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

CREACIÓN DE INVENTARIO DE LUMINARIAS DE CARRETERA MEDIANTE LA IMPLEMENTACIÓN DE ALGORITMOS DE DETECCIÓN, CLASIFICACIÓN, SEGUIMIENTO Y GEOLOCALIZACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

JOSÉ MANUEL ÁLVAREZ ROMERO

PROFESOR GUÍA:
GONZALO SANDOVAL PALMA

MIEMBROS DE LA COMISIÓN:
ANDRÉS CABA RUTTE
ÁLVARO SILVA MADRID

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: **JOSÉ MANUEL ÁLVAREZ ROMERO**
FECHA: 2023
PROF. GUÍA: GONZALO SANDOVAL

CREACIÓN DE INVENTARIO DE LUMINARIAS DE CARRETERA MEDIANTE LA IMPLEMENTACIÓN DE ALGORITMOS DE DETECCIÓN, CLASIFICACIÓN, SEGUIMIENTO Y GEOLOCALIZACIÓN

La automatización de procesos ha tenido un gran impacto en múltiples industrias, reduciendo costos y tiempo que diversas tareas costaban. La empresa *Apsa Ltda* actualmente realiza sus inventarios de carretera mediante un proceso semi-automático, para optimizar los tiempos de procesamiento *Apsa Ltda* busca automatizar este proceso, con estos antecedentes surge el tema de memoria *creación de un inventario de luminarias de carretera mediante la implementación de algoritmos de detección, seguimiento y geolocalización*. Esta memoria es una etapa del proceso de automatización final que busca conseguir *Apsa Ltda*.

Para una realización exitosa de la memoria, se desarrolló una metodología que permite resolver el problema simplificado de manera completa, desde el pre procesado de datos, evaluación y entrenamiento de modelos, junto con un post-procesamiento que consiste en obtener la información importante de los modelos utilizados y exportarlo a un DataFrame, el cual en formato Excel representará el inventario de la ruta analizada.

Los modelos utilizados fueron los siguientes, para el problema de detección de objetos se utilizó el algoritmo YOLOv7, para el seguimiento de objetos, se utilizó StrongSORT, para la geolocalización de objetos se utilizó la librería GeoPandas y GeoPy, finalmente para la exportación de inventario en formato Excel se utilizó la librería pandas.

Los resultados muestran que YOLOv7 pose un *mean average precision*(mAP) con umbral de 0.5 por sobre el 95 %, los errores en la mayoría de los casos son compensados por el algoritmo StrongSORT, pero este algoritmo de vez en cuando también comete errores y asigna identificadores equivocados.

Para evaluar el tiempo de cómputo del código general que junta todos los algoritmos se utilizó una CPU *Ryzen 7 1700X* para evaluar el código sin aceleración mediante GPU y una GPU *NVIDIA Tesla T4* para comparar la ganancia de tiempo al utilizar la aceleración de GPU que habilita PyTorch CUDA para las detecciones de YOLOv7. Cuando se utiliza aceleración mediante GPU el tiempo de computo es aproximadamente $\frac{1}{5}$ del tiempo de computo comparado con solo utilizar la CPU para ejecutar el código.

Finalmente se logró entregar el inventario del proceso de automatización simplificado que *Apsa Ltda* buscaba, el proyecto tiene un amplio rango de mejora mediante la obtención de más data, ampliación de elementos a detectar, profundización en los algoritmos de geolocalización y la creación de una aplicación para facilitar el uso e interacción con el usuario.

Agradecimientos

Quiero partir agradeciendo a mi padres por todo el apoyo que me han dado, la motivación cuando la universidad de ponía cuesta arriba, escucharme cuando las cosas salían mal y alegrarse conmigo cuando salían bien, agradezco que me hayan otorgado con esfuerzo una gran educación, la cual llevaré conmigo toda la vida. A mi Abu por escucharme y aconsejarme todos estos años, por compartir conmigo su sabiduría cuando me sentía perdido y abrumado. También a mis tatas por compartir y enseñarme cosas desde pequeño, que en paz descansen.

A mis amigos del colegio por siempre darnos un tiempo para salir y relajarnos a lo largo del semestre, mi buen amigo Nani por motivarme a regresar a hacer deporte en los últimos años de la universidad y establecer un equilibrio entre estudiar/trabajar y vida personal, gracias por las charlas entre los sets y distraerme para que puedas descansar más rato, agradecer a mi gran amigo Nachín que lo conocí gracias a Inducción, gracias por ser un pilar a lo largo de muchos ramos de eléctrica, gracias por escucharme hablar estupideces y reírte conmigo, agradecer a Feña por motivarme y empujarme a ser un mejor profesional, por compartir risas conmigo en momentos difíciles, gracias Fesa que te conocí siendo mi ayudante de Laboratorio de Ing eléctrica y nos hicimos buenos amigos, por esos ARAM nocturnos y hablar de la vida. Gracias a cada persona que fue parte de mi vida universitaria, muchas gracias.

Agradecer también a Gonzalo Sandoval y *Apsa Ltda* por permitirme hacer este trabajo de memoria con ellos, descubrí una pasión por la visión computacional que antes no tenía, trabajar con usted me ha hecho crecer como profesional, muchas gracias por la oportunidad de trabajar con ustedes.

A mi mejor amigo Gonza... gracias por todo, si estoy acá en gran parte es por ti, sin ti me habría rendido a la mitad, sin ti muchos ramos habrían sido insufribles, gracias por tantas risas, siempre recordaré el ir por papas después de una prueba sabiendo que nos habían rajado jajaja, no podría haber deseado un mejor amigo más genial que tú, gracias por todo bro <3.

Una cita del discurso de Snoop Dogg: *Last but not least... I wanna thank me, I wanna thank me for believing in me, I wanna thank me for doing all this hard work, I wanna thank me for having no days off, I wanna thank me for never quitting, I wanna thank me for always being a giver and trying to give more than I receive, I wanna thank me for trying to do more right than wrong, I wanna thank me for being me at all times... Thank you God, because I'm nothing without you.*

Thank you everyone!!

Tabla de Contenido

Lista de tablas

Lista de figuras

1. Introducción

1.1. Motivación

A lo largo de los años ha existido un aumento en el uso de Machine learning o aprendizaje automático, el cual corresponde a una sub-categoría de la inteligencia artificial, este se basa en crear algoritmos capaces de descubrir patrones en conjuntos de datos, comportamientos, imágenes, palabras, entre otros.

Para el Machine learning lo más importante son los datos, ya que teniendo una mayor base de datos, se tiene más ejemplos para comparar las características de los datos y encontrar patrones, mejorando así el rendimiento del algoritmo. Entre los algoritmos clasificadores de datos se tienen los algoritmos de regresión, árbol de decisiones y cuando se poseen los datos pero no se sabe a qué corresponden se pueden utilizar algoritmos de 'clustering' como K-means.

Un algoritmo de Machine learning son las redes neuronales, las cuales consisten en una imitación matemática sobre la comunicación entre neuronas biológicas, estas redes neuronales están compuestas por capas, cuando se poseen redes con múltiples capas ocultas se denominan redes profundas.

Las redes profundas conforman la rama del Machine learning llamada Deep Learning, la profundidad de estas redes ha permitido realizar avances revolucionarios en la inteligencia Artificial, haciendo que los algoritmos de aprendizaje sean más precisos y pueden trabajar tareas más complejas con una mayor cantidad de datos.

Otro avance importante en la redes neuronales son las redes neuronales convolucionales (CNN), Estas redes facilitan el procesamiento de imágenes mediante el uso repetido de máscaras que pasan por sobre las imágenes para encontrar patrones a lo largo de la imagen y así poder obtener características de esta.

En *Apsa Ltda* buscan automatizar el proceso de creación de inventarios viales, en la actualidad existen 2 métodos, obtener los datos manualmente, esto es recorrer la ruta y a medida que se observa el elemento, se registra en una planilla, indicando el tipo de elemento, ubicación y dimensiones, entre otras características y en caso de ser necesario se pasa la planilla manualmente a un Excel, el otro método corresponde a un proceso semi-automático, en este segundo método, se utiliza una cámara georeferenciada sobre un vehículo, el cual realiza el trayecto por la ruta deseada y luego mediante un programa con un leve proceso de automatización, se seleccionan todos los objetos manualmente.

Debido a que ambos procesos consumen mucho tiempo, se busca automatizar la creación de inventario para poder reducir su costo y tiempo de obtención. Esto se realizará mediante el uso de CNN, Deep Learning y algoritmos de Machine learning. La data será proporcionada por *Apsa Ltda*, a la cual se realizará el preprocesamiento debido para poder ser utilizada por los algoritmos y el código general que será creado.

1.2. Objetivos

Objetivo General

Crear un código que entrega un inventario de luminarias en carretera mediante la implementación de algoritmos de detección, clasificación, seguimiento y localización, el formato de entrega del inventario es Excel.

Objetivos Específicos

1. Realizar un preprocesamiento de datos.
2. Entrenar un modelo de *object detection*
3. Implementar seguimiento de objetos al código general
4. implementar y utilizar la información georeferenciada.
5. Crear un DataFrame con toda la información deseada
6. Exportar el DataFrame junto a elementos de interés

2. Marco teórico

Inteligencia Artificial

2.1. Perceptrón

El perceptrón es la unidad más simple de neurona artificial, el cual se basa en el concepto de una neurona biológica, para el perceptrón, los inputs y outputs son números y dentro de la neurona existe una ponderación de los datos de entrada lo que otorga un valor de salida. Como se puede observar en la figura 1 existen 'n' inputs (x_i), cada input está ponderado por su peso correspondiente w_i, además existe la entrada de una constante también ponderada. La suma de todas las ponderaciones se entrega a una **función de activación**, como por ejemplo la función de escalón(figura 2), esto lo que hace es crear un threshold, para finalmente poder hacer una separación simple de clases a partir de una recta (figura 3).

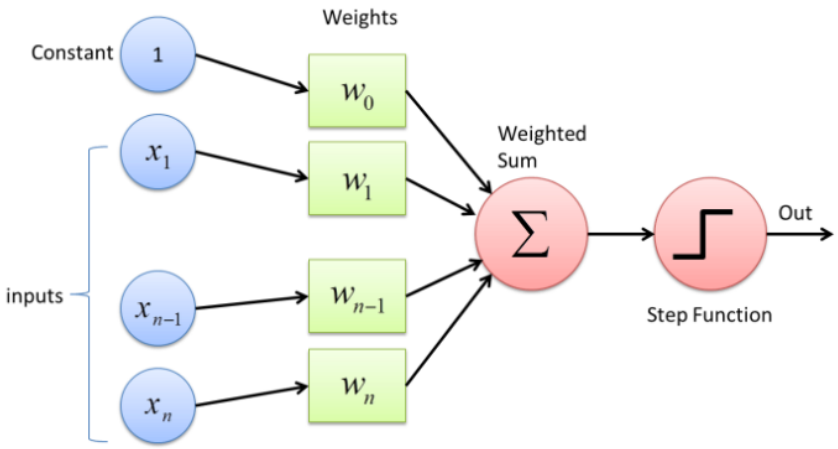


Figura 1: Perceptrón [1]

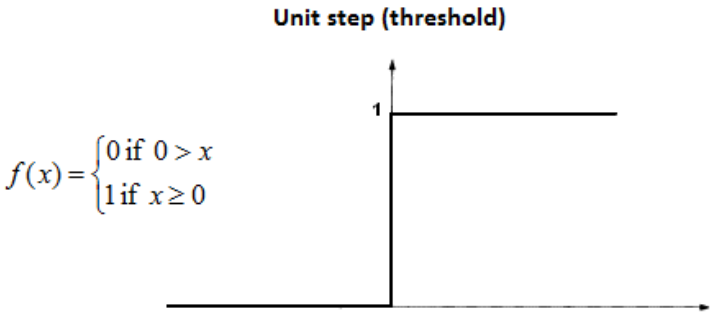


Figura 2: Función de activación escalón [1]

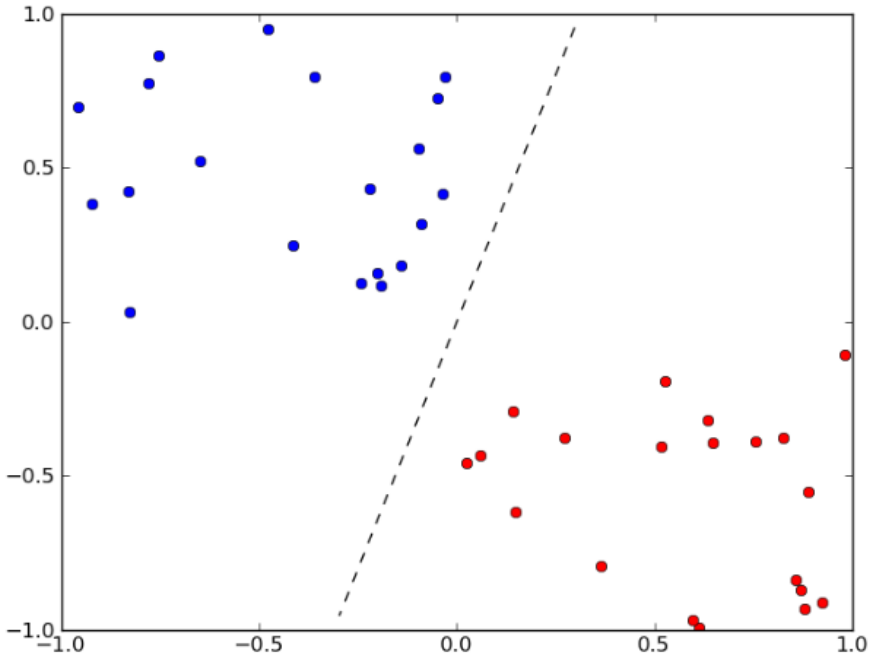


Figura 3: Ejemplo de clasificación mediante perceptrón [1]

2.2. Funciones de activación

También conocida como función de transferencia, estas funciones tuvieron un avance importante en la inteligencia artificial, ya que la creación de mejores funciones de activación permitieron optimizar de mejor manera las redes neuronales artificiales.

Estas funciones son importantes, porque si se juntan múltiples ecuaciones lineales, el resultado es otra ecuación lineal, pero no todos los problemas son resueltos por una linea recta, por ejemplo $f(x) = 4x+3$ y $g(x) = 3x-1$ son 2 outputs de diferentes perceptrones sin sus respectivas funciones de activación, por lo que el resultado al juntarlas en cadena sería $f(g(x)) = 4(3x-1)+3 = 12x-1$, esto significa que si hacemos una cadena de perceptrones, es decir agregamos capas, el resultado es lo mismo que utilizar un único perceptrón.

Debido a esto la utilización de funciones de activación es importante, ya que agregan un factor de no linealidad al modelo, permitiendo resolver problemas no lineales. Es decir una gran cantidad de capas de perceptrones con funciones de activación teóricamente pueden aproximar cualquier función continua.

A continuación se muestran gráficamente alguna de las funciones de activaciones existentes junto a sus derivadas.

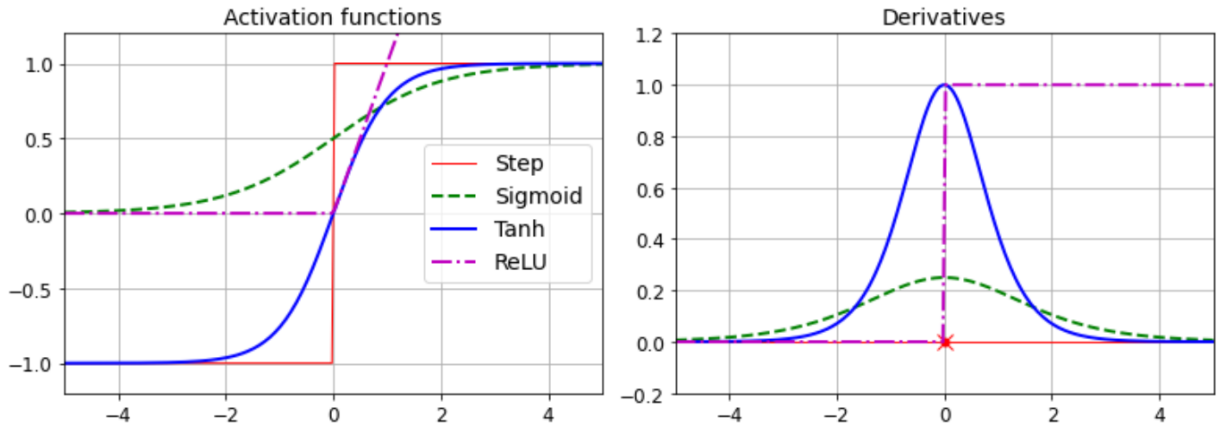


Figura 4: Funciones de activación [2]

Fórmulas de las funciones de activación mostradas:

- Función escalón:

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } 0 > x \end{cases}$$

- Función sigmoide:

$$f(x) = \frac{1}{1 + exp(-\alpha x)}$$

- Tangente hiperbólica:

$$f(x) = \frac{exp(2x) - 1}{exp(2x) + 1}$$

- ReLu (*Rectified Linear Unit*):

$$f(x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{si } 0 > x \end{cases}$$

2.3. Multi Layer Perceptron (MLP)

Los perceptrones multi capa, corresponde a agregar capas creando una red neuronal más profunda, de esta manera se pueden resolver problemas más complejos como el XOR, el cual no lo puede resolver una única recta (figura 5).

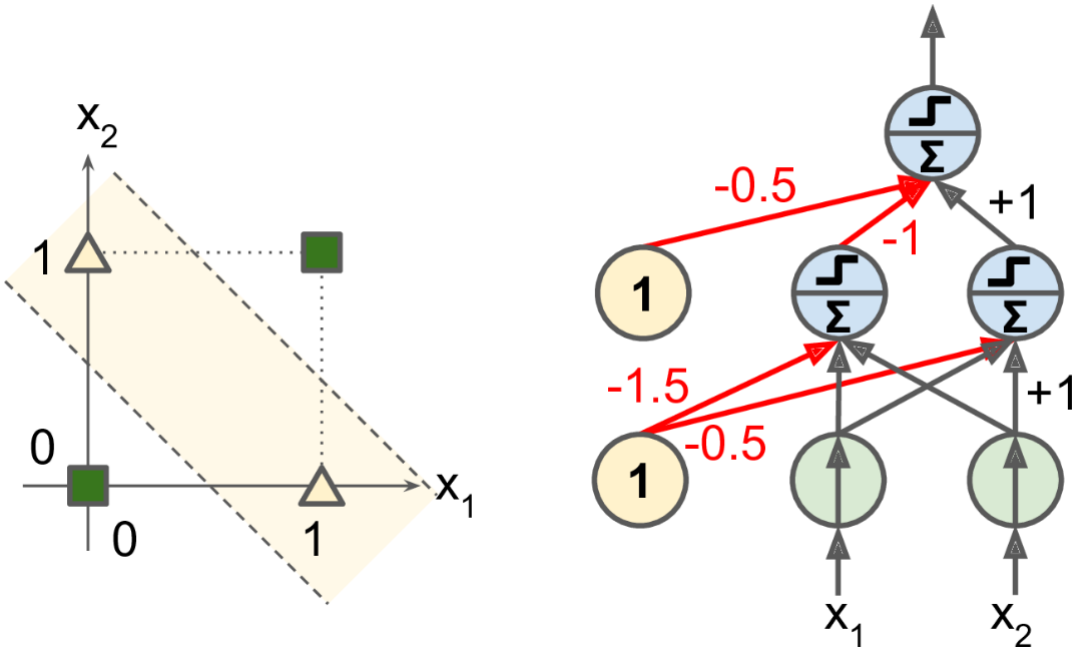


Figura 5: Problema de clasificación XOR y el MLP que lo resuelve [2]

De esta manera se pueden crear múltiples capas, las cuales se pueden clasificar en 3 grupos:

- Capa de entrada (input layer): Corresponde a la capa que recibe los datos.

- Capa escondida (hidden layer): Una capa oculta está compuesta por neuronas que ponderan valores de la capa anterior y cada neurona obtiene un output al pasar la sumatoria de los valores ponderados por la función de activación.

- Capa de salida (output layer): Puede ser una única neurona o múltiples neuronas, la salida de las neuronas de la capa de salida también pasan por una función de activación. Cuando se busca clasificar se suele ocupar Softmax como función de activación, la diferencia es que para softmax se crea una capa a la cual se le pasan las sumatorias de todas las salidas, se puede observar a continuación la función Softmax de la neurona de salida i-ésima, con m salidas existentes:

$$Softmax(y_i) = \frac{exp(y_i)}{\sum_{j=1}^{m} exp(y_j)}$$

Con lo mencionado anteriormente, se muestra un ejemplo de un clasificador MLP con solamente una capa oculta, la cual posee ReLu como función de activación y como es un MLP clasificador, se utiliza Softmax en la capa de salida.

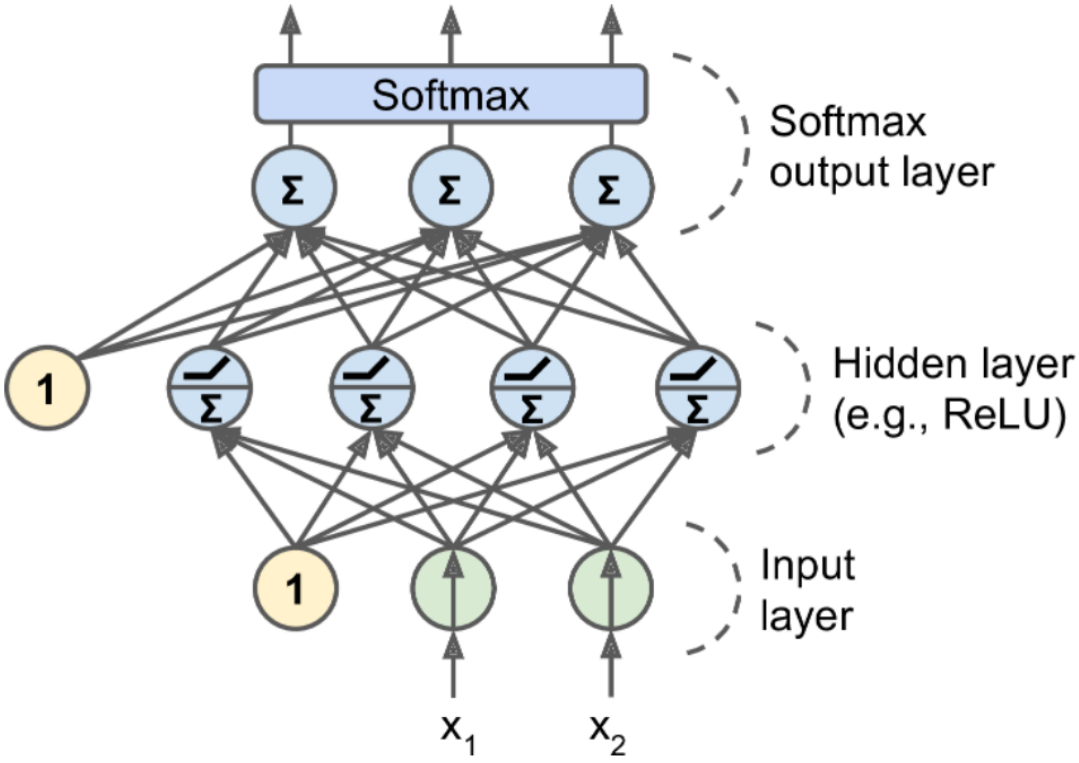


Figura 6: Ejemplo de MLP [2]

2.4. Red Neuronal Convolucional (CNN)

Las CNN (*Convolutional neural networks*) son redes neuronales profundas (*Deep neural network*), estas redes son capaces de ejecutar tareas complejas sobre imágenes, obteniendo características (*features*) de ellas. Antes de la creación de CNN se extraían características de imágenes con métodos matemáticos como HOG[3] o LBP[4], estos extractores de características poseían un potencial limitado, el cual fue sobrepasado por las CNN.

Las CNN generalmente están conformadas por una capa de entrada (imagen en RGB o Gris), capas convolucionales, capas de pooling y capas fully connected. A continuación se muestra un ejemplo de CNN a la que le entra una imagen de un número del 1-9 en escala de grises y al finalizar la pasada por la CNN la red predice el el número al que corresponde la imágen.

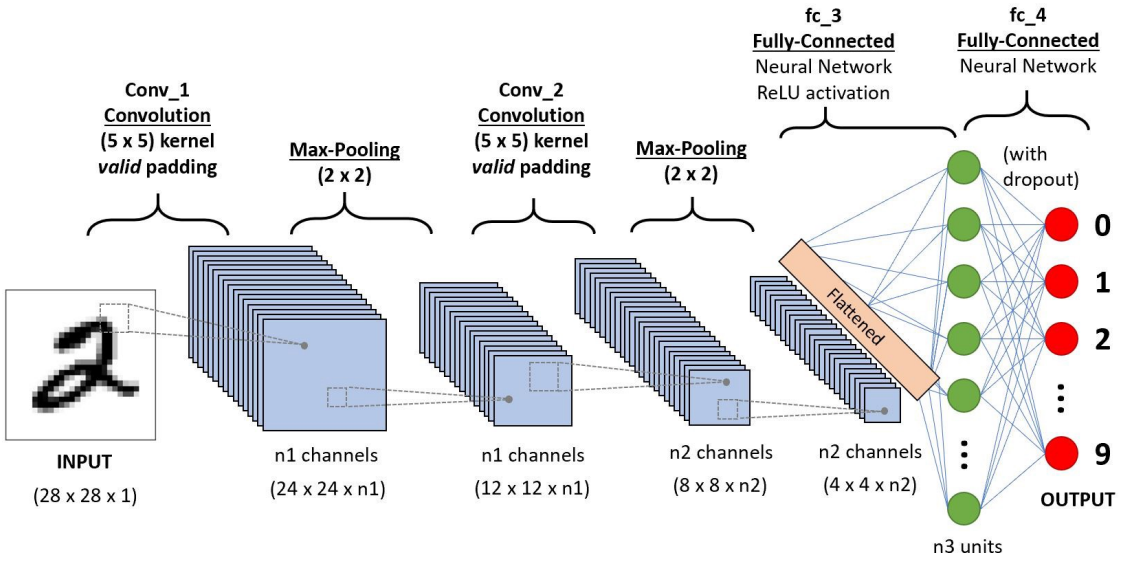


Figura 7: Ejemplo de CNN [5]

2.4.1. Capa convolucional

Corresponde al bloque fundamental de las CNN, el cual corresponde a la operación de convolución que es un barrido de un kernel o máscara K(m, n) donde (m, n) son las dimensiones del kernel, generalmente los Kernel poseen dimensiones impares, para que el Kernel tenga un centro definido, el cual se posiciona sobre el pixel I(i, j) que se desea evaluar. para notación $n' = (n-1)/2$ y $m' = (m-1)/2$.

$$s[i,j] = \sum_{v=-n'}^{n'} \sum_{u=-m'}^{m'} I(i+u, j+v)K(m,n)$$

Como se puede deducir, si $m = n = 3$ para que el Kernel pueda encontrarse completo en la imagen, se debe partir del 2do pixel y terminar en el penúltimo, de esta manera se pierden 2 pixeles de información cuando se realiza la convolución, esto se puede ver reflejado en la siguiente figura.

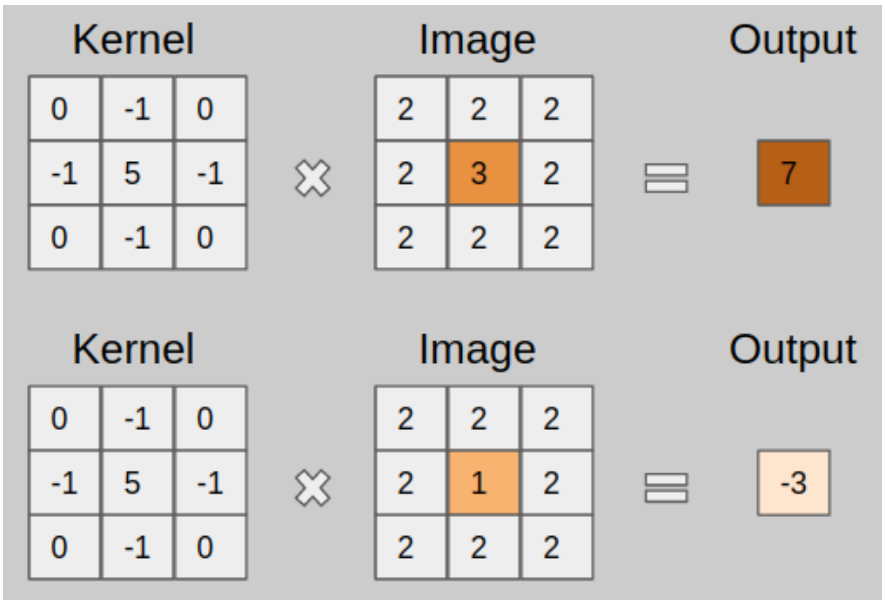


Figura 8: Ejemplo de Kernel(3,3) [6]

Padding

Para lograr conservar las dimensiones de la imagen entrante, se realiza un padding a la imagen, como por ejemplo *Zero padding* de cuanto sea necesario para conservar el tamaño de la imagen, esto consiste en agregar un contorno de valor 0 la imagen. Se puede observar la figura 9 como una imagen de 5x5 conserva sus dimensiones después de pasar por un Kernel(3,3) agregándole un contorno de padding.

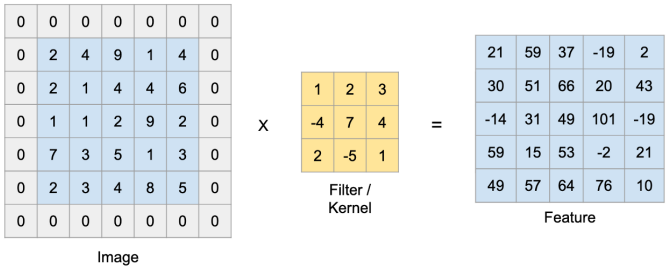


Figura 9: Ejemplo de *zero padding* [7]

Ejemplo de Kernel

La capa convolucional está compuesta por múltiples Kernels o filtros que obtienen diferentes mapas de características (*feature maps*) de las imágenes, se le pasan múltiples Kernel, generando así diversas matrices (imágenes) y el conjunto de estas matrices son tensores (imagen 2D x cantidad de imágenes), un ejemplo de Kernel es el siguiente filtro de bordes.

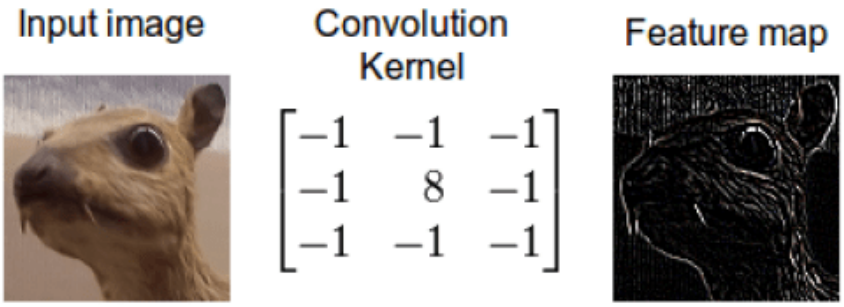


Figura 10: Filtro de bordes [8]

2.4.2. Capa de *Pooling*

En busca de optimizar el costo computacional afectando lo menos posible a la información que posee la capa de CNN correspondiente al potencial de los resultados de las CNN se busca reducir, debido a esto se ocupan las capas de pooling, las cuales poseen generalmente un Kernel(2,2), los tipos más utilizados de pooling son:

- **Max Pooling**: El output de pixel es el máximo valor que se encuentra en el área del Kernel.

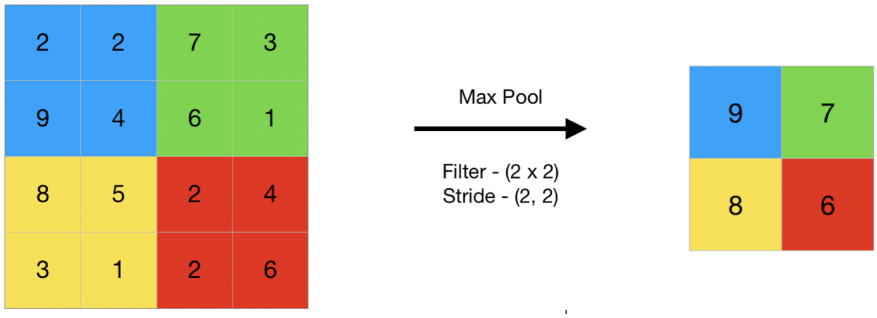


Figura 11: Max pooling [9]

- **Average Pooling**: El output corresponde al promedio de los valores que se encuentren en el Kernel

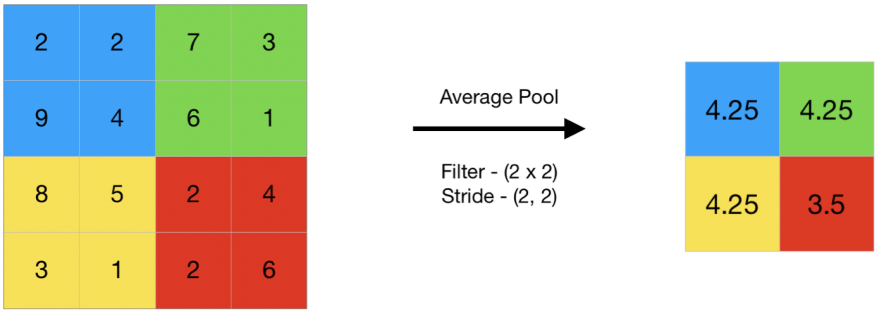


Figura 12: Average pooling [9]

Stride

Corresponde a cuantas unidades se desplaza el Kernel para realizar otra convolución, esto es útil, ya que si el *stride* del Kernel(2,2) de la capa de pooling fuera 1, entonces la capa resultante solo sería de 1 unidad más pequeño lo que no ayuda a reducir el costo computacional, debido a esto el *stride* es de 2, así la dimensión de la capa anterior se ve reducida a la mitad.

De esta manera se procede a realizar un downsampling de los resultados de las capas de convolución reduciendo así el costo computacional sin perder exceso de información, esto se observa en la siguiente figura:

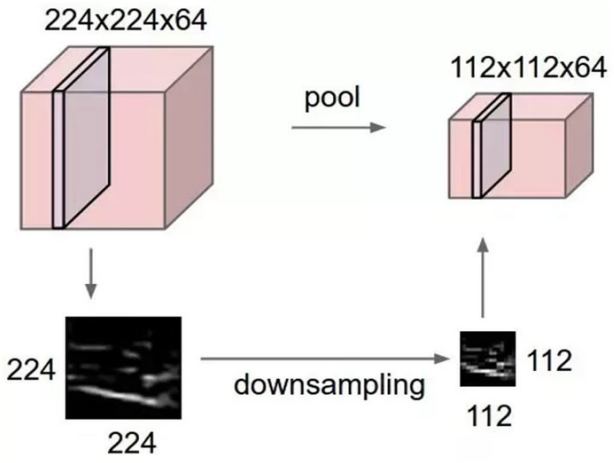


Figura 13: Ejemplo de downsampling de una capa convolucional [10]

2.4.3. Capa de *Fully connected*

Una vez obtenida las capas de features, esta información debe ser procesada, para esto se usan capas *fully connected*, es decir este conjunto de capas corresponde a un MLP, pero como se vio en la sección de MLP, los datos ingresados son vectores, pero actualmente se posee un tensor (por ej se busca pasar de 4x3x3 a 36x1x1). Debido a esto es necesario hacer un *flattening* del mapa de características obtenido de las capas de convolución y pooling. En la figura 14 se observa un ejemplo genérico del proceso de *flattening* seguido por las capas fully connected (MLP). Como se mencionó en la sección de MLP, la salida del MLP finalmente pasa por la función de activación *softmax* para así encontrar la clase a la corresponde la imagen que inicialmente se le ingresó a la CNN.

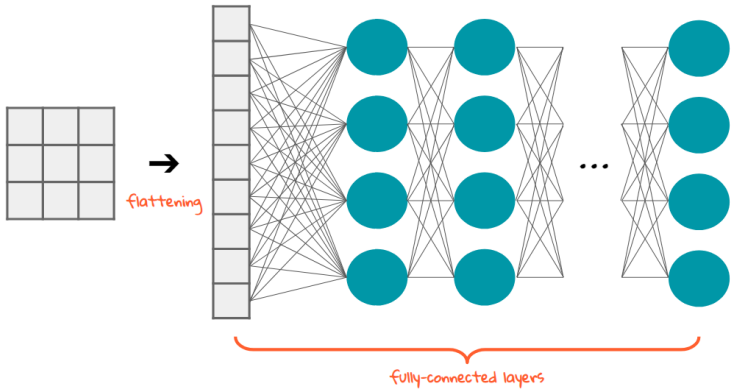
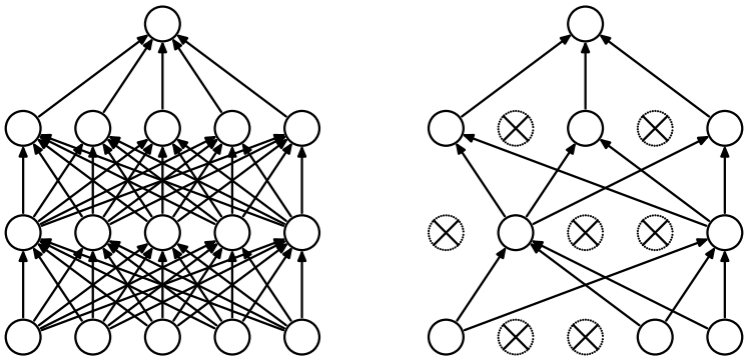


Figura 14: Ejemplo de *flattening* [8]

Dropout

Para asegurar el correcto funcionamiento de los modelos, se aplica *dropout* [11] en las capas *fully connected*, este dropout es una probabilidad de que cada neurona de la capa a la que se le aplicó se apague, es decir, que la neurona no cuente en la sumatoria correspondiente, esto se puede ver ilustrado en la siguiente figura.



(a) Standard Neural Net (b) After applying dropout.

Figura 15: Ejemplo de *dropout* [11]

Como se puede observar, después de aplicar *dropout*, hay neuronas que no contribuyen a los valores de entrada de la siguiente capa, esto provoca que se creen MLP robustas, es decir, en caso de que una parte deje de funcionar, se sigue consiguiendo resultados óptimos.

2.5. Detección de objetos

Cuando se busca clasificar y localizar múltiples objetos dentro de una imagen con diferentes elementos, se llama *object detection*. Cuando se comenzó a realizar estas tareas se utilizaba una CNN junto a una ventana deslizante, esta se pasa a lo largo de la imagen generando clasificaciones a lo largo de esta.

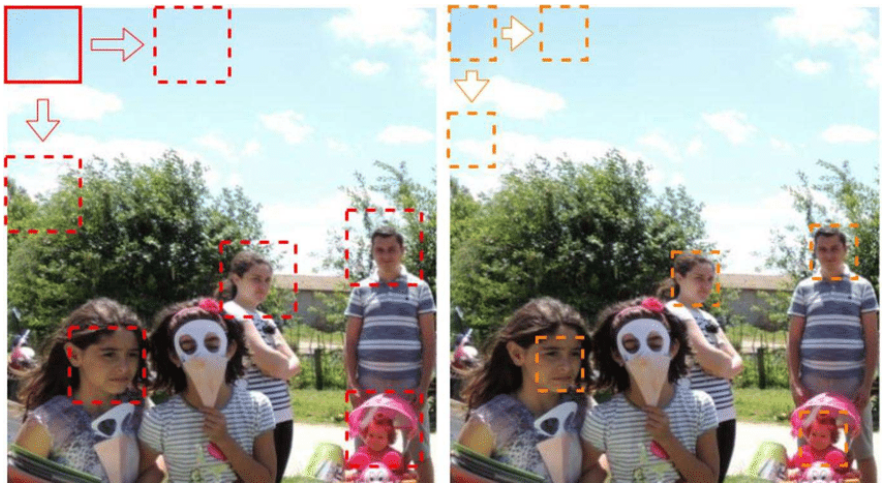


Figura 16: Ejemplo de *sliding window* [12]

Non-Maximum supression

Como se puede observar en la figura 16 se crean ventanas de diferentes dimensiones, esto permite detectar objetos en múltiples escalas. Este método tiene un problema, se pueden generar múltiples detecciones para un mismo objeto, esto se soluciona con la técnica de filtrado *Non-maximum suppression* (NMS), para esto, a cada detección se le debe agregar una probabilidad de que pertenezca a la clase, de esta manera se crean *bounding boxes* asociados a una clase junto con la probabilidad de pertenecer a esa clase.

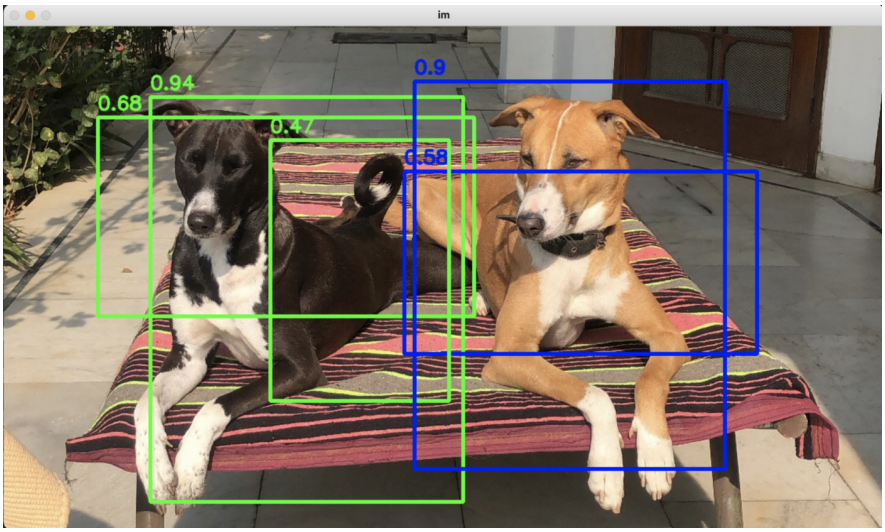


Figura 17: Detecciones pre NMS [13]

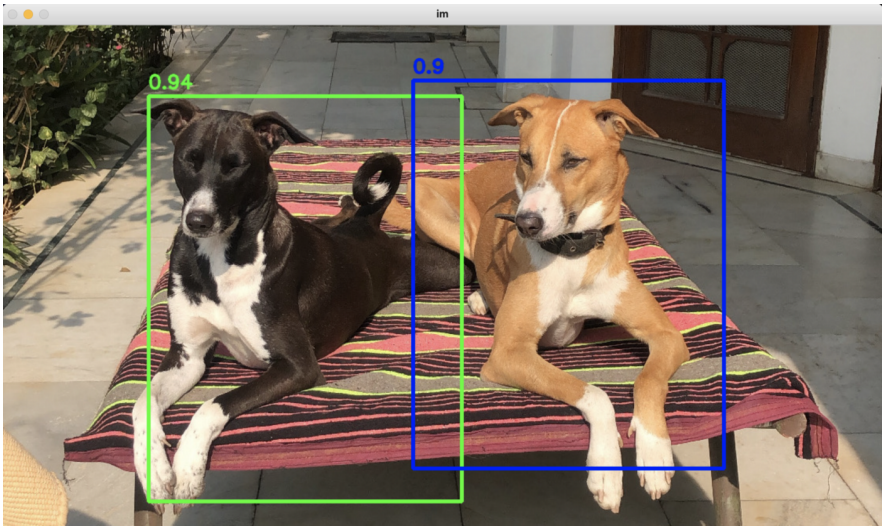


Figura 18: Resultado post NMS [13]

Existen *thresholds* para la creación de *bounding boxes*, donde valores de detección menores al *threshold* son inmediatamente descartados, los *bounding boxes* que superen el *threshold* se analizan de la siguiente manera:

1. Se Elige el *bounding boxe* con la mayor probabilidad
2. Se eliminan todos los *bounding boxes* que tengan algún criterio de superposición con la detección dominante, como por ejemplo un IoU superior a 60 % (IoU se explicará en la sección de métricas de desempeño).
3. Repetir hasta que no queden *bounding boxes* para eliminar.

2.6. Métricas de desempeño

Para evaluar el desempeño de los modelos, existen diversas métricas. Para poder comprenderlas con mayor facilidad se deben entender los siguientes conceptos.

2.6.1. Verdaderos v/s falsos y positivos v/s negativos

Se obtienen cuatro grupos a partir de estos conceptos:

- **True Positive** (TP): Objetos predichos y clasificados correctamente.
- **True Negative** (TN): Correctamente se predijo que cierto objeto no correspondía a la clase.
- **False Positive** (FP): Se predice que existe un objeto, cuando en realidad no existe.
- **False Negative** (FN): El modelo no detecta un objeto, pero sí debería ser detectado.

A partir de estos concepto surgen sus derivados como *Ground truths* que corresponde a todo lo que debería ser detectado y clasificado (TP+FN) y el total de detecciones realizadas (TP+FP).

2.6.2. Precisión

De todas la detecciones realizadas, cuantas fueron correctas.

$$Precisin = \frac{TP}{TP + FP} = \frac{Detecciones\ correctas}{Total\ de\ detecciones}$$

2.6.3. Recall

De todas las detecciones posible, cuantas fueron correctamente realizadas.

$$Recall = \frac{TP}{TP + FN} = \frac{Detecciones\ correctas}{Ground\ truths}$$

2.6.4. Accuracy

Corresponde al que porcentaje de todos los resultados corresponden los correctos, cuando se trabja con bounding boxes no suele utilizarce debido a que los TN son demasiados (todo lo que no es bounding box), por lo que accuracy puede ser un métrica de evaluación muy engañosa.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} = \frac{Resultados\ correctos}{Todos\ los\ resultados}$$

2.6.5. Intersection over Union (IoU)

Es la razón entre la intersección del *bounding boxes* predicho y su *ground truth* sobre la unión de estos. Esto se puede visualizar como:

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

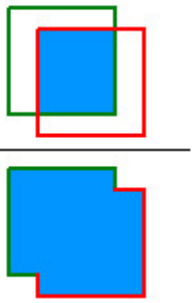


Figura 19: Intersection over union [14]

2.6.6. Average precision y mean average precision

Si *Precision* y *recall* se utilizan aisladamente pueden ser contraproducentes, ya que es posible tener un *precision* muy alto a costa de bajar mucho el *recall* y viceversa. Es por eso que existen otras métricas de desempeño complementarias para detecciones mediante *boudning boxes*.

Average Precision (AP) soluciona este problema. Corresponde a el comportamiento promedio de *precision* a medida que aumenta *recall* a medida que se cambian los *threshold* de detección. Si usamos un *threshold* más alto, objetos que antes se detectaban ahora no lo harán y viceversa, con esto también cambian los valores de *precision* y *recall*.

Se deben introducir TP, FP y FN para IoU:

- **True Positive**: IoU_ts el threshold que define si el *bounding box* corresponde o no a un TP, es decir, si el IoU es mayor al IoU_t y la clase detectada es la misma, entonces la detección se considera como TP.

- **False Positive**: De manera opuesta al caso True Positive, si el IoU es menor al IoU_t y la clase es la misma, entonces corresponde a un FP, porque supera el *threshold* para que exista el *bounding box*, pero su IoU es muy pequeño. También si para un mismo objeto hay diferentes detecciones correctas, solo la con la mayor confianza de detección cuenta para TP, el resto son FP.

- **False Negative**: Cuando no hay ninguna detección o su IoU es mayor al IoU_t, pero la clasificación es errónea.

- **True Negative**: No se usa, ya que en imágenes grandes hay mucha área no clasificada por lo que TN llevaría una parte muy grande de los datos y esto es contraproducente.

Se puede observar que solo se considera *Positive* cuando las clases detectadas coinciden. Para poder calcular *Average Precision* se necesita la curva *precision v/s recall*, para obtenerla se realiza el siguiente procedimiento para cada clase C_m que se desea detectar.

1. Se ordena todas las detecciones de mayor a menor según la confianza de la detección (probabilidad de la clase asociada al *bounding box*).
2. Saber si existe algún **FN**.
3. Se determina si las clasificaciones corresponden a un **TP** o **FP**.
4. Se calcula el **TP** + **FN** para todos los cálculos de recall.
5. Se comienza del primer elemento, se calcula su *precision* y *recall*, se debe destacar que para el cálculo de *precision*, los **TP** y **FP** se acumulan a lo largo de los elementos, es decir ambos valores parten de 0 y se les suma 1 cuando corresponda según si el elemento es **TP** o **FP**. Para *recall*, el denominador es fijo y se calcula en el paso 4, pero su numerador se acumula igual que para *precision*.

Con esto se genera una tabla, a continuación se ilustra un ejemplo donde hay 5 elementos a detectar, con los siguientes tipos de detecciones, 5 detecciones **TP**, 5 detecciones **FP**, **FN** $= 0$ y un *bounding box threshold* $= IoU_t = 0{,}5$

Rank	Confianza	IoU	TP o FP	Precision	Recall
1	0.95	0.7	TP	1.0	0.2
2	0.9	0.8	TP	1.0	0.4
3	0.78	0.45	FP	0.67	0.4
4	0.7	0.3	FP	0.5	0.4
5	0.65	0.4	FP	0.4	0.4
6	0.62	0.9	TP	0.5	0.6
7	0.59	0.6	TP	0.57	0.8
8	0.55	0.42	FP	0.5	0.8
9	0.53	0.27	FP	0.44	0.8
10	0.51	0.55	TP	0.5	1.0

Tabla 1: Ejemplo tabla AP

Para el elemento en rank 5, se tiene que hasta el momento **TP** $= 2$, **FP** $= 3$, **FN** $= 0$, por lo que su *precision* y *recall* son los siguientes:

$$Precision = \frac{TP}{TP + FP} = \frac{2}{2+3} = 0{,}4$$

$$Recall = \frac{TP}{5} = \frac{2}{5} = 0{,}4$$

Esto se puede representar con la siguiente curva:

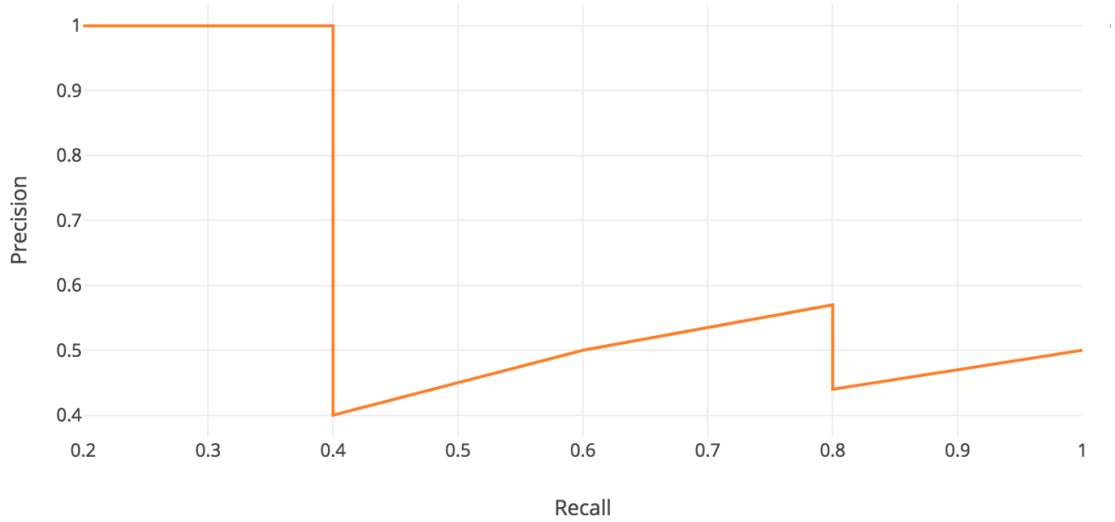


Figura 20: Curva *precision* v/s *recall* [15]

Para obtener el área bajo la curva se utiliza la siguiente fórmula, dónde p(r) corresponde a la curva de precisión dado un recall y como se encuentran en el intervalo [0,1], se tiene la siguiente fórmula.

$$AP = \int_0^1 p(r)dr$$

Como *precision* y *recall* se encuentran en el intervalo [0,1], el valor de AP también se encuentra en el intervalo [0,1]. Hay un procedimiento generalizado para hacer más simple este cálculo, ya que en la configuración de una red neuronal, se debe realizar múltiples veces, por lo que simplificar su costo computacional es importante, debido a esto se suaviza el patrón zigzag y se toma el valor más alto de precisión hacia la derecha posible, tal y como se muestra en la siguiente figura.

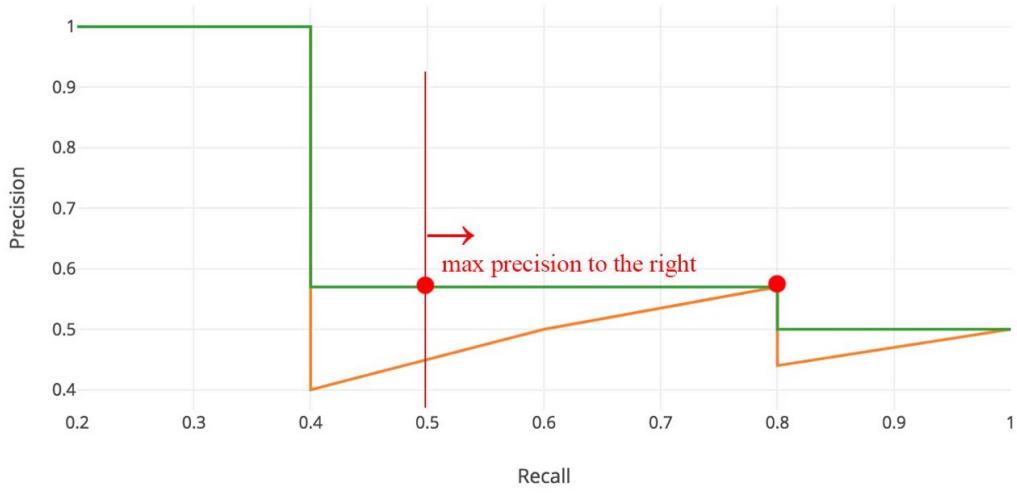


Figura 21: Suavizado Curva AP [15]

La nueva curva AP pasa a ser la suavizada (color verde) y para calcular el área bajo la curva se realiza una sumatoria, dado n particiones, para este caso sirve $n = 10$, esto crea n+1 puntos de contacto en las rectas [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0] por lo que se tiene la siguiente fórmula.

$$AP = \sum_{n=0}^{10} (r_{n+1} - r_n) p_{interp}(r_{n+1})$$

Dónde r_n corresponde al valor del *recall* del punto n y $p_{interp}(r_n)$ corresponde al valor de *precision* del punto n en la curva suavizada (*interpolated precision*). Esta repartición de puntos e intervalos se puede ver representado en la siguiente figura.

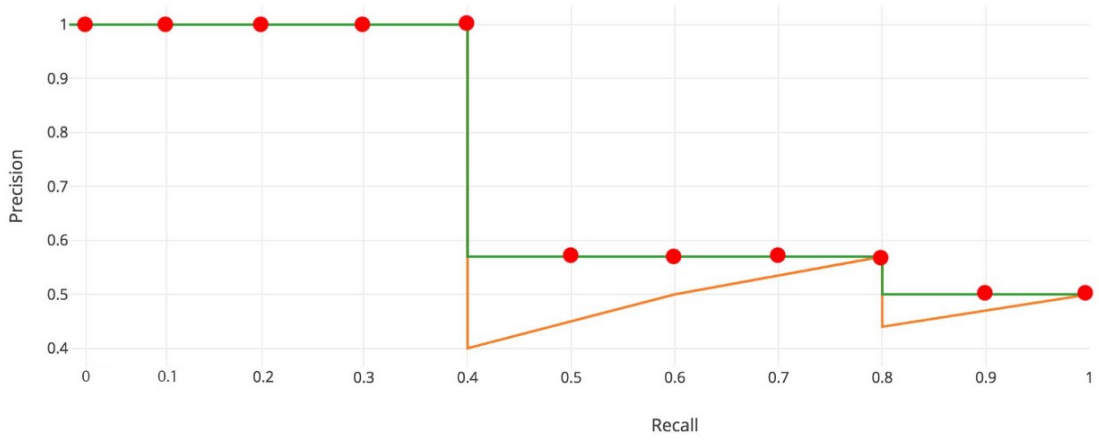


Figura 22: Intervalo curva AP [15]

Para otros casos, se utilizan otros intervalos, acá también hubiera funcionado el intervalo [0, 0.4, 0.8, 1.0], pero para esto se necesita conocer los cambios en la curva, debido a esto se realizan particiones iguales con n intervalos.

Mean average precision (mAP)

El AP calculado anteriormente corresponde por ejemplo a la clase de *perro*, pero hay otras clases que puede identificar la red neuronal, como *gatos*, *caballos*, *personas*, etc. Debido a esto se busca encontrar el promedio de los AP, Siendo M la cantidad de clases existentes y AP_c es el valor del AP de la clase c, se tiene que mAP se obtiene con la siguiente fórmula.

$$mAP = \frac{1}{M} \sum_{c \in Clases} AP_c$$

2.7. Object detection (YOLO)

Existen 2 modelos de *object detection*, uno de dos fases, donde se extraen *region proposal* con posibles elementos de la imagen y la siguiente es la clasificación de elementos en las regiones propuestas. La detección de 1 fase corresponde a predecir la zona de la clase directamente, sin entregar primero una region de propuesta. La principal ventaja es que realiza la detección y clasificación en un único proceso, es decir evita el apartado de *region proposal*, por lo que no necesita la red neuronal que realiza esa tarea, acortando así el costo computacional.

Se utilizará la versión YOLOv7 [16], que cuenta con los mejores resultados en la actualidad de algoritmos de detección con las siglas de YOLO, YOLO cuenta con múltiples versiones siendo YOLO[17], YOLOv2 [18] y YOLOv3 [19] las que crearon la base para el resto de algoritmos que usarían las siglas de YOLO.

YOLO (You only look once)

El paper de YOLO fue publicado en 2016 por Redmon et. al [17], permitiendo detecciones de objetos de manera rápida, por lo que se podría utilizar en tiempo real.

Detección

Se muestra el proceso de detección, junto con una representación visual de esta.

1. Divide la imagen de entrada en grilla de tamaño S x S y si el centro de un objeto se encuentra en una celda, esa celda es responsable en determinar la clase a la que corresponde.
2. Cada celda predice B *bounding boxes* con su probabilidad y según el IoU_t se define si es un TP o FP. Si no hay objetos asociados a esa celda, la probabilidad es 0.
3. Se construyen los elementos asociado a cada *bounding box* [x,y,w,h] y su probabilidad asociada.
4. Se calcula la probabilidad de que cada celda corresponda a la clase C_i, esto entrega un mapa de probabilidad por clase con S x S elementos.
5. Se genera la salida, la cual es de la estructura S x S x (B $*5 + C$), es decir entrega un tensor con dimensiones (S, S, $B * 5 + C$).

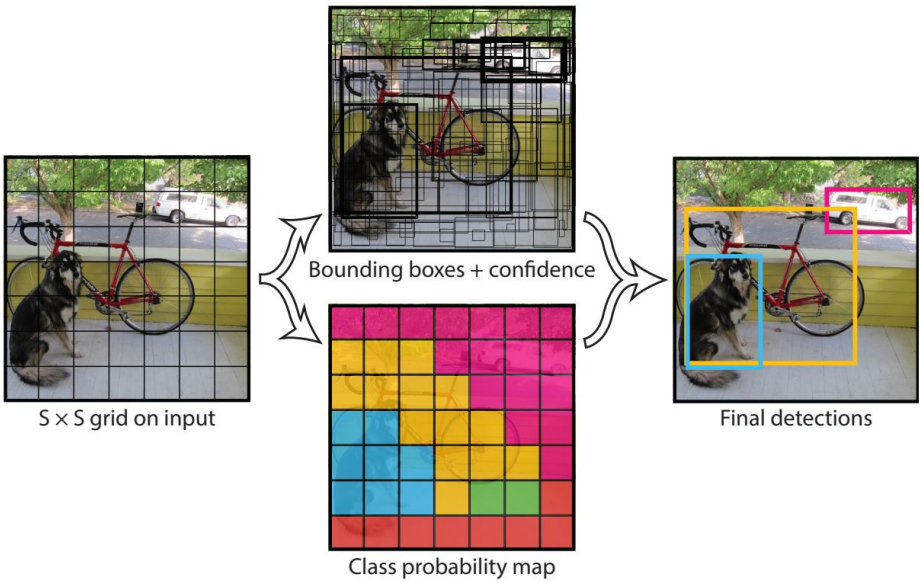


Figura 23: Detecciones de YOLO [17]

YOLOv2 (Yolo9000: better, faster, stronger)

En 2017 Redmon et. al [18] introducen YOLOv2, que es una mejor al algoritmo YOLO que habían desarrollado anteriormente.

Entre las mejoras se encuentran las siguientes:

- **Mejora en resolución**: Se pasó a entrenar de resolución 224x224 a 448 x 448, esto produce una mejora de 4 % en mAP.

- **Anchor boxes**: YOLO predecía desde las capas *fully connected* los *bounding boxes*, en YOLOv2 se eliminan las capas *fully connected* y se utilizan alteraciones de los *anchor boxes*, es decir se modifica la forma base de los *bounding boxes* de cada clase para predecir los objetos de la misma clase, esto elimina el problema de YOLO de predecir una clase por celda.

- **Dimension clusters**: Se deben elegir correctamente los *anchor boxes* de cada clase, esto se hace regulando el IoU de los datos de entrenamiento.

- **Entrenamiento multi-escala**: Debido a que ya no existen las capas *fully connected*, si la imagen de entrada es mayor al mínimo requerido para que funcionen las capas convolucionales y de pooling, no hay problemas, esto permite entradas de tamaño variable, esto permite una mejor generalización del modelo.

En cuanto a velocidad de cómputo, se tiene lo siguiente:

- Se introduce una arquitectura de red llamada **Darknet-19**, esta posee 19 capas convolucionales y 5 de *max pooling*, los Kernels en la mayoria son de 3x3, esta arquitectura es más rápida que la utilizada en YOLO.

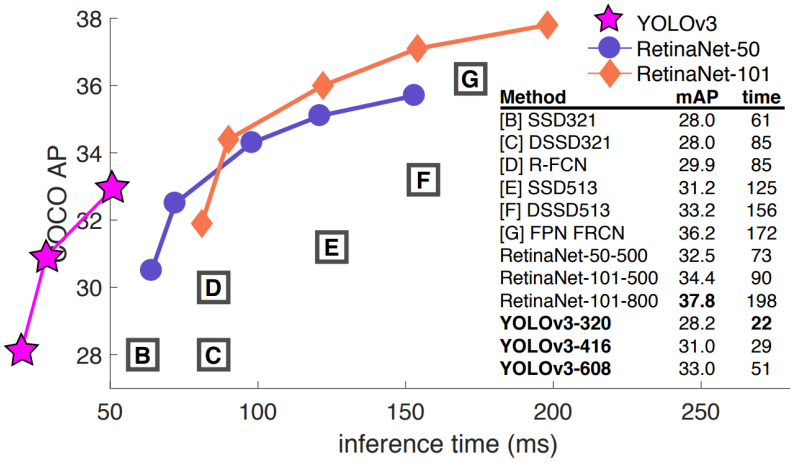
YOLOv3 (Yolov3: An incremental improvement)

En 2018 Redmon et. al [19] introducen YOLOv3, el cual presenta cambios en la arquitectura, entre los cambios se encuentran los siguientes:

- **Predicción multi escala**: Predice *bounding boxes* en 3 diferentes escalas, la última capa convolucional posee tensor de salida, el cual se le entrega a la capa antepenúltima, pero escaldo en factor de 2, es decir se obtiene un *feature map* con mayor resolución este procedimiento se repite una segunda vez, para así obtener 3 salidas de tensores con diferentes escalas.

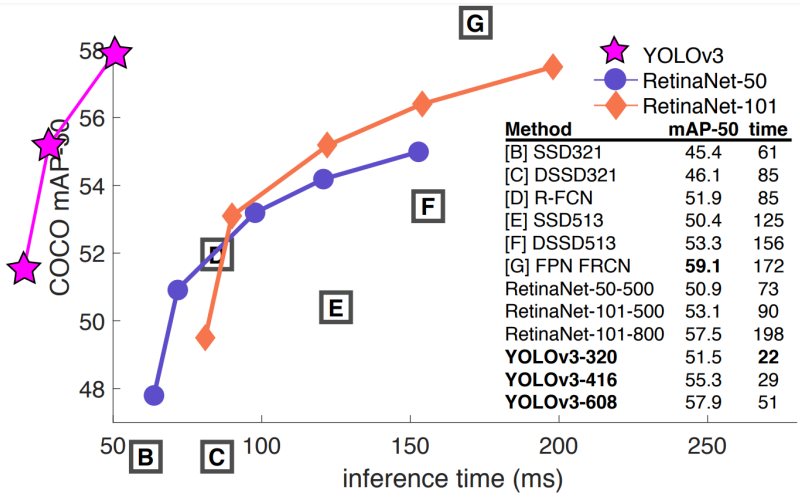
- **Mejora en estructura**: Se utiliza Darknet-53, esta red es más profunda, pero posee *shortcuts*.

YOLOv3 se enfocó en producir resultados competitivos en un tiempo bastante reducido, en las siguientes 2 figuras se puede observar la comparación del modelo YOLOv3 con otros modelos de estado del arte contemporáneos en el dataset COCO.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figura 24: YOLv3 AP vs tiempo de inferencia [19]



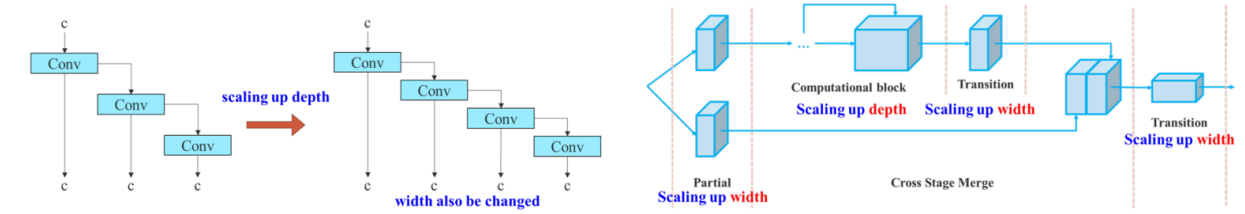
Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figura 25: YOLOv3 mAP-50 vs tiempo de inferencia [19]

YOLOv7 (Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors)

En 2022 A. Bochkovskiy et. al [16] introducen YOLOv7, el cual infiere más rápido y con mejores resultados que versiones previas, mejorando el estado del arte en *object detection*. Para lograr estos resultados se hicieron cambios a la arquitectura y a la forma de entrenar el modelo, alguna de las mejoras realizadas son las siguientes:

- **Model Scaling Techniques**: modificaron el escalado del modelo para sus diferentes versiones, esto permite mantener las mejoras de manera óptima lo largo sus diferentes modelos, desde el más pequeño y simple (-*precisión*, + velocidad), hasta el más robusto (+*precisión*, - velocidad).

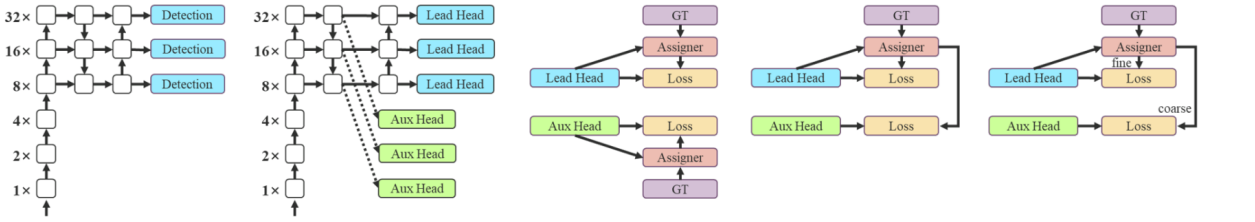


(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figura 26: Yolov7 Model Scaling

- **Auxiliary Head**: Al final del modelo de YOLOv7 se realizan las predicciones del modelo, pero como se encuentra tan lejos del resto de las operaciones del modelo existe pérdida de información, es por esto que se agregan *auxiliary heads* a partir de elementos que se encuentren en medio de la red. Estas *auxiliary heads* no se pueden entrenar tan eficientemente, ya que no se encuentran tan cerca del final de la red, que es donde se realizan las predicciones y se comienzan a realizar los ajustes.

  Es por esto que los autores de YOLOv7 experimentaron con diferentes métodos de supervisión de las *auxiliary heads*, finalmente proponen un método *Lead guided assigner*, el cual propone entrenar la obtención de clases de la textitauxiliary head y textitLead head al mismo tiempo, a partir de la predicción de la textitLead head y el textitground truth del elemento analizado.



(a) Normal model (b) Model with auxiliary head (c) Independent assigner (d) Lead guided assigner (e) Coarse-to-fine lead guided assigner

Figura 27: Yolov7 Auxiliary Head

Como bien se mencionó anteriormente, con las mejoras realizadas YOLOv7 supera a los modelos de *object detection* en cuanto a rendimiento y velocidad de inferencia, esto se puede observar en la siguiente figura.

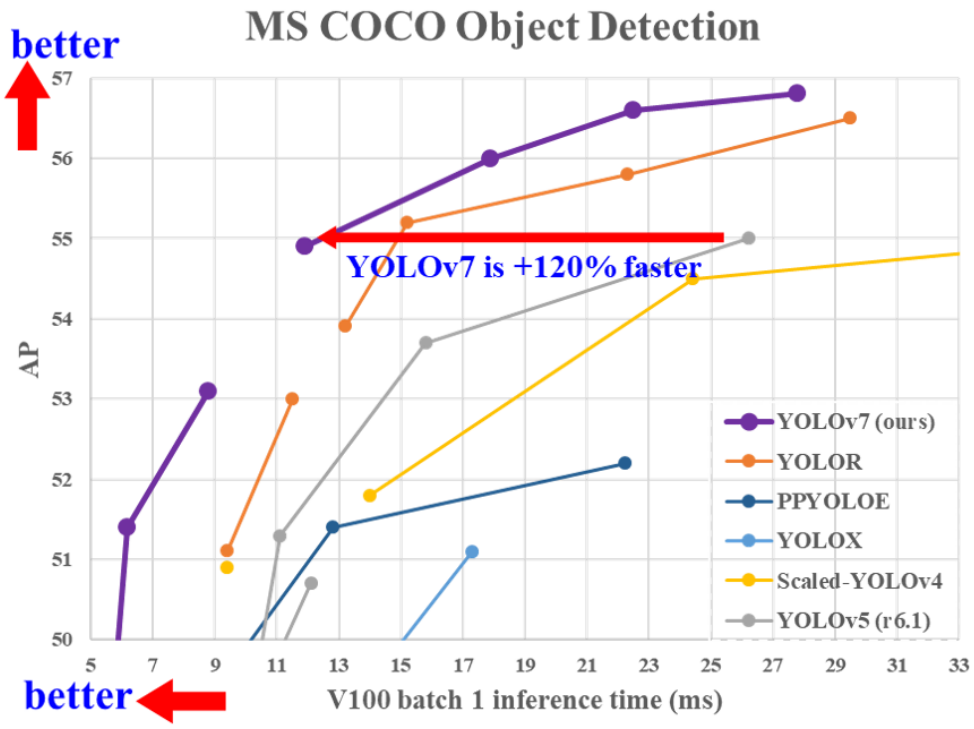


Figura 28: Yolov7 AP v/s tiempo de inferencia [16]

2.8. Object tracking

Específicamente nos referimos al área de *video tracking* el cual consiste en poder identificar objetos que a lo largo de un video cambian de posición y otorgarles un identificador. Esto se realiza analizando los cambios de frame en frame, utilizando la última posición del objeto y haciendo una predicción de su futura posición, creando así un área en la que se podría encontrar el objeto en el siguiente frame.

Existe diversas dificultades al implementar *object tracking* tales como:

- Movimientos aleatorios del objeto, tales como giros abruptos de dirección.
- Oclusión del objeto y su reaparición en lugares no esperados por el algoritmo predictivo, por lo que se pierde el rastro del objeto y se puede identificar como un elemento nuevo.
- Deformaciones a lo largo del video, estas variaciones pueden ser en tamaño y formas, esto dificulta la asociación de características con su estado previo por lo que es posible que se le entregue un identificador incorrecto.
- Alteraciones masivas del fondo del objeto, esto afecta porque los algoritmos predictivos muchas veces sacan información del fondo del objeto y luego buscan esta información en la nueva posición, esto es contraproducente para la asociación de cada elemento con su identificador correspondiente.

2.8.1. Kalman filter

Kalman filter corresponde a un filtro de Bayes, estos filtros entregan la probabilidad de obtener cierto valor o acción dependiendo de un estado entregado. En el caso particular del *Kalman filter*, este es un algoritmo recursivo que utiliza mediciones a lo largo del tiempo para estimar el futuro de una variable, esta predicción se realiza mediante un proceso probabilístico, el cual asume una distribución Gaussiana para las mediciones y modelar las variables de manera lineal, el modelo busca generar una distribución Gaussiana con la menor varianza posible, para así tener una mayor seguridad en la predicción realizada.

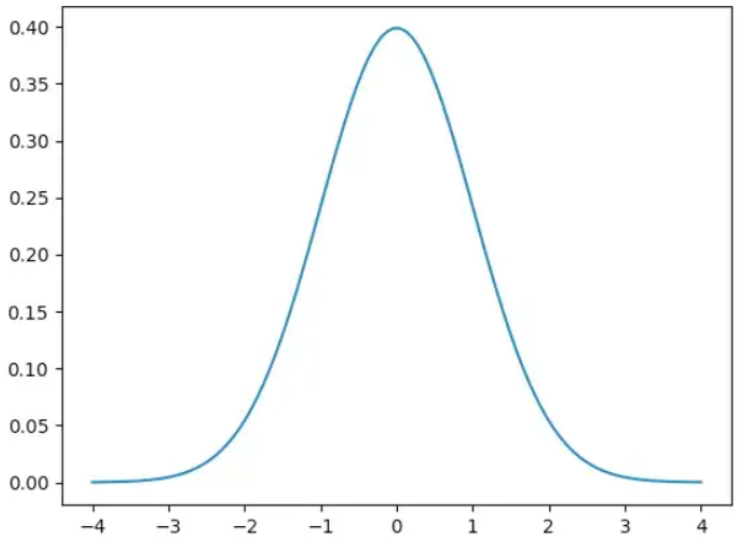


Figura 29: Distribución Gaussiana [20]

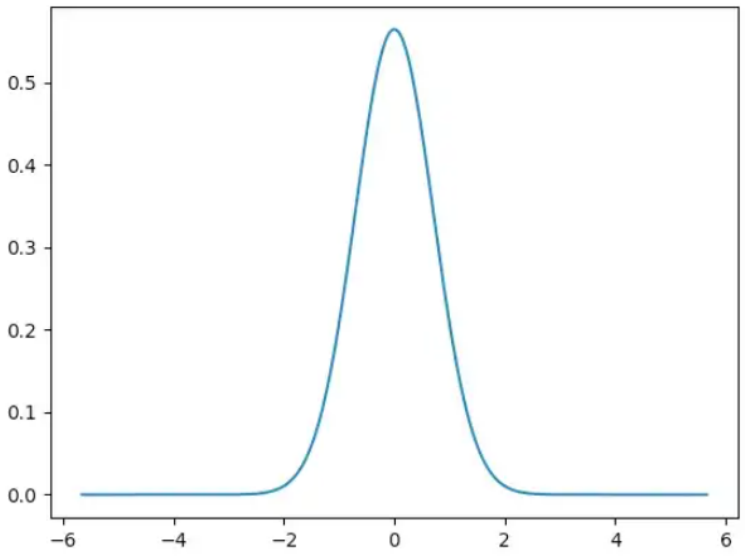


Figura 30: Distribución Gaussiana con una menor varianza [20]

Este filtro posee dos partes, primero se realiza una predicción y luego se corrige, ambas partes se utilizan para la predicción final entregada. Existe una constante llamada la ganancia de Kalman, la cual regula estas dos partes mediante la minimización de las varianzas de las predicciones.

Parte 1: Predicción

De manera generalizada, tenemos n dimensiones, las cuales poseen una distribución Gaussiana, el vector $\vec{X}_{k-1}$ representa el estado en tiempo $k-1$ del objeto que se desea trackear, por lo que se busca estimar el vector $\vec{X}_k$, debido a que es una distribución Gaussiana, esta tiene un vector de promedio asociado $\vec{\mu}$ y una matriz de co-varianza asociada P_{k-1}. Existe una matriz que representa la cinemática del sistema (Matriz A), esta matriz no cambia a lo largo del tiempo, así también existen variables de control la cuales se ven representadas en la matriz B y finalmente como toda medición posee ruido, estas se ven representadas en la matriz Q y el vector $\vec{w}$. A continuación se muestras las ecuaciones para obtener las predicciones del vector $\vec{X}_k^p$ y si matriz de co-varianza asociada P_k^p, a continuación se muestran las ecuaciones de cada predicción.

$$\vec{X}_k^p = A\vec{X}_{k-1} + B\vec{\mu}_k + \vec{w}$$
$$P_k^p = AP_{k-1}A^T + Q_k$$

Parte 2: Corrección

La ganancia de Kalman se utiliza para determinar de la nueva predicción se va a utilizar para predecir y actualizar la nueva estimación, esta se calcula utilizando los errores de la nueva estimación (E_{est}) y el error de la medición ($E_{mea} = R$), de esta manera si el error de la estimación es menor, este tiene mayor influencia y viceversa, debido a la estructura de su ecuación, el valor se encuentra entre 0 y 1, a continuación se muestran dos ecuaciones de la ganancia de Kalman, una más específica y la versión simplificada (La matriz H ayuda a transformar las dimensiones de la matriz P para que quede en el formato de la matriz K):

$$K = \frac{P_k^p H^T}{H P_k^p H^T + R}$$

$$K = \frac{E_{est}}{E_{est} + E_{mea}}$$

Para actualizar la información usando al ganancia de Kalman, es necesario entender las mediciones de entrada Y_k^m que puede contener información como (x, y, velocidad de en x, velocidad en y), también es necesario una matriz C la cual decide que valores de Y_k^m se van a utilizar (puede que existan más mediciones de las necesarias) y el ruido de las mediciones, representado como Z_k, de esta manera se tiene la ecuación de observación:

$$Y_k = CY_k^m + Z_k$$

Para obtener el estado final del objeto en su tiempo k y su matriz de co-varianza asociada, se utilizan las predicciones, mediciones y la ganancia de Kalman, las ecuaciones son las siguientes:

$$\vec{X}_k = \vec{X}_k^p + K(Y_k - H\vec{X}_k^p)$$

$$P_k = P_k^p - KHP_k^p$$

Debido a la constante actualización de todos los parámetros y las características de la ganacia de Kalman, es posible iterar sin tener mediciones (Y_k^m), de esta manera en caso de oclusión temporal el algortimo no se queda estancado.

2.8.2. SORT

SORT [21] utiliza el filtro de Kalman para realizar las predicciones, el algoritmo cuenta con 4 partes:

- **Detección:** Se utiliza una CNN para identificar objetos
- **Estimación:** Se utiliza el filtro de Kalman para estimar la futura posición del objeto detectado y se crea una bounding box la posición estimada.
- **Asociación:** Se utiliza el IoU de un objeto en la posición estimada y misma clase de objeto
- **Identificación:** En caso de descubrir un nuevo elemento, se crea un nuevo ID, en caso de que que la asociación haya sido exitosa, se conserva el ID y el caso final consiste en numerosas iteraciones en que el ID no se ha asociado, por lo que es descartado definitivamente.

Este algoritmo tiene dificultades en asociar correctamente ID de objetos de misma clase que intercambian rutas, ya que para asociar el ID solo utiliza la clase, posición estimada e IoU para asociar el ID, esto provoca que hayan múltiples cambios de ID incorrectos. Para solucionar estos problemas se creo el algoritmo DeepSORT.

2.8.3. DeepSORT

DeepSORT [22] agrega una componente de *deep learning* al algoritmo SORT, la cual consiste en utilizar una CNN entrenada y sacarle su capa de clasificación, esto nos deja un vector de características como se puede observar en la siguiente imagen.

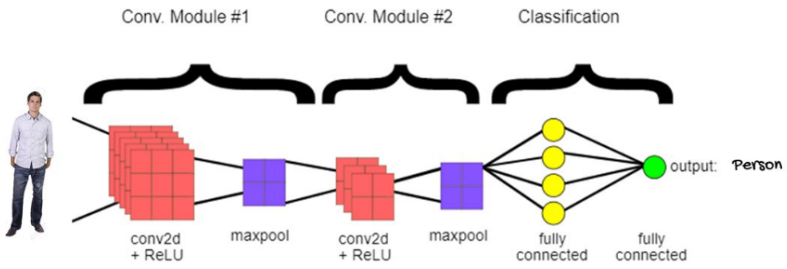


Figura 31: CNN con capa de clasificación [23]

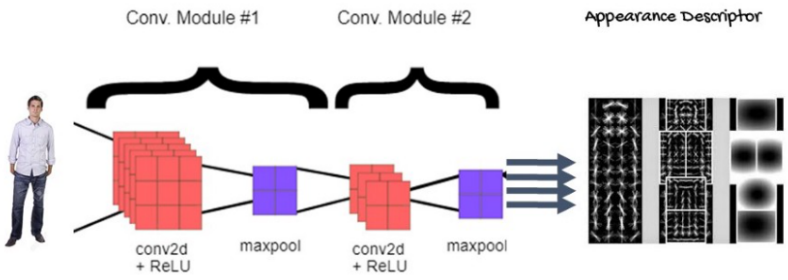


Figura 32: CNN sin capa final de clasificación [23]

Este vector es un descriptor de apariencia, el cual sirve en al etapa de asociación de IDs, de esta manera si dos objetos de la misma clase pero con características distintas intercambian levemente sus rutas, el algoritmo es más propensa a asociar correctamente y en caso de no poder asociar correctamente se crean nuevos IDs, esto genera que la confusión de IDs sea mucho menor en comparación a SORT.

2.8.4. StrongSORT

El algoritmo a utilizar es StrongSORT [24] el cual implementa unas mejoras sobre DeepSORT, estas mejoras consisten en lo siguiente:

- Reemplazar la simple CNN original por una más compleja, entrenada específicamente para obtener características de rostros de personas.
- Se implementa una estrategia para compensar el movimiento de la cámara (ECC)
- El filtro básico de Kalman es susceptible a detecciones de baja calidad, para prevenir eso se utiliza una versión más robusta llamada NSA Kalman algorithm.
- DeepSort utiliza un banco de características (*feature bank*) donde guarda las características de las posiciones de los 100 últimos frames. StrongSORT reemplaza el método en que el *feature bank* encuentra distancias y las asocia por un *exponential moving average* (EMA), esto mejora la calidad de las asociaciones.
- Se le agrega un costo a la información que contiene el movimiento, antes solo la apariencia tenía un costo asociado
- El método de de cascada para elegir los pares de objetos y ID pasaba por información redundante al tener que pasar múltiples veces, por lo que simplemente se utiliza un método de asociación global.

A continuación se puede observar una representación visual de los cambios realizados en StrongSORT.

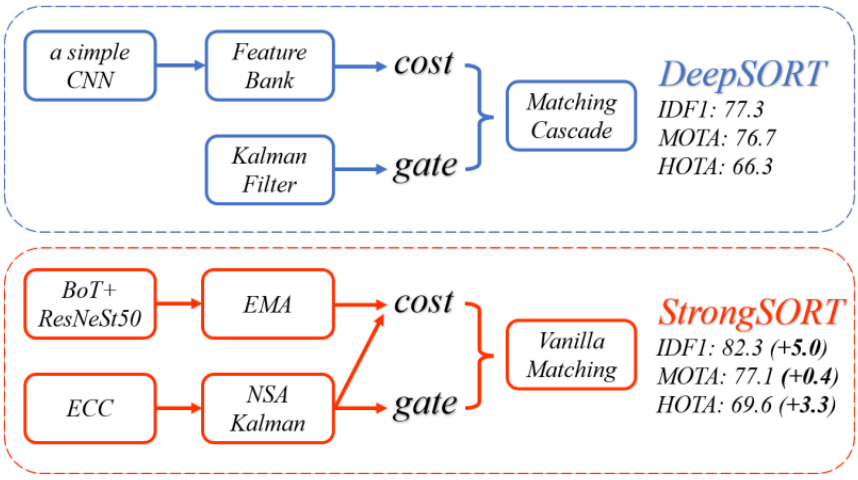


Figura 33: diferencia entre DeepSORT y StrongSORT [24]

2.9. Estado del Arte

Se ha utilizado YOLO para múltiples tareas de identificación en carretera, tales como detección y cuenta de vehículos [25], detección de señales de tránsito [26] y extracción de características de carretera [27]. Todos estos estudios fueron realizados en elementos que poseen diferencias con los elementos nacionales y elementos diferentes a los que se desea clasificar, debido a esto se debe hacer el entrenamiento completo.

Actualmente en *Apsa Ltda* se graban los recorridos y luego se procesan con una herramienta llamada *imajview*, en la que manualmente se seleccionan los elementos y sus características, esto corresponde a una solución semi automática que posee mucho consumo de tiempo.

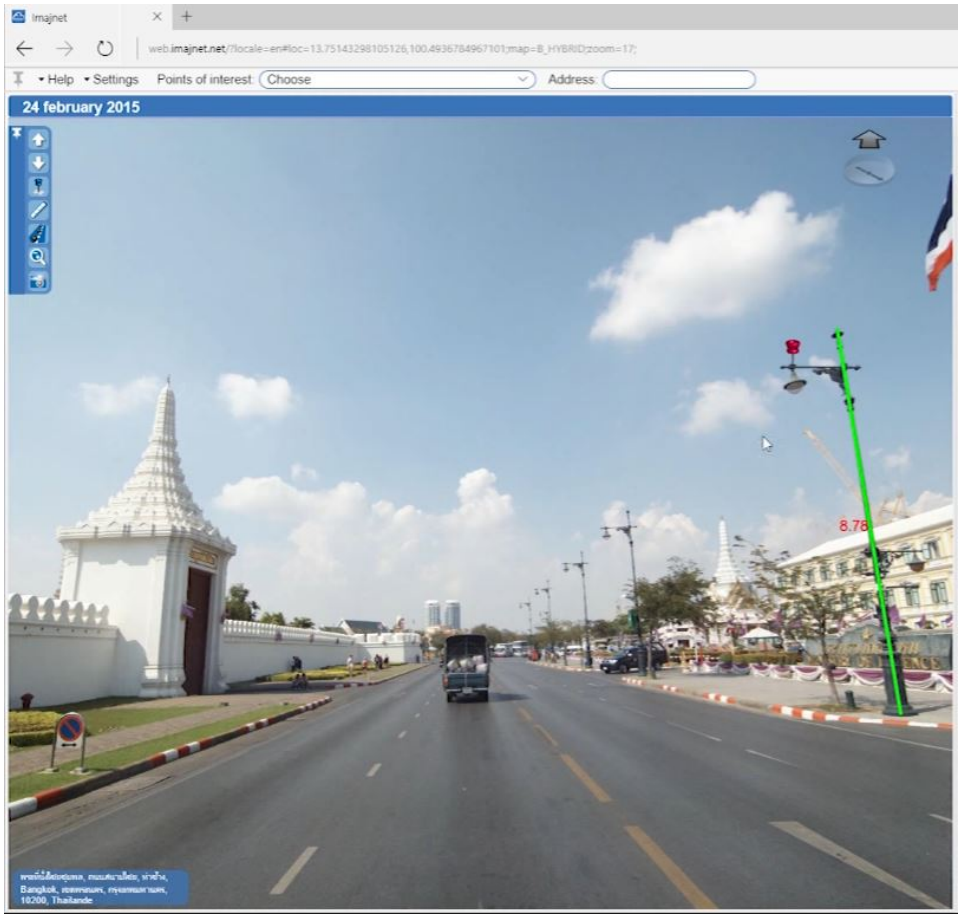


Figura 34: Ejemplo de *Imajview* (fuente: página web de imajview)

Las soluciones planteadas no otorgan solución al problema, ya que los modelos de detección existentes funcionan para elementos no deseados actualmente y los elementos que sí se desean son físicamente diferentes a los encontrados en Chile por lo que los elementos nacionales no serían detectados por el algoritmo, además dentro de los posibles algoritmos, no existe un algoritmo que entregue la información georeferenciada para elementos únicos. Por otra parte la herramienta *imajview*, tampoco soluciona el problema, ya que consume mucho tiempo en etiquetar los elementos manualmente cuando en una ruta existen sobre cientos de elementos a identificar, de los cuales muchos son muy similares entre ellos, es decir se gasta tiempo en identificar elementos se pueden reconocer como patrón, por lo que se busca ahorrar tiempo en este aspecto.

3. Metodología

El trabajo se realizará en el lenguaje de programación **python**, se utilizará un servicio de nube llamado *Google Colaboratory* para entrenar el modelo y evaluarlo, por otra parte se utilizará el entorno local para crear el código general, el cual unirá los múltiples algoritmos a utilizar y generará un DataFrame del inventario, el cual será exportado en formato Excel.

3.1. Obtención de Data

Se decidió utilizar como elemento las luminarias de carretera debido a su alta repetitividad en las rutas de carretera, para obtener la data (imágenes de luminarias de carretera) *Apsa Ltda* entregó videos de rutas y sus archivos KML correspondientes, estos videos contienen las luminarias de interés. Para obtener esta información, se separa el video en sus fotogramas (frames) correspondientes, luego se utiliza la aplicación *labelImg* para realizar las anotaciones de los elementos en formato YOLO para poder entrenar el modelo.

3.2. Preparación de data

A continuación se muestra una imagen obtenida a partir de un video de ruta y la copia de esta imagen pero junto a sus *bounding boxes* para entrenar el modelo, este proceso se realiza con la aplicación *labelImg*.

Figura 35: Foto de ruta

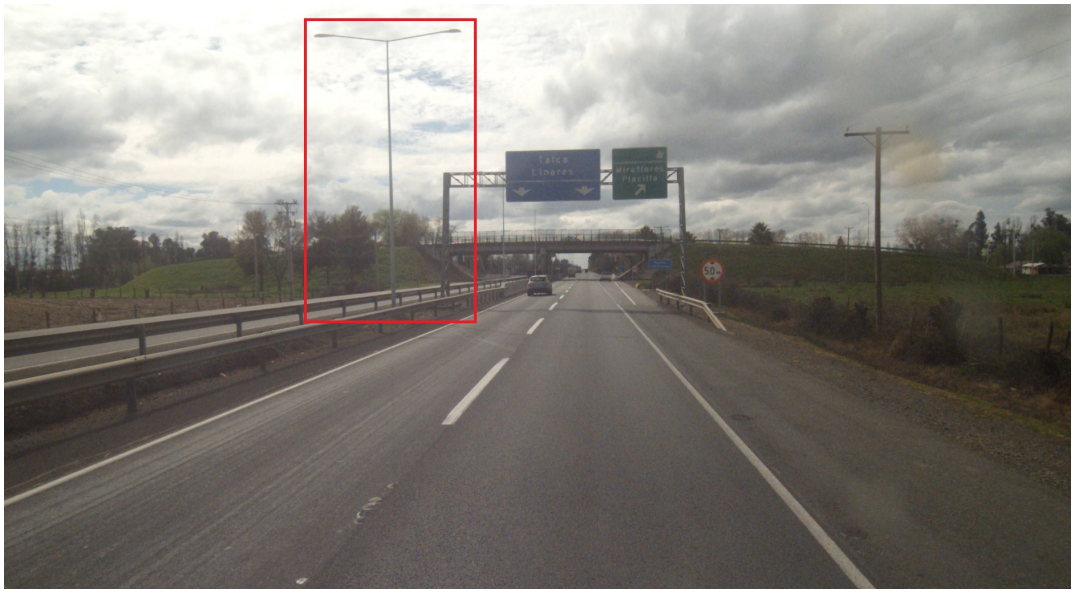


Figura 36: Foto de ruta con *bounding boxes*

Para poder lograr esto, se debe tener un archivo .txt (el archivo debe tener el mismo nombre de la imagen a la que corresponde), este debe contener:

<class id> <x_center> <y_center> <width> <height>

Donde:

- <**class id**> es el número de identificación asociado a la clase correspondiente (van de 0 a num_cls-1).

- <**x_center**> <**y_center**> son los centros del *bounding box*, estos valores son relativos a la dimensión de la imagen, por lo que se encuentran en el rango (0,1].

- <**width**> <**height**> son el ancho y alto del *bounding box*, también relativo a la imagen

Los archivos .txt que contienen los *bounding boxes* deben estar en la misma carpeta que la imagen a la que representan. Para saber que imágenes utilizar para entrenamiento, prueba y validación, se crean archivos .txt con el directorio de las imágenes para cada conjunto.

data/img1.jpg
data/img2.jpg
data/img3.jpg
data/img4.jpg

3.3. Elección de modelo

Debido a la velocidad de computo y precisión de resultados obtenidos se decidió utilizar YOLOv7, el cual posee compatibilidad con StrongSort que es algoritmo de *object tracking* que se utilizará para otorgar IDs a los elementos.

3.4. Tratamiendo de datos

Se utilizará el modelo base de YOLOv7, por lo que las imágenes de entrada deben se de 640 x 640 píxeles, esto significa que se debe hacer un *reshape* de las imágenes, ya que las dimensiones de las imágenes obtenidas del video son de 2448 x 1376 píxeles.

Además se realizaron técnicas de *data augmentation* para poder aumentar la cantidad de imágenes de entrenamiento y crear una mayor diversificación del dataset, aproximadamente 1800 imágenes se utilizaron para entrenamiento, 500 para validación y 100 para prueba. Se sacaron 3 imágenes por cada una de las 1800 imágenes del conjunto de entrenamiento debido a las técnicas de *data augmentation* utilizadas, por lo tanto se ocuparon cerca de 5400 imágenes para entrenar el modelo. Para decidir las características de la imagen obtenida mediante las técnicas de *data augmentation*, se elegía un valor en el rango entregado para cada técnica por separado, generando así un vector el cual como output entregaba la imagen con las modificaciones realizadas. Las técnicas de *data augmentation* utilizadas fueron las siguientes.

Crop

Consiste en recortar la imagen original, se utilizó un rango entre 0 % y 5 %.

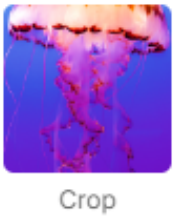


Figura 37: Recorte

Rotation

Consiste en rotar la imagen original, se utilizó un rango de -5° y 5°

Figura 38: Rotación

Saturation

Exagera o disipa la intensidad de los colores en la imagen original, se utilizó un rango de -10 % y 10 %

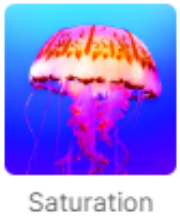


Figura 39: Saturación

Brightness

Aumenta o reduce la cantidad de luz en la imagen original , se utilizó un rango entre -15 % y 15 %.

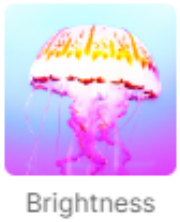


Figura 40: Brillo

Blur

Genera que la imagen se vea borrosa mediante la implementación de un desenfoque Gaussiano (*Gaussian blur*), se utilizó kernels impares de 1x1 hasta 5x5.

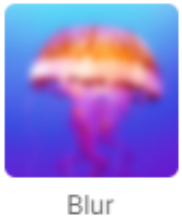


Figura 41: Difuminado

Noise

Para los humanos es simple diferenciar ruido en una imagen, ya que el contexto basta para identificar, pera facilitar este proceso a los algoritmos, se implementar ruido a las imágenes, se implementó hasta un 2 % de los pixeles totales como ruido

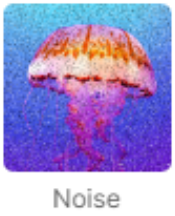


Figura 42: Ruido

3.5. Entrenamiento Object Detection

Como se mencionó anteriormente, se utilizó *Google Colaboratory* para entrenar el modelo de YOLOv7, la GPU utilizada para entrenar fue una **NVIDIA Tesla T4 de 15GB**, se realizaron múltiples entrenamientos y evaluaciones, con diferentes datasets creados mediante variaciones en las técnicas de *Data augmentation* y los valores de estas, el mejor resultado se obtuvo con las técnicas mencionadas en el apartado anterior. En el mejor resultado se realizaron aproximadamente 200 epoch de entrenamiento, llevando un total de aproximadamente 16 horas de entrenamiento.

3.6. Object Tracking

Para realizar el *object tracking* se implementó StrongSORT, los valores de entrada a que recibía StrongSORT era el Tensor de output que generaba el algoritmo de NMS, el cual recibía las el Tensor con las detecciones del modelo YOLOv7. NMS entregaba las coordenadas de los *boudning boxes* en formato xyxy (xy superior izq y xy inferior der) y StrongSORT utiliza el mismo formato de coordenadas que YOLOv7 por lo que se debe regresar al formato xywh.

Una vez realizado todos los cambios de coordenadas, StrongSORT entrega los outputs con sus respectivos IDs, estos outputs se pueden ver alterados bajo los siguientes valores configurables:

- **Ecc**: Corresponde a la posible activación del método de compensación de movimiento de la cámara.
- **Mc_lambda**: Que tan similares deben ser los objetos para compartir el ID.
- **Ema_alpha**: Valor de formula de asociación de elementos.
- **Max_dist**: Distancia máxima entre objetos para ser considerados un posible emparejamiento.
- **Max_IoU_distance**: Distancia máxima entre las posiciones del IoU.
- **Max_age**: cantidad de frames seguidos sin detectar un ID antes de eliminarlo..
- **N_init**: Cantidad de frames que se debe volver a encontrar un objeto consecutivamente antes de ser entregado como elemento con ID por el algoritmo.
- **NN_budget**: Máxima cantidad de posiciones guardadas de un elemento con ID.

Diferentes combinaciones en los valores de configuración de StrongSORT generen variaciones en los ID otorgados, los valores con los que se obtuvieron los mejores resultados fueron los siguientes.

Ecc	Mc lambda	Ema alpha	Max dist	Max IoU distance	Max age	N init	NN budget
True	0.995	0.9	0.2	0.7	7	1	100

Tabla 2: Valores configuracion StrongSORT

3.7. Implementación de georeferencias

Se transformó el contenido del archivo KML en un DataFrame mediante la librería *GeoPandas* y *pandas*, junto con una función auxiliar creada que permitió entregar cada frame con un valor de Latitud, Longitud y Altitud asociado, para calcular la distancia entre 2 puntos diferentes, primero se calculó la Distancia 2D y luego la distancia 3D.

La distancia 2D se calcula utilizando la distancia Geodésica que corresponde a la distancia más corta en la superficie del modelo elipsoide de la Tierra, Esta operación fue realizada mediante una librería llamada GeoPy. Para calcular la distancia 3D se procede a calcular la distancia entre 2 puntos como la hipotenusa de un triángulo, se utiliza la distancia 2D como la base del triángulo y la diferencia de altura como el lado en 90° junto a la base, la distancia entre los 2 puntos corresponde a la hipotenusa del triángulo.

Con esta implementación se puede asociar distancias entre los frames, es decir se obtiene la distancia recorrida por el vehículo que lleva la cámara que graba el video.

3.8. Creación de DataFrame

Para crear el Dataframe se guardan elementos que se les entregó su ID, número de frame en el que se observa el objeto, Distancia métrica del objeto desde el comienzo del video, clase de objeto al que corresponde, el máximo y mínimo valor de seguridad asociado a este elemento, valores de Latitud y Longitud del último frame que observó el elemento identificado.

Para obtener estos valores se probó con listas de valores guardados y diccionarios, se decidió por un combinado entre ambos, donde las llaves del diccionario corresponden a los IDs encontrados y se crean listas con valores asociados a este ID, esto permite una búsqueda rápida de sus elementos y facilita en caso de querer agregar o quitar información asociada al ID.

3.9. Exportación de DataFrame

Para exportar el DataFrame se ocupó la función *pandas.DataFrame.to_ excel*, la cual corresponde a la librería *pandas*, esto genera un Excel con el nombre que se le entregue a la función, se le pueden modificar cosas básicas como por ejemplo el nombre de la hoja y si se desea tener un index.

Junto a la exportación del Dataframe también se decidió hacer una exportación de las imágenes con mejor confianza de predicción de cada elemento identificado, para esto se crea

una carpeta y se guardan las imágenes dentro de esta.

3.9.1. Ejemplo simple de exportación de DataFrame

```
import pandas as pd

# Hay 4 elementos, cada uno con 2 caracteristicas
list1 = [10,20,30,40]
list2 = [40,30,20,10]

col1 = "X"  #Nombre de la columna 1
col2 = "Y"  #Nombre de la columna 2
data = pd.DataFrame({col1:list1,col2:list2})
data.to_excel('sample_data.xlsx', sheet_name='sheet1', index=False)
```

Esto entrega un archivo llado *sample_ data.xlsx*, que contiene lo siguiente:

	A	B
1	**X**	**Y**
2	10	40
3	20	30
4	30	20
5	40	10
6		

Figura 43: Ejemplo Excel

4. Resultados

4.1. Aumento de data

Como bien se mencionó en la metodología, se realizó *data augmentation* para aumentar la cantidad de imágenes y la diversidad de información del conjunto de entrenamiento, a continuación se muestra un ejemplo de una imágen original y 2 imágenes resultantes.

Figura 44: Imagen despés de hacer reshape a 640x640

(a) Variación 1 (b) Variación 2

Figura 45: Resultados *data augmentation*

4.2. Detecciones

El modelo de YOLOv7 fue entrenado aproximadamente por 200 epoch sobre 2 clases de elementos, luminarias de 1 brazo(*1 arm*) y luminarias de 2 brazos(*2 arm*). A continuación se muestran las métricas generales utilizadas para evaluar al modelo YOLOv7, las cuales serán explicadas en la sección de análisis de resultados.

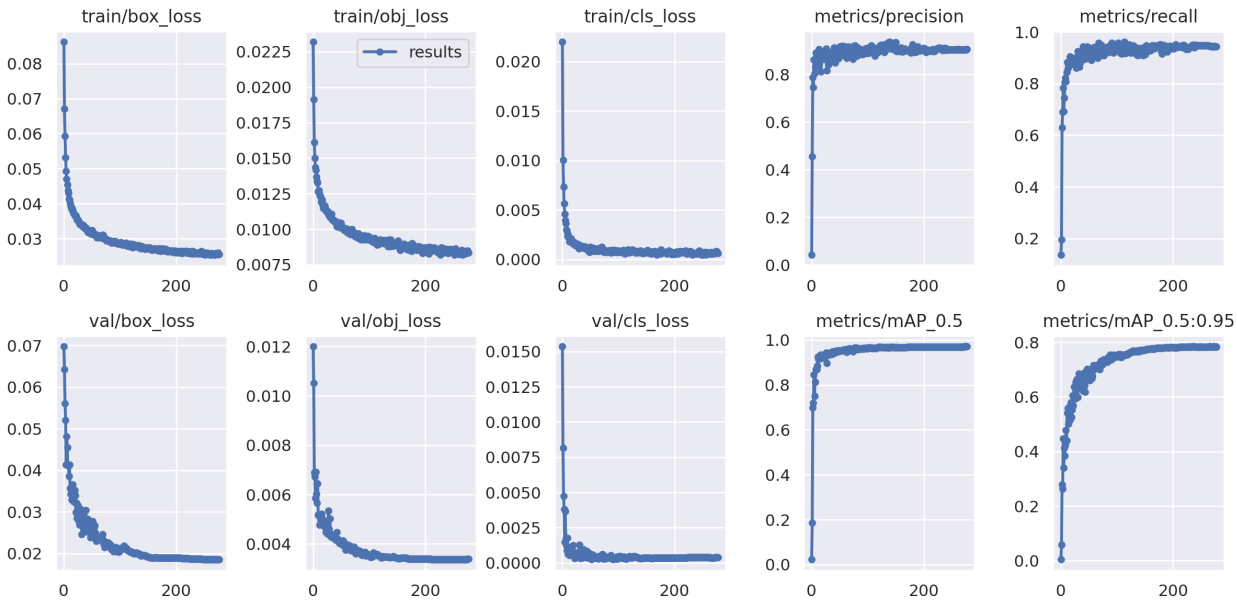


Figura 46: Evaluación del mejor modelo

El modelo obtiene bastantes buenas detecciones tales como se puede observar en la siguiente imagen.

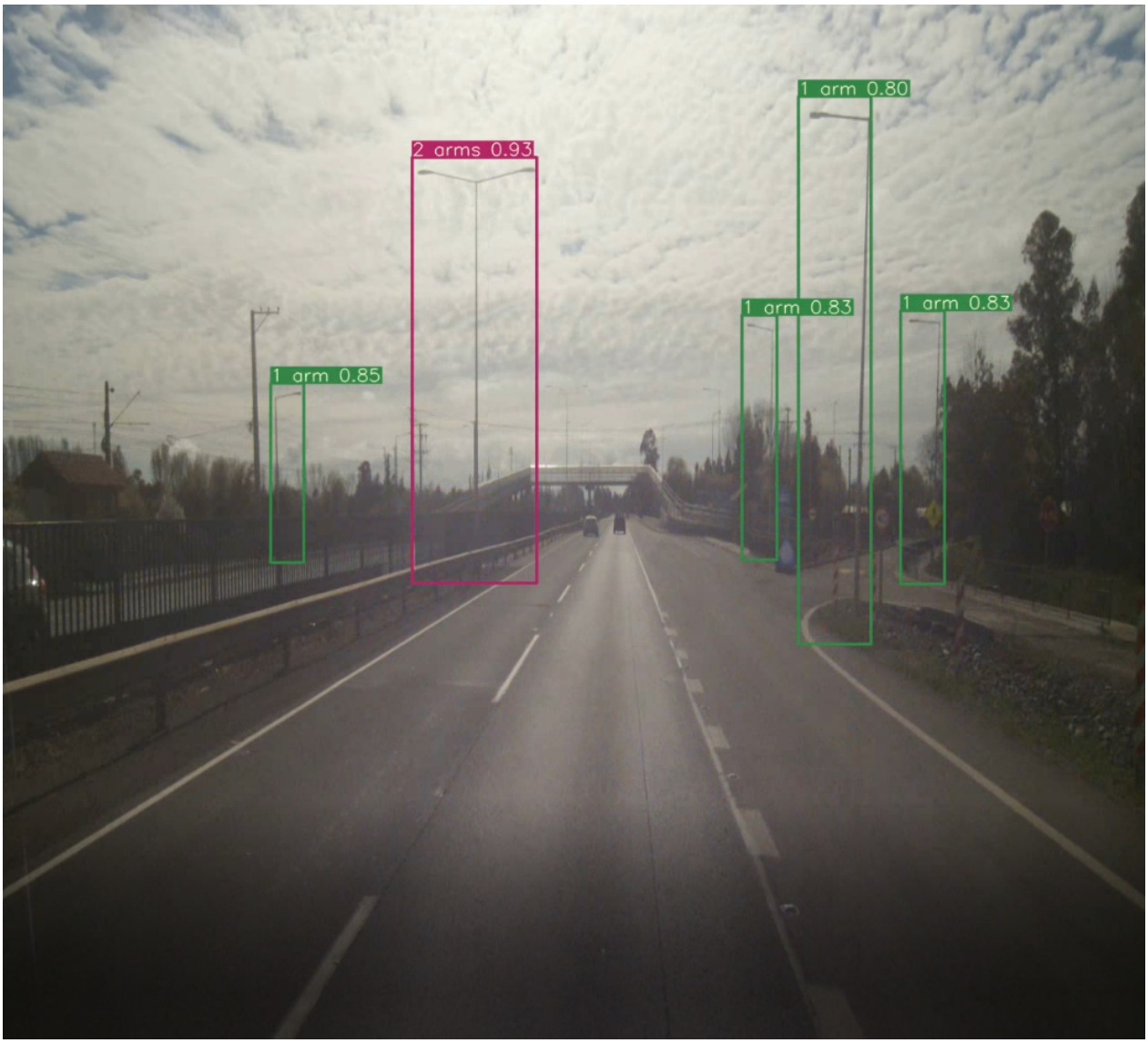
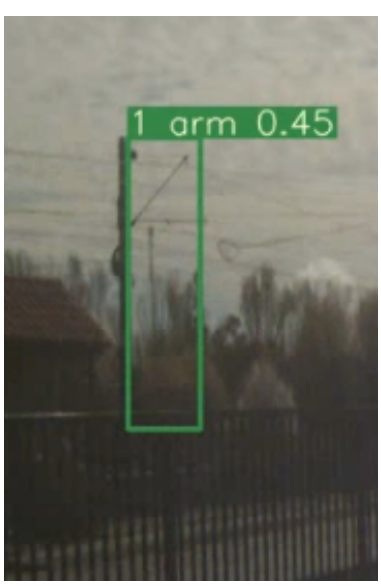
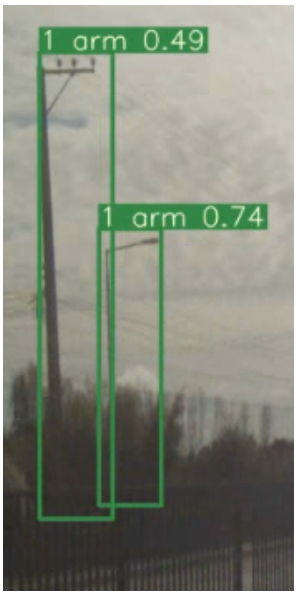


Figura 47: Ejemplos de detecciones

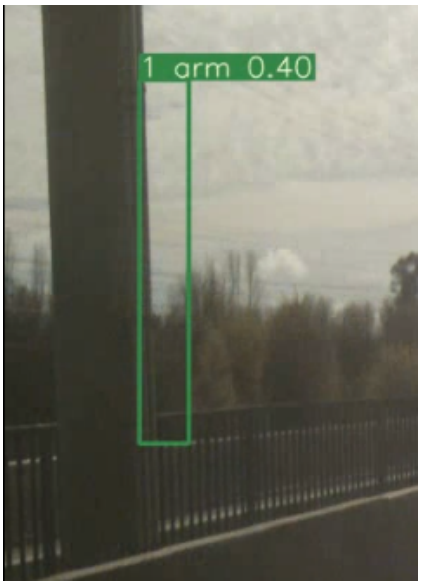
El modelo no es perfecto y en instancias identifica erróneamente objetos o no detecta los que si debería, a continuación se muestran ejemplos.



(a) Poste de tensión



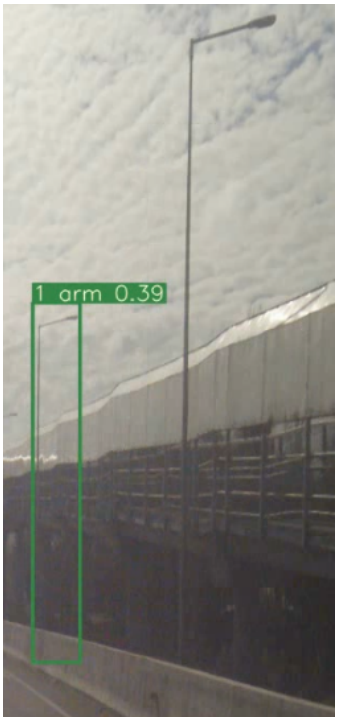
(b) Poste de tensión



(c) Fondo

Figura 48: Detecciones de Falsos Positivos

(a) Luminaria en plano lateral



(b) Reconocimiento izq pero no der

(c) Reconocimiento der pero no izq

Figura 49: Detecciones de Falsos Negativos

4.3. Object tracking

A continuación se van a mostrar dos frames consecutivos obtenidos de un video el cual fue exportado después de haber sido procesado mediante el código creado.

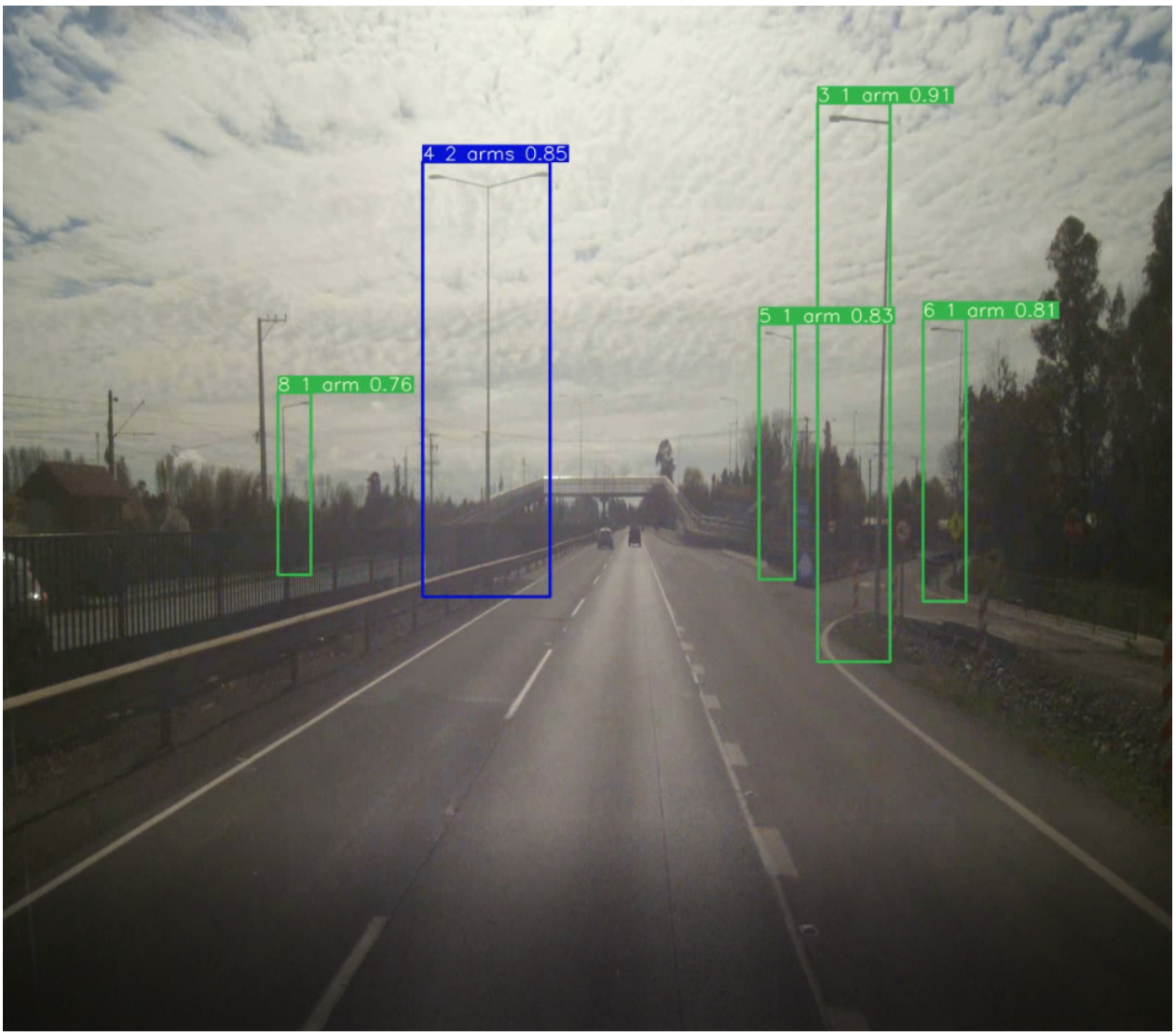


Figura 50: Tracking en frame 1 (IDs base)

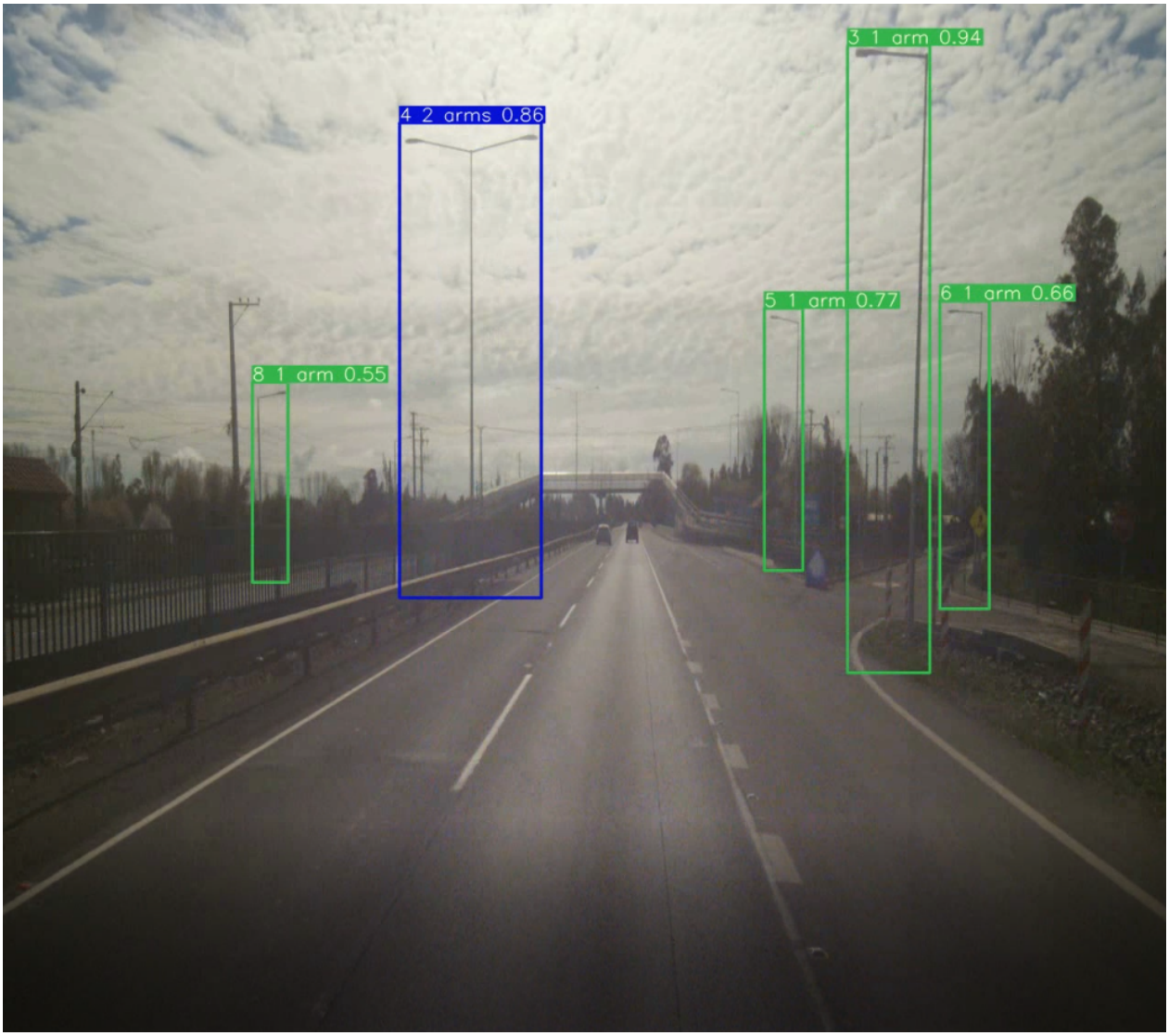


Figura 51: Tracking en frame 2 (Continuación de IDs)

A continuación se muestran errores en el seguimiento de objetos.

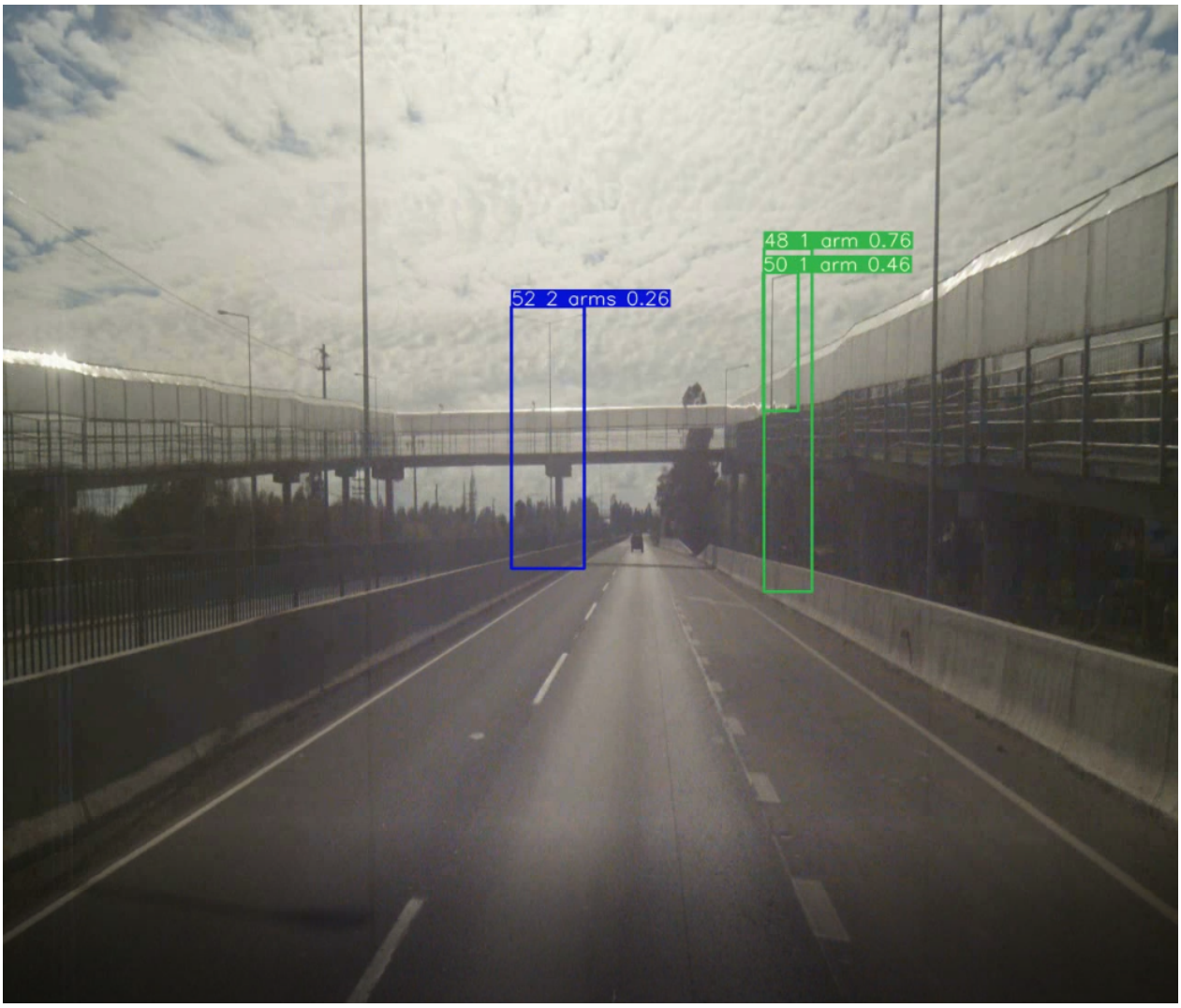


Figura 52: Doble ID para mismo objeto

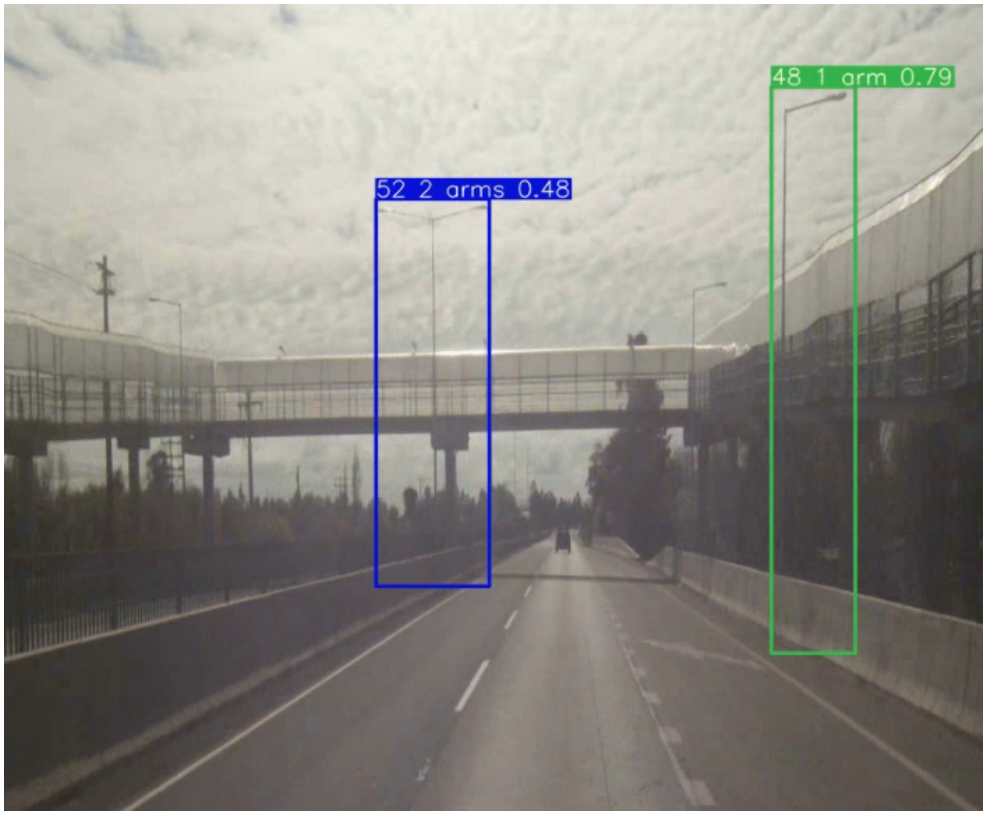


Figura 53: Asociación errónea de ID 48 (primero objeto)

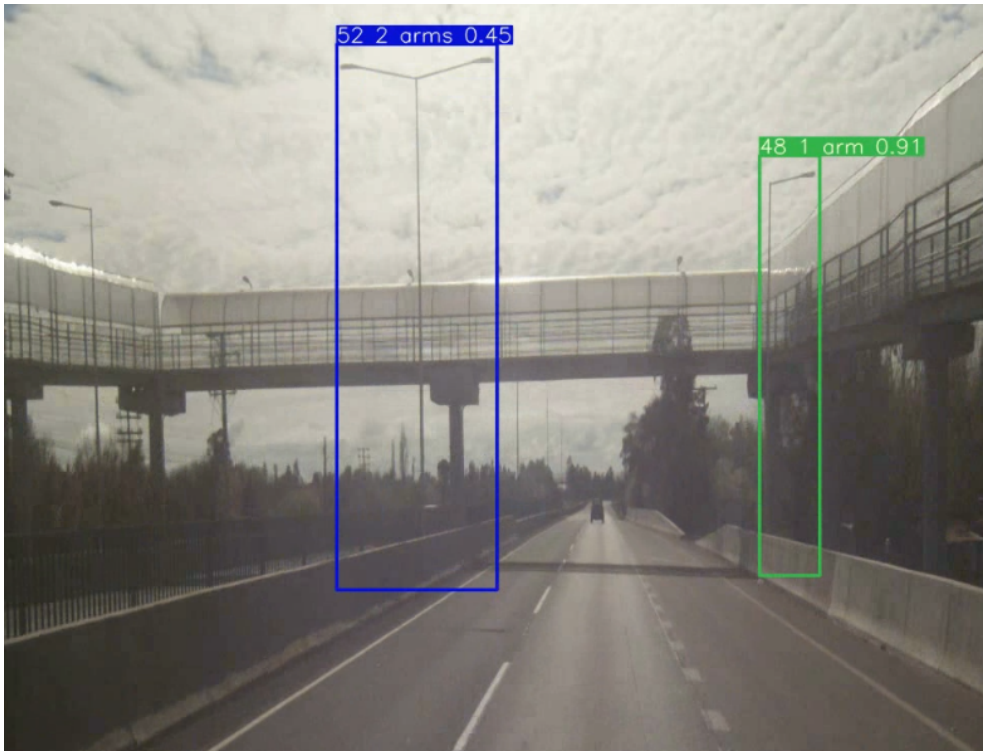


Figura 54: Asociación errónea de ID 48 (segundo objeto)

4.4. Exportación de Dataframe y elementos de interés

Se crea una carpeta con el nombre del archivo KML original y la iteración que corresponde, dentro de esta carpeta se encuentra el DataFrame con la información del inventario en formato Excel, el video con el trackeo completo y una carpeta con imágenes de los elementos Individuales. A continuación se muestra un recorte de la carpeta y sus elementos.

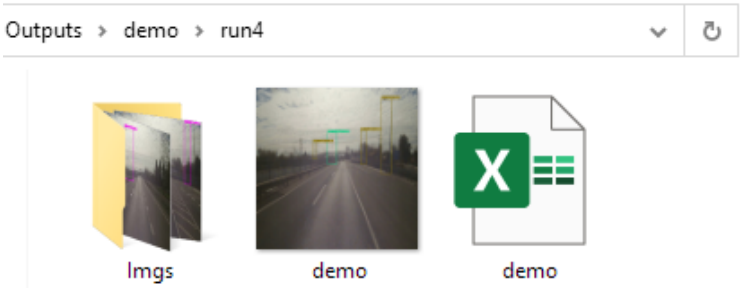


Figura 55: Carpeta con la información exportada

	ID_Objeto	ID_Fotograma	Dist Met (Km)	Clase	Max seguridad	Min seguridad	Latitud	Longitud
0	1	6	0.0294	2 arms	0.9	0.78	-36.0649	-71.7809
1	3	13	0.0732	1 arm	0.95	0.31	-36.0645	-71.7806
2	4	15	0.082	2 arms	0.93	0.6	-36.0645	-71.7805
3	5	19	0.1125	1 arm	0.89	0.71	-36.0643	-71.7803
4	6	22	0.0951	1 arm	0.82	0.43	-36.0644	-71.7805
5	8	24	0.0994	1 arm	0.82	0.29	-36.0643	-71.7804
6	14	35	0.1473	1 arm	0.94	0.31	-36.064	-71.7801
7	22	37	0.1648	2 arms	0.93	0.29	-36.0639	-71.78
8	25	42	0.1779	1 arm	0.88	0.27	-36.0638	-71.7799
9	38	44	0.2174	2 arms	0.94	0.5	-36.0635	-71.7797
10	39	44	0.1868	1 arm	0.73	0.43	-36.0637	-71.7799
11	42	34	0.1429	1 arm	0.43	0.42	-36.064	-71.7802
12	47	47	0.2042	1 arm	0.66	0.34	-36.0636	-71.7798
13	48	56	0.2662	1 arm	0.93	0.29	-36.0631	-71.7794
14	50	52	0.2218	1 arm	0.46	0.34	-36.0635	-71.7796
15	52	59	0.2662	2 arms	0.89	0.26	-36.0631	-71.7794
16	63	70	0.3237	2 arms	0.95	0.47	-36.0627	-71.779
17	64	73	0.3193	2 arms	0.79	0.3	-36.0627	-71.779

Figura 56: Inventario en formato Excel

Figura 57: Imagen en carpeta exportada

5. Análisis de resultados

5.1. Aumento de data

Las técnicas de aumento de data fueron exitosas, ya que permitieron generar un modelo más robusto y generalizado en comparación a ocupar las imágenes sin realizar ninguna modificación. En la figura 45 se puede observar como se alteró la imagen original 44 mediante las técnicas de *data augmentation* mencionadas anteriormente en el apartado de metodología para obtener 3 variaciones, de las cuales se muestran 2, se debe recalcar que como la generación de las imágenes es aleatorio, existe una pequeña posibilidad de que las variaciones sean idénticas.

5.2. Evaluación YOLOv7

En la figura 46 se observan diversas métricas para evaluar el desempeño del modelo utilizado, se debe destacar que el modelo tiene un desempeño muy bueno sobre los datos del conjunto de entrenamiento y validación, esto se puede observar de mejor manera en las siguientes métricas:

- *precision*: Sobre el **90 %**, esto significa que de todos los elementos detectados, sobre un 90 % son detecciones realizadas correctamente
- *recall*: Aproximadamente un **95 %**, esto significa que de todos los elementos posibles para detectar, un 95 % si se detectó y clasificó correctamente
- *mAP 0.5*: Sobe el **95 %** de las detecciones tuvieron un IoU mayor a 0.5, esto demuestra que para cada elemento detectado, escoge las coordenadas correctas del *bounding box*.
- *mAP 0.5:0.95* Significa el promedio de los resultados de mAP cambiando el umbral de 0.5 a 0.95 en un intervalo de 0.05, este promedio de aproximadamente 0.8 demuestra un resultado notable por parte del modelo, ya que baja aproximadamente un 16 % de su mejor posible resultado (mAP 0.5).

5.3. Detecciones

Se puede observar en la figura 47 la cual posee múltiples elementos a ser detectados, estos son identificados sin mayor dificultad, sin embargo el modelo también se puede equivocar como en las múltiples imágenes de la figura 48, donde Realiza detecciones de Falsos positivos, confundiendo diferentes postes de tensión que poseen algo similar a un brazo y también existen ocasiones donde confunde el fondo con un elemento. Se debe recalcar que en estos elementos la confianza de la detección no supera o.5, por lo que si se modificara el umbral de detección se eliminarían estas detecciones, pero esto también puede evitar que se detecten elementos que si deberían detectarse.

En cuanto los Falsos Negativo de la figura 49, esto se pueden deber a que no se entrenó el modelo con suficientes imágenes de luminarias desde una perspectiva alteral, el segundo caso corresponde a un caso bastante frecuente, el cual consiste en que un elemento no sea detectado en un frame específico, pero si es detectado anteriormente o después, esto se soluciona con StrongSORT ya que le puede asociar un ID para cuando si sea detectado.

5.4. Object tracking

Se puede observar como StrongSORT asigna correctamente un ID a los elementos, por ejemplo en la figura 50 y 51 se puede observar como el ID 42 corresponde a la clase de *2 arms*, seguido por la confianza de la predicción, en estas mismas figuras se observa como todos los elementos poseen su ID correspondiente.

Se debe resaltar los casos que pueden complicar las asignación correcta de IDs, en la figura 52 el algoritmo de detección genera 2 *bounding boxes* para el mismo elemento, y en una misma instancia no se puede asociar el mismo ID a múltiples objetos, por lo que se crea un ID extra, duplicando así el objeto en el inventario. Otro error posible es cuando el contenido dentro del *bounding box* de 2 elementos es prácticamente idéntico dentro, esto se puede ver en las figuras 53 Y 54, donde para dos luminarias diferentes se asigna el mismo ID, esto ocurre debido a que el contenido de ambos elementos en sus *bounding boxes* es bastante similar, una posible solución es ser más estricto con los parámetros modificables de asociación de StrongSORT, pero esto puede producir que el mismo elemento no sea asociado con su ID correctamente, ya que es posible que el fondo (contenido que no pertenece estrictamente al elemento) del *bounding box* varíe considerablemente, evitando así que exista un una asociación correcta del elemento y su ID.

5.5. Exportación de DataFrame y elementos de interés

Se crea una carpeta para los elementos a ser exportados (Figura 55), asumiendo que el nombre del archivo KML es el mismo que el de su video asociado, ya que el nombre de la carpeta utiliza el nombre del archivo KML (en este caso 'demo'), Se genera un video para poder evaluar en profundidad lo realizado mediante los algoritmos y modelos utilizados.

Respecto a la utilización de la Latitud, Longitud y Altitud para obtener la distancia métrica, se realizó una aproximación exitosa utilizando la distancia geodésica y una triangulación para considerar el cambio de altitud, esto se puede realizar debido a que la frecuencia de la información georeferenciada es 5 veces por segundo, por lo que el recorrido del vehículo a lo largo de la Tierra entre las dos obtenciones de información georeferenciada es prácticamente recta, el caso sería diferente si el recorrido del vehículo entre dos obtenciones de información georeferenciada fuera algunos Kilómetros, ya que el vehículo habría realizado un recorrido sobre la superficie circular de la Tierra (recorrido en forma de arco) y la aproximación de una distancia recta entre los 2 puntos de medición podría generar un error considerable en la distancia final recorrida.

Para obtener la Latitud y Longitud de cada elemento identificado, se guardan los valores correspondientes a la última posición de la cámara en la que elemento fue detectado, esto es una aproximación que en un futuro puede ser mejorada. El valor de la distancia métrica guardado corresponde a la distancia desde el inicio del recorrido hasta también la posición de la cámara en la última detección del elemento.

Para el DataFrame generado y exportado en formato Excel (ejemplo en figura 56), Se guarda el ID del objeto, El número del frame en que se tiene la mayor seguridad del objeto, la distancia métrica desde el inicio del recorrido (inicio del video) y el punto en que el objeto se ve por última vez, la clase del a la que perteneces el objeto identificado, los valores de máxima y mínima seguridad que se tuvo del objeto, finalmente se guarda la Latitud y Longitud otorgada al elemento.

Finalmente se guarda una carpeta con imágenes individuales de los elementos identificados, esto se puede ver representado en la figura 57 en un futuro se espera poder agregar estas imágenes al DataFrame para tener toda esta información en el Excel.

5.6. Tiempo de ejecución

A lo largo del proceso de creación del código general el tiempo de cómputo varió constantemente, ya que el código general se fue haciendo cada vez más complejo, al final se ejecutó el código en una máquina local y Google Colaboratory para comparar el tiempo de cómputo sin aceleración mediante GPU y con, la CPU utilizada fue un *Ryzen 7 1700X*, para el caso en que **no se utilizaba aceleración con GPU**, el tiempo en analizar un frame era aproximadamente **1.2 segundos**, los videos proporcionados por *Apsa Ltda* estaban compuestos por 5 fotogramas por segundo, por lo que 1 segundo de video era analizado aproximadamente en 6 segundos, para el caso con **aceleración mediante GPU** se utilizó una *NVIDIA Tesla T4 de 15GB*, cada frame se demoraba aproximadamente **0.25 segundos** en ser analizado, esto significa que al utilizar aceleración de GPU en este caso particular el tiempo de cómputo de reducía aproximadamente a $\frac{1}{5}$ en comparación a no utilizar la aceleración por GPU, se debe recalcar que diferentes combinaciones de CPU y GPU entregan diferentes tiempos de cómputo.

Asumiendo una velocidad promedio de 80 Km/h, en el caso **sin aceleración de GPU** se analizan aproximadamente **13 Km por hora de procesamiento** y para el caso **con aceleración de GPU** se analizan aproximadamente **64 Km por hora de procesamiento**. Se debe recalcar que a medida que se agreguen más etapas de análisis al código general, el tiempo de cómputo aumentará, por lo tanto los Kilómetros analizados por hora de procesamiento disminuirán. Además no todas las GPU son compatibles con PyTorch CUDA que es el programa que permite realizar la aceleración mediante GPU en el algoritmo de detección YOLOv7.

6. Conclusiones

A lo largo de esta Memoria se avanzó el proceso de automatización que *Apsa Ltda* está investigando y evaluando. El trabajo realizado es una aproximación que aborda múltiples áreas de importancia como lo son el tipo de objeto, la posición del objeto respecto al inicio del recorrido y su información georeferenciada, estas aproximaciones son simples pero demuestran que es posible automatizar el proceso. Dado los alcances de un trabajo de título se lograron los objetivos, pero efectivamente estos trabajos son muy grandes y se deben seguir trabajando.

Exitosamente se creó manualmente el dataset original a partir de videos proporcionados por *Apsa Ltda*, la herramienta labelImg fue de gran ayuda, facilitando el proceso debido a la simpleza y rapidez de la herramienta. Así mismo la creación de un dataset más generalizado mediante la utilización de técnicas de *data augmentation*, esto mejoró la capacidad de detección de objetos del modelo YOLOv7, ya que presentaba una mejor generalización y tolerancia a diferentes cambios como leves rotaciones, cambios en la luminosidad, entre otros.

La metodología planteada permitió cumplir con el objetivo principal exitosamente, desde la obtención de data, pre-procesamiento de esta, múltiples iteraciones de entrenamiento del modelo de detección YOLOv7 y el modelo de seguimiento de objetos StronSORT, la obtención de información georreferenciada y la utilización de esta, seguido por la creación del DataFrame que contiene la información deseada del inventario y la exportación de este en formato Excel.

El entrenamiento del modelo YOLOv7 como bien se mencionó en la metodología, tomaba aproximadamente 16 horas, pero se probó con diferentes intervalos de epochs por lo que en ocasiones se demoraba más y en otras menos, seguido por errores en los conjuntos de datos, el cambio de estos mediante implementación de diversas técnicas de *data augmentation* y la curiosidad por saber si mínimos cambios mejorarían el resultado, todo esto provocó que la mayor parte del tiempo se gastara generando el modelo de detección de objetos, el cuál es la base para el correcto funcionamiento del resto de algoritmos utilizados.

A través de los resultados se observa que los resultados no son perfectos, pero permiten automatizar el proceso más de un 90 % de las veces. Esto demuestra que el ser humano sigue siendo una pieza importante del proceso. Sin embargo debido a la estructura de los modelos YOLOv7 y StrongSORT es posible seguir mejorando los resultados a lo largo del tiempo, ya sea con más data o incluso trasladar la data actual a futuros algoritmos que tengan mejores resultados en un tiempo similar. También se pudo observar como StrongSORT compensa errores de detecciones y clasificación de YOLOv7, ya que posee una fase de inicialización antes de entregar un ID, esto permite que detecciones erróneas exclusivas de un solo frame, no sean tomadas en cuenta, reduciendo así errores potenciales.

Uno de los ámbitos más importantes son el tiempo de procesamiento, ya que *Apsa Ltda* buscaba optimizar los tiempos de procesamiento. Este ámbito se efectuó de manera exitosa, ya que el código realizado es capaz de analizar 13 Km por hora de procesamiento (Sin aceleración de GPU) y 64 Km por hora de procesamiento (con aceleración de GPU), es decir, se podría analizar la ruta de Talca - Chillán en aproximadamente 12 hrs y 2.5 hrs de procesamiento respectivamente, como bien se mencionó en el análisis de tiempo de ejecución, a medida que se agreguen más etapas de análisis al código, más tiempo se va a demorar en procesar la misma cantidad de Km.

Con todo lo mencionado anteriormente, se logró cumplir con el objetivo general de entregar un código que produjera el inventario de luminarias de carretera mediante el análisis de un video y su archivo KML asociado, el proceso fue realizado exitosamente cumpliendo los objetivos específicos propuestos y siguiendo una metodología que permite entrenar modelos iterativamente, cambiando parámetros o conjuntos de data para así obtener los mejores modelos posibles en la ventana de tiempo disponible.

6.1. Trabajo futuro

Este proyecto tiene potencial, ya se demostró que es posible realizar una automatización de la creación de inventario a partir de video y los resultados actuales son mejorables, entre las mejoras realizables para el trabajo futuro se encuentran:

- **Añadir más clases**: El agregar más clases de objetos implica entrenar un modelo desde cero, ya que no es posible modificar la estructura de las redes neuronales y generar un output extra, por lo que se requiere gastar tiempo en entrenar un modelo nuevo cada vez que se desea agregar una nueva iteración de clases.

- **Mejorar referencias geográficas**: Actualmente se utiliza una aproximación simple, donde se utiliza como referencia geográfica la última instancia en la que se identificó el elemento. Esto se puede mejorar al implementar un algoritmo de distancia y profundidad, de esta manera se puede aproximar la posición del objeto relativa a la posición de la cámara, obteniendo así una referencia geográfica más precisa.

- **Agregar active learning**: Consiste en otorgarle al algoritmo la capacidad de guardar data con la que no fue entrenada y en caso de ser necesario consultar a un usuario para verificar el contenido de esta, esta data nueva se agrega a la data anterior para ser ajustado, esto permite que el modelo sea mejorado constantemente a lo largo del tiempo, obteniendo cada vez modelos más generalizables y confiables.

- **Segregar zonas de interés**: Actualmente el código evalúa los algoritmos en todo los píxeles de la imagen entregada, pero esto no es exactamente lo que se busca, ya que esto detecta elementos fuera de ruta, es decir elementos no deseados. Debido a esto es importante en un futuro agregar un algoritmo que solo permita la detección de elementos que se encuentran en la ruta deseada.

- **Creación de aplicación**: Para facilitar el uso del código y la interacción del usuario con los algoritmos, es necesario construir una aplicación intuitiva, para que personas no especialistas en el área de visión computacional puedan utilizar.

- **Actualizaciones futuras**: Es necesario estar al tanto de posibles nuevos algoritmos que sean capaces de mejorar los resultados actuales, por esto es necesario estar actualizado con los algoritmos del estado del arte en las áreas de detección de objetos, seguimiento de objetos y geolocalización de objetos.

7. BIBLIOGRAFÍA

[1] S. Sharma, "What the hell is perceptron?" 2017.

[2] A. Géron, in *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, ser. 2nd edition. O'Reilly Media, Inc., 2019, pp. 284–294.

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[4] T. Ojala, M. Pietikainen, and D. Harwood, "Performance evaluation of texture measures with classification based on kullback discrimination of distributions." IEEE, 1994.

[5] S. Saha, "A comprehensive guide to convolutional neural networks - the eli5 way," 2018.

[6] P. Ganesh, "Types of convolution kernels: Simplified," 2019.

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

[8] J. Jeong, "The most intuitive and easiest guide for convolutional neural network," 2019.

[9] S. Khosla, "Cnn | introduction to pooling layer," 2019.

[10] M. J. Awan, O. A. Masood, M. A. Mohammed, A. Yasin, A. M. Zain, R. Damaševičius, and K. H. Abdulkareem, "Image-based malware classification using vgg19 network and spatial convolutional attention," *Electronics*, vol. 10, no. 19, p. 2444, 2021.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] M. Drożdż and T. Kryjak, "Fpga implementation of multi-scale face detection using hog features and svm classifier," in *IPC*, vol. 21, no. 3, 2016, pp. 27–44.

[13] P. Chhikara, "Intuition and implementation of non-max suppression algorithm in object detection," 2022.

[14] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE, 2020, pp. 237–242.

[15] J. Hui, "map (mean average precision) for object detection," 2018.

[16] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and*

pattern recognition, 2016, pp. 779–788.

[18] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[20] J. Teow, "Understanding kalman filters with python," 2018.

[21] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[22] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[23] R. Kanjee, "Deepsort — deep learning applied to object tracking," 2020.

[24] Y. Du, Y. Song, B. Yang, and Y. Zhao, "Strongsort: Make deepsort great again," *arXiv preprint arXiv:2202.13514*, 2022.

[25] H. Song, H. Liang, H. Li, Z. Dai, and X. Yun, "Vision-based vehicle detection and counting system using deep learning in highway scenes," vol. 11, no. 1. Springer, 2019, pp. 1–16.

[26] J. Wan, W. Ding, H. Zhu, M. Xia, Z. Huang, L. Tian, Y. Zhu, and H. Wang, "An efficient small traffic sign detection method based on yolov3," vol. 93, no. 8. Springer, 2021, pp. 899–911.

[27] H. M. Eraqi, K. Soliman, D. Said, O. R. Elezaby, M. N. Moustafa, and H. Abdelgawad, "Automatic roadway features detection with oriented object detection," vol. 11, no. 8. MDPI, 2021, p. 3531.