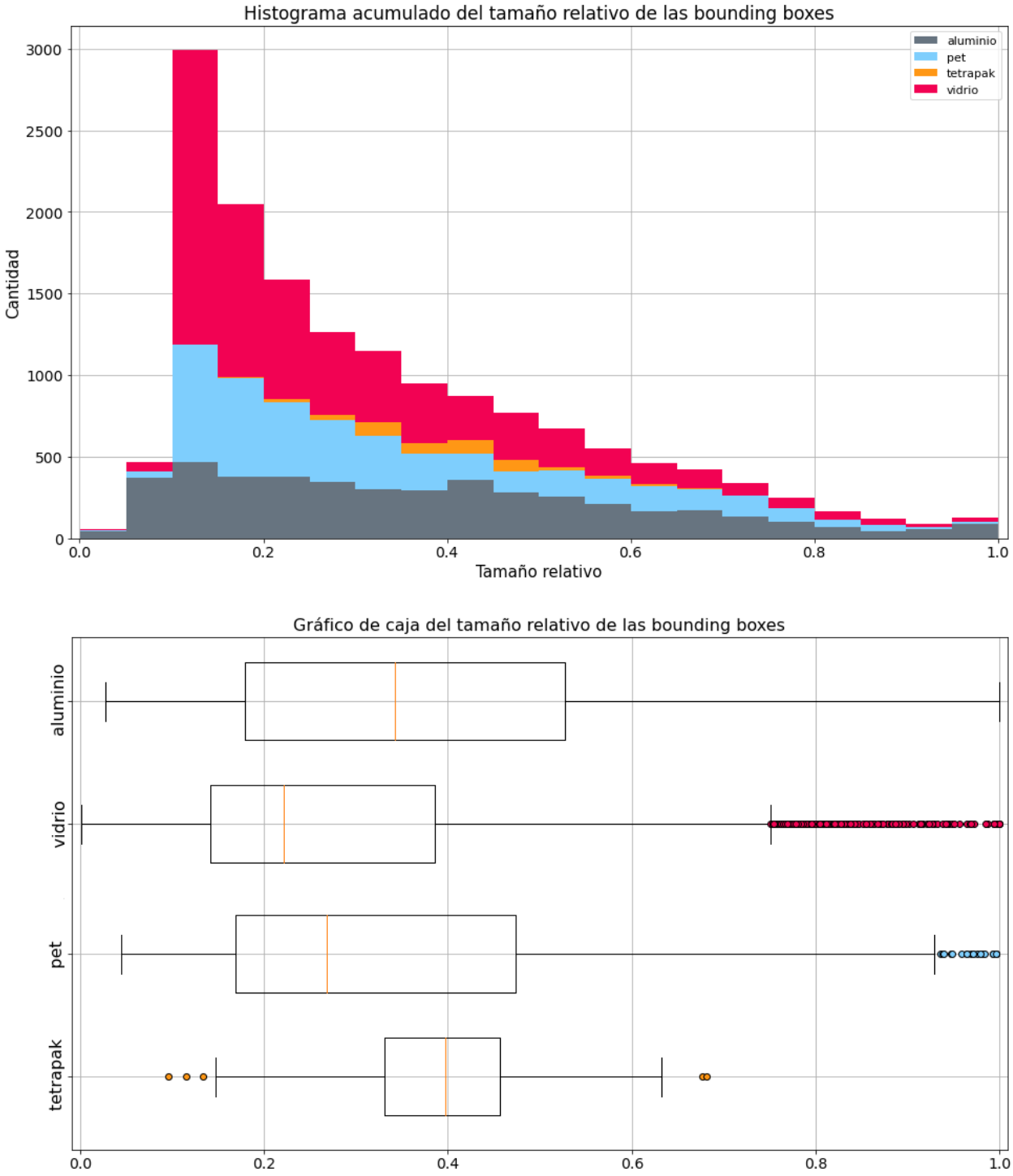


Figura 5.9: Histograma (arriba) y gráfico de caja (abajo) del tamaño relativo de las *bounding boxes*.


Figura 5.10: Imágenes con tamaño relativo pequeño y marcas erróneas. En (a) y (b) se observan dos ejemplos de marcas de aluminio de la base de datos *OIDV4* y en (c) y (d) se muestran dos imágenes con ejemplos de vidrio mal etiquetados.

Además del tamaño, se analiza también la relación de aspecto de la caja, calculado según la ecuación (5.2). Así, se pueden detectar cajas que son muy "delgadas" o "alargadas". La Figura 5.11 muestra un histograma y un gráfico de caja de las relaciones de aspecto de las *bounding boxes*, se observa que el aluminio posee una alta cantidad de *outliers* con una relación de aspecto muy grande, también presentan *outliers* el vidrio y el PET. Al revisar estas imágenes se observan elementos ocluidos a los que se les alcanza a ver una pequeña parte como muestra la Figura 5.12, y grupos de objetos marcados como un solo elemento como muestra la Figura 5.13. Estas son marcas erróneas que no debiesen estar presentes al momento del entrenamiento ya que dificultan el proceso de aprendizaje de la red.

$$AR = \frac{w_{boundingbox}}{h_{boundingbox}} \tag{5.2}$$

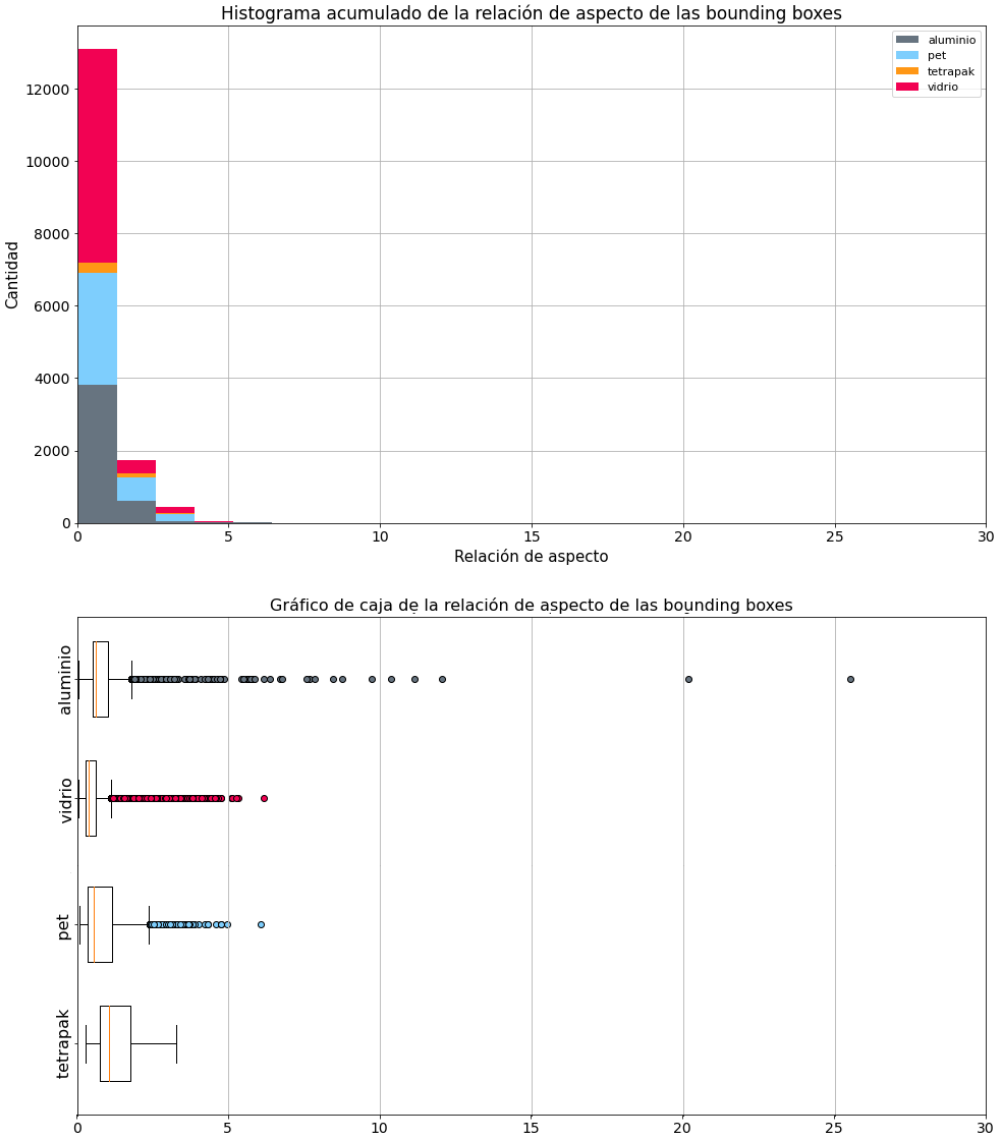


Figura 5.11: Histograma (arriba) y gráfico de caja (abajo) de la relación de aspecto de las *bounding boxes*.


(a) (b)

Figura 5.12: Ejemplos de imágenes con relación de aspecto pequeño (menor a 0.1). En (a) Se muestra una botella de PET ocluida por otra y en (b) se muestra una lata de aluminio truncada por el término de la imagen.


(a) (b)

Figura 5.13: Ejemplo de imágenes con relación relación de aspecto grande, en (a) se muestran dos grupos de latas contado como dos marcas de aluminio y en (b) un grupo de envases de vidrio marcados como un elemento de vidrio

# Capítulo 6

# Conclusiones

En este trabajo se avanza en la automatización del proceso de separación de materiales reciclables. Como una primera aproximación se aborda el problema de la detección de estos materiales. Para esto, se implementa una red neuronal convolucional en un sistema *on the edge* que es capaz de detectar cuatro tipos de materiales reciclables presentes en la basura domiciliaria: vidrio, PET, tetrapak y aluminio.

Se presenta una metodología que permite resolver el problema planteado de principio a fin, desde el diseño del montaje y la adquisición de imágenes, hasta la implementación en el sistema *on the edge*. La resolución del problema se hace de manera iterativa (captura, entrenamiento e implementación), lo que permite que la solución encontrada pueda seguir mejorando con el tiempo.

Para lograr el objetivo, se crea un *dataset* de imágenes de materiales reciclables, etiquetados con *bounding boxes* para lo que se utilizan distintos métodos de recopilación de imágenes. Uno de estos métodos es la exploración de bases de datos abiertas de detección de objetos, y ampliamente utilizadas, como *COCO* y *Open Images*. También, se exploran bases de datos abiertas de clasificación de imágenes que son cercanas al problema abordado, estás son: *Waste Classification* y Trashnet, que buscan clasificar materiales reciclables de no reciclables y el tipo de material, respectivamente.

Otro método utilizado es el *crowdsourcing*, donde se les solicita a estudiantes de la Universidad de Chile que envíen imágenes de materiales reciclables. Todos estos métodos son gratuitos pero consumen una enorme cantidad de tiempo, en particular el etiquetado (*bounding box* y clase) es más lento que el reetiquetado (solo clase). Finalmente, el último método para obtener imágenes es contratar a una empresa especializada en la recopilación y etiquetado de datos. Este método es más rápido ya que la empresa se especializa en eso, pero tiene un coste monetario. Se presenta, también, una herramienta de etiquetado de imágenes, *Labelbox*, disponible de manera gratuita para estudiantes que es utilizada para etiquetar y reetiquetar gran parte de las imágenes.

Por otro lado, se entrena la red neuronal con los datos recopilados. Esto se realiza utilizando el *framework* abierto *Tensorflow Object Detection API* donde se encuentra el modelo *SSD MobileNetV2* preentrenado en *COCO*. Para realizar el entrenamiento de manera más rápida se utiliza como hardware una máquina virtual utilizando Google Cloud Platform.

Una de las mayores limitaciones del trabajo es el dispositivo *on the edge* utilizado. Este presenta restricciones respecto a la cámara que se puede utilizar y respecto a la red neuronal, ya que esta debe ser liviana, lo que implica una pérdida en la precisión. Otra limitante del trabajo es el contexto internacional de pandemia, en particular la cuarentena no permite el trabajo en terreno de captura de imágenes.

Los resultados del trabajo muestran que se logra cumplir el objetivo planteado, ya que se detectan los materiales en la simulación experimental del montaje con un mAP de 0,53 y con un costo efectivo del proyecto de $293.374 [CLP], mayoritariamente en el hardware utilizado para la detección *on the edge*. El análisis de los resultados muestra que también se puede seguir mejorando el modelo a través de la revisión y corrección de los datos marcados, la obtención de más datos y la exploración de otras técnicas de aumento de datos.

Respecto de la aplicación final del sistema, los resultados muestran que en términos de velocidad de inferencia el dispositivo puede funcionar perfectamente a las velocidades típicas de las plantas de reciclaje o teniendo un amplio margen para mejorar precisión perdiendo velocidad de inferencia. Por otro lado, con la tasa de error actual, el sistema puede ser implementado en una planta de reciclaje pero no como un sistema autónomo, tiene que ser complementado con otras formas de clasificación, ya sea manual o automático, para aumentar su porcentaje de recuperación. Como comentario final, este trabajo demuestra que es factible implementar captura e identificación de materiales reciclables en tiempo real (para esta aplicación) con un costo relativamente bajo.

## 6.1. Trabajo futuro

Existen varias maneras de continuar con este trabajo experimental para obtener mejores resultados. Algunas de ellas son:

- Tomar en cuenta el análisis realizado y corregir los errores encontrados en las marcas ya existentes.
- Etiquetar completamente *OIDV4* para botellas de vidrio, plásticos y aluminio.
- Utilizar las imágenes del *dataset TACO* de forma de tener una base de datos más amplia.
- Implementar el diseño del montaje experimental en un centro de acopio y/o planta de reciclaje donde permanentemente hay materiales reciclables, lo que permitiría la rápida captura de imágenes.
- Añadir más tipos de materiales reciclables, ya sean otros plásticos o papeles y cartones.
- Entrenar, con la base de datos generada, otra arquitectura de red con mejor desempeño para premarcar imágenes, agilizando el proceso de marcado pues para esta etapa no es necesario una inferencia rápida.
- Explorar otras técnicas de recopilación de imágenes como *scrapping* de redes sociales tales como Instagram.
- Utilizar hardwares *on the edge* más poderosos que ofrece la línea Jetson de NVIDIA tales como Jetson Xavier NX y Jetson AGX Xavier, que permitirían implementar redes

del estado del arte como EfficientDet y YOLOv4, que tienen una mayor precisión pero necesitan de más recursos computacionales para funcionar a las velocidades necesarias para implementarse en una línea industrial.

# Bibliografía


[1] P. T. Espinoza, E. M. Arce, D. Daza, M. S. Faure, and H. Terraza, "Informe de la evaluación regional del manejo de residuos sólidos urbanos en américa latina y el caribe 2010," 2010.

[2] C. M. del Medio Ambiente, "Ley 20.920: ley marco para la gestión de residuos, la responsabilidad extendida del productor y fomento al reciclaje," 2016.

[3] M. del Medio Ambiente, "Cuarto reporte del estado del medio ambiente," 2018.

[4] CONAMA, "Primer reporte sobre manejo de residuos sólidos en chile," 2010.

[5] I. e Inmobiliaria Huaiquilaf Ltda., "Catastro nacional de instalaciones de recepción y almacenamiento, e instalaciones de valorización de residuos en chile," 2018.

[6] K. Ip, M. Testa, A. Raymond, S. C. Graves, and T. Gutowski, "Performance evaluation of material separation in a material recovery facility using a network flow model," *Resources, Conservation and Recycling*, vol. 131, pp. 192 – 205, 2018.

[7] R. Bogue, "Robots in recycling and disassembly," *Industrial Robot: the international journal of robotics research and application*, vol. ahead-of-print, 07 2019.

[8] Y. Han, X. Wang, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *CoRR*, vol. abs/1907.08349, 2019.

[9] G. Thung and M. Yang, "Classification of trash for recyclability status," 2017.

[10] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: real-time instance segmentation," *CoRR*, vol. abs/1904.02689, 2019.

[11] D. P. Papadopoulos, J. R. R. Uijlings, F. Keller, and V. Ferrari, "Extreme clicking for efficient object annotation," *CoRR*, vol. abs/1708.02750, 2017.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[13] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1 – 17, 1964.

[14] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.

[15] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, 01 2014.

[18] W. A. Giovinazzo, "Overfitting/underfitting - how well does your model fit?," 05 2017.

[19] D. M. Hawkins, *Identification of Outliers*. Monographs on Applied Probability and Statistics, Springer, 1980.

[20] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," *The VLDB Journal*, vol. 8, p. 237–253, 02 2000.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 11 1998.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[23] K. Patel, "Convolutional neural networks — a beginner's guide," 07 2019.

[24] K. Bai, "A comprehensive introduction to different types of convolutions in deep learning," 02 2019.

[25] F. F. Li and A. Karpathy, "Cs231n: Convolutional neural networks for visual recognition," 2015.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[27] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[28] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[29] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[30] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[31] A. Ouaknine, "Review of deep learning algorithms for object detection," 02 2018.

[32] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[33] J. Hale, "Deep learning framework power scores 2018," 09 2018.

[34] Y. Wu, L. Liu, C. Pu, W. Cao, S. Sahin, W. Wei, and Q. Zhang, "A comparative measurement study of deep learning as a service framework," *CoRR*, vol. abs/1810.12210, 2018.

[35] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *CoRR*, vol. abs/1611.10012, 2016.

[36] S. L. N. Rafael Padilla and E. A. B. da Silva, "Survey on performance metrics for object-detection algorithms," 2020.

[37] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[38] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[40] A. Gupta, P. Dollar, and R. Girshick, "LVIS: A dataset for large vocabulary instance segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[41] M. Liberman, "Reproducible research and the common task method," 2015.

[42] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–308, 09 2009. Printed version publication date: June 2010.

[43] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[44] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *arXiv:1811.00982*, 2018.

[45] P. F. Proença and P. Simões, "Taco: Trash annotations in context for litter detection," *arXiv preprint arXiv:2003.06975*, 2020.

[46] Labelbox, "Labelbox," *Online*, 2020.

[47] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[48] A. Rosebrock, "How to configure your nvidia jetson nano for computer vision and deep learning," 03 2020.

[49] A. Allan, "Getting started with the nvidia jetson nano developer kit," 04 2019.

[50] C. Zhang, "How to run tensorflow object detection model on jetson nano," 05 2019.

# Anexo A

# Presupuesto y compras

Una vez realizada la selección de los componentes, estos son adquiridos y se espera el tiempo de envío correspondiente. En la Tabla A.1 se comparan los precios de los componentes en la plataforma internacional *Mouser* y la plataforma nacional *Ingeniería MCI*. El precio en el mercado nacional incluye el IVA y no incluye costos de envío[1]:

Tabla A.1: Cotización de los componentes en el mercado nacional e internacional.

| **Componente Componente** | **Precio mercado internacional [USD]** | **Precio mercado nacional [CLP]** |
|---|---|---|
| NVIDIA Jetson Nano Developer Kit | $ 123,75 | $ 149.990 |
| Raspberry Pi Camera Module V2 | $ 37,44 | $ 42.990 |
| Fuente de alimentación 5V 4A | - | $ 10.290 |
| Tarjeta microSD UHS-1 64 GB | - | $ 19.990 |

[1] Precios consultados el 29 de diciembre de 2019

# Anexo B

# Tabla de precios etiquetado imágenes

Tabla B.1: Cotización de los componentes en el mercado nacional e internacional.

| **Tarea** | **Precio Julio 2020 [USD]** | **Precio Septiembre 2020 [USD]** |
|---|---|---|
| Recolección | $ 0.1 | $ 0.15 |
| *Bounding box* | $ 0.01 | $ 0.03 |
| Clasificación | $ 0.004 | $ 0.02 |
| Polígono | $ 0.025 | $ 0.08 |