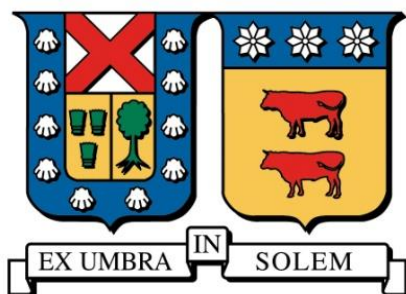


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INGENIERÍA MECÁNICA
SANTIAGO – CHILE



**“Detección y Segmentación de módulos PV en imágenes
RGB aéreas para análisis en condición de operación”**

Claudio Sebastián Hurtado Astorga

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE:
INGENIERO CIVIL MECÁNICO**

PROFESOR GUÍA: Ph.D. Rodrigo Barraza Vicencio
PROFESOR CORREFERENTE: MSc. Robinson Cavieres Abarca

Agosto de 2022

Agradecimientos

Quiero agradecer a todas las personas que me han acompañado durante esta etapa y a lo largo de mi vida influyendo en quien soy hoy.

A mi pareja Catalina y a toda mi familia, en especial a mis padres Yenny y Claudio, a mis hermanos Francisca y Martin quienes siempre me han apoyado en todos los momentos.

A mis amigos y compañeros de universidad, en especial, Cote, Daniel G, Daniel A, Thomas, Pablo y Bryan con quienes compartí grandes momentos, estoy muy agradecido por las risas, ayuda y amistades que formamos durante esta etapa.

A mis amigos del liceo Chelo, Polaco, Mora, Matías, Jessica y Pablo, en quienes siempre puedo confiar y encontrar apoyo, gracias por todos los momentos vividos.

Resumen

En el presente trabajo se desarrolla un modelo basado en técnicas de machine learning para poder detectar y segmentar módulos fotovoltaicos en imágenes RGB. Tras un breve estudio del estado del arte en análisis de imágenes se opta por desarrollar un modelo basado en Mask R-CNN, ya que es capaz de detectar y segmentar objetos en imágenes. Para lograr este objetivo se genera un conjunto de ejemplos de 110 imágenes con ambientes o fondos complejos, segmentando manualmente los módulos PV dentro de estas, con las cuales se logra un entrenamiento efectivo del modelo desarrollado. A partir de este entrenamiento el modelo es capaz de detectar y segmentar paneles fotovoltaicos en ambientes complejos logrando una precisión de 99.6%, una puntuación F1 de 0.97 y un parámetro de pérdida de área segmentada “loss” del 15%, lo que demuestra la capacidad del modelo y su fiabilidad para detectar y segmentar imágenes futuras.

Abstract

In the present work, a model based on machine learning techniques is developed to be able to detect and segment photovoltaic modules in RGB images. After a brief study of the state of the art in image analysis, it's decided to develop a model based on Mask R-CNN, since it's capable of detecting and segmenting objects in images. To achieve this goal, a set of examples of 110 images with complex backgrounds is generated, manually segmenting the PV modules in the images, with which an effective training of the developed model is achieved. Based on this training, the model is capable to do an instance segmentation of photovoltaic panels in complex environments, achieving an accuracy of 99.6%, an F1 score of 0.97 and a segmented area loss parameter call “loss” of 15%, which demonstrates the ability of the model and its reliability to detect and segment future images.

Glosario

COCO Dataset	: Conjunto de imágenes “Common Objects in Context”.
Dron	: Pequeño vehículo aéreo no tripulado.
Épocas	: Un pase de entrenamiento completo sobre todo el conjunto de datos de modo que cada ejemplo se haya visto una vez.
Falso negativo	: Sobreestimación por parte del modelo.
Falso positivo	: subestimación por parte del modelo.
Faster R-CNN	: Red convolucional profunda utilizada para la detección de objetos.
Feature	: Una variable de entrada utilizada para hacer predicciones.
Feature map	: Mapa de características.
Función de activación:	Una función que toma la suma ponderada de todas las entradas de la capa anterior y luego entrega un valor de salida a la siguiente capa.
F1 score	: Media armónica de precisión y recuperación.
Label	: En el aprendizaje supervisado, la parte de "respuesta" o "resultado" de un ejemplo.
Layer	: Un conjunto de neuronas en una red neuronal que procesa un conjunto de características de entrada o la salida de esas neuronas.
Lote o “Batch”	: El conjunto de ejemplos utilizados en una iteración del entrenamiento del modelo.
Machine learning	: Aprendizaje de máquinas.
Mask R-CNN	: Red convolucional profunda utilizada para la detección y segmentación de objetos.
Pasos por época	: Una evaluación hacia adelante y hacia atrás de un lote.
Pérdida	: Porcentaje de objeto no segmentado.

Precisión	: Identifica la frecuencia con la que un modelo fue correcto al predecir la clase positiva.
RGB	: Colores rojo, verde y azul para crear el espectro visible en imágenes.
ReLU	: Rectified Lineal Unit.
ROI	: Regiones de interés.
ROI Allign	: Ajusta el tamaño de las regiones de interés sin cometer redondeos.
ROI pooling	: Ajusta el tamaño de las regiones de interés.
Sobreajuste	: Crear un modelo que coincida tanto con los datos de entrenamiento que el modelo no pueda hacer predicciones correctas sobre nuevos datos.
Soiling	: Ensuciamiento de la cara frontal del panel fotovoltaico.
String	: Conjunto de paneles solares conectados en serie.
Tasa de aprendizaje	: Un escalar utilizado para entrenar un modelo a través del descenso de gradiente.
Verdadero negativo	: Un ejemplo en el que el modelo predijo correctamente la clase negativa.
Verdadero positivo	: Un ejemplo en el que el modelo predijo correctamente la clase positiva.

Índice de contenidos

Agradecimientos	1
Resumen	2
Abstract.....	2
Glosario	3
Índice de contenidos	5
Listado de figuras	7
1. Introducción	9
1.1. Introducción general	9
1.2. Objetivo general.....	10
1.3. Objetivos específicos	10
2. Estado del arte	11
2.1. Métodos convencionales.....	11
2.2. Métodos mediante Machine Learning	11
3. Modelo de Mask R-CNN	19
3.1. Faster R-CNN	19
3.2. Mask R-CNN	22
3.3. Alteraciones al modelo original de Mask R-CNN.....	24
3.4. Configuraciones del modelo	25
3.5. Especificaciones de la máquina	27
3.6. Requerimientos de entrada.....	28
4. Conjunto de ejemplos o dataset.....	30
4.1. Obtención de imágenes.....	30
4.1.1. Dron DJI Matrice 200.....	30

4.1.2.	Cámara ZENMUSE XT2	31
4.2.	Conjunto de pre-entrenamiento	32
4.3.	Segmentación manual para imágenes de entrenamiento	33
4.4.	Agrupación de ejemplos	34
5.	Resultados y análisis	35
5.1.	Formato de resultados	35
5.2.	Métricas de desempeño.....	38
5.2.1.	Orden impuesto	40
5.2.2.	Orden aleatorio	41
5.3.	Comparación.....	44
6.	Conclusiones	46
7.	Trabajos futuros	47
8.	Bibliografía	48
	Anexos.....	51
	Anexo 1: Softwares y librerías de Python	51
	Anexo 2: Código en lenguaje Python utilizado	51

Listado de figuras

Figura 1: Distribución del conjunto de ejemplos.....	12
Figura 2: Proceso de entrenamiento para un modelo de aprendizaje supervisado [9].	13
Figura 3: Esquema de red neuronal en machine learning [9].	14
Figura 4: diferencias entre detección y segmentación.....	15
Figura 5: Resultado de Faster R-CNN [11].	16
Figura 6: Resultado de la detección y segmentación utilizando Mask R-CNN.	18
Figura 7: Funcionamiento de Faster R-CNN.....	20
Figura 8: Ejemplo de imagen en conjunto de validación y su mapa de características.....	20
Figura 9: Regiones de interés para imagen de ejemplo.	21
Figura 10: Imagen resultante de Faster R-CNN.	22
Figura 11: Mapa conceptual de una Mask R-CNN.	23
Figura 12: Resultado de Mask R-CNN.....	24
Figura 13: Comportamiento de función ReLU [19].	26
Figura 14: Ejemplo de imagen ideal como input del modelo.....	29
Figura 15: Imagen referencial de dron utilizado [22].....	31
Figura 16: Imagen referencial de cámara utilizada [24].....	31
Figura 17: Imagen perteneciente a COCO Dataset [25].....	33
Figura 18: Resultado de segmentación original de Mask R-CNN.	36
Figura 19: Resultado de segmentación utilizando el comando "splash".	37
Figura 20: Resultado con módulos en imágenes separadas.....	38
Figura 21: Matriz de confusión [29].....	39
Figura 22: Matriz de confusión utilizando el primer orden del conjunto de ejemplos.....	40
Figura 23: Gráfico de loss vs épocas para el primer orden.	41
Figura 24: Matriz de confusión utilizando el segundo orden del conjunto de ejemplos.	41
Figura 25: Gráfico de loss vs épocas para el segundo orden.....	42
Figura 26: Matriz de confusión utilizando el tercer orden del conjunto de ejemplos.	42
Figura 27: Gráfico de loss vs épocas para el tercer orden.	43
Figura 28: Matriz de confusión utilizando el cuarto orden del conjunto de ejemplos.	43

Figura 29: Gráfico de loss vs épocas para el cuarto orden.	44
---	----

1. Introducción

1.1. Introducción general

Actualmente el ser humano se ve en una problemática con respecto al uso de energía eléctrica y al cambio climático, la generación de esta energía es responsable de aproximadamente dos tercios de las emisiones mundiales de CO_2 [1] y el consumo de esta va en un constante aumento, es por esto que es necesario cambiar las fuentes de la matriz energética global, en Chile la Comisión Nacional de Energía notifico que en enero de 2022 un 16% de la capacidad instalada en la matriz energética corresponde a plantas fotovoltaicas.

Los módulos fotovoltaicos utilizados en plantas solares tienen una vida útil estimada entre 25 a 30 años según los fabricantes [2], pero esta vida útil puede ser más corta debido a problemas de transporte, instalación y durante su operación. Existen distintos tipos de fallas, las cuales se definen como cualquier efecto negativo que impacte en la producción de energía y que no es reversible a través del funcionamiento normal [3], estas fallas se pueden clasificar dentro de tres áreas, fallas durante el inicio de la vida del panel, en donde se incluyen fallas durante la manufactura y la instalación, fallas en la mitad de su vida, relacionadas a las condiciones de operación y las fallas al final de la vida del módulo en donde ya deja de ser útil. Todas estas fallas implican pérdidas de energía y acortan la vida útil del panel, por lo que es imprescindible crear estrategias de mantenimiento que logren identificar las fallas en fases tempranas para asegurar su correcto funcionamiento y vida útil.

Existen distintos problemas o fallas que pueden ser analizados por medio de imágenes RGB como lo son el “cracking” o trizaduras, fatiga de los materiales, sombras parciales y el soiling, aunque este último no es considerado como una falla según la definición de Köntges [3], pero es necesario su estudio, ya que, logra disminuir significativamente la producción de energía de cada módulo. El ensuciamiento o la acumulación de polvo disminuye la transmisión de radiación incidente a través del cristal de protección del módulo, reduciendo así la radiación solar que logra ser convertida en energía eléctrica. Múltiples estudios tanto

en Chile como en el mundo muestran que estas pérdidas de energía van desde un 3% hasta un 50% exclusivamente por la acumulación de soiling en el panel [4], [5].

Es por esto que en el presente trabajo se plantea el desarrollo de un modelo capaz de detectar y segmentar módulos fotovoltaicos dentro de un “string” o arreglo PV en imágenes RGB a partir del modelo desarrollado por Cavieres [6] mediante técnicas de machine learning, las imágenes utilizadas para el entrenamiento son obtenidas de plantas fotovoltaicas operativas en Chile mediante un dron (vehículo aéreo no tripulado).

1.2. Objetivo general

Desarrollar el proceso de detección y segmentación en imágenes RGB para identificar módulos en un arreglo fotovoltaico mediante técnicas de machine learning.

1.3. Objetivos específicos

1. Analizar metodología utilizada en el modelo anterior y el estado del arte relevante en procesamiento de imágenes.
2. Recopilar y adaptar nuevas imágenes para ser utilizadas en el entrenamiento del proceso de detección y segmentación.
3. Desarrollar modelo para lograr la segmentación de módulos PV mediante el nuevo conjunto de imágenes.

2. Estado del arte

La investigación que se realiza para este trabajo está centrada en las formas que existen para detectar y segmentar objetos en imágenes, a continuación, se muestran distintas formas desde las más simples a más complejas.

2.1. Métodos convencionales

- Segmentación manual

Es de los primeros métodos utilizados en donde un operador manualmente seleccionaba los paneles fotovoltaicos dentro de las imágenes para luego someterlas a distintos análisis de falla.

- Métodos Estadísticos

El más utilizado es el método basado en umbrales de colores dentro de las imágenes, en donde se utiliza un filtro dentro de la imagen y se le entrega cual es el color que debe buscar la máquina dentro de esta, en investigaciones anteriores como los trabajos realizados por Grimaccia [7] se ha utilizado el color azul del panel como parámetro de búsqueda, así el programa ejecutará algoritmos para encontrar donde es más probable que se encuentre el color deseado, es decir, los paneles fotovoltaicos. Este método no es recomendable actualmente, ya que, se necesitan extensos ajustes manuales de los hiperparámetros, conlleva un gran gasto computacional y generalmente no tienen gran eficacia para generalizar el método para ser utilizado en imágenes nuevas.

2.2. Métodos mediante Machine Learning

Para poder entender de mejor manera los distintos modelos de machine learning es necesario primero introducir lo que son los modelos de aprendizaje automático supervisado, los cuales se pueden entender como modelos que buscan predecir una etiqueta verdadera $\mathbf{y}$ luego de ser entrenados con ejemplos etiquetados $(\mathbf{x}, \mathbf{y})$, siendo la predicción o inferencia $\mathbf{y}'$

obtenida al aplicar el modelo entrenado a ejemplos sin etiqueta $(x, ?)$ [9]. En donde se utilizan atributos (x) los cuales son variables de entrada que describen los datos y etiquetas (y) las que corresponden al valor verdadero que se busca predecir.

Como se mencionó anteriormente es necesario entregar un conjunto de ejemplos al modelo para que este pueda aprender y realizar las predicciones, este conjunto se separa comúnmente como se muestra en la Figura 1.

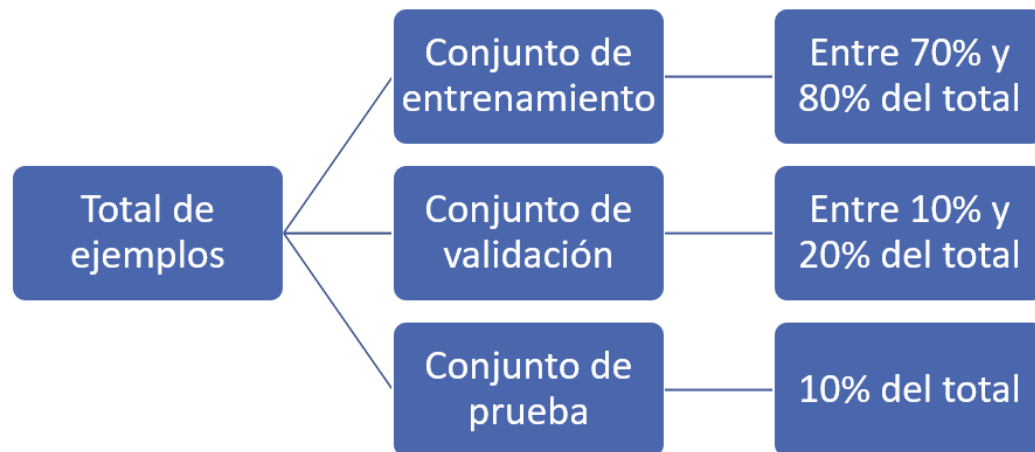


Figura 1: Distribución del conjunto de ejemplos.

Con el conjunto total de ejemplos ya separado se puede comenzar el proceso cíclico de entrenamiento ejemplificado en la Figura 2, todo inicia entrenando el modelo con el conjunto de entrenamiento, luego los resultados o su capacidad de realizar predicciones es evaluado con el conjunto de validación, en el caso de que las métricas de los resultados obtenidos sean satisfactorias se puede realizar una última prueba al modelo con el conjunto de prueba para ratificar resultados, en el caso de que al evaluar los resultados con el conjunto de validación y estos no sean satisfactorios se ajustan hiperparámetros del modelo y se inicia nuevamente el ciclo de entrenamiento hasta obtener un modelo que logre realizar las predicciones deseadas.

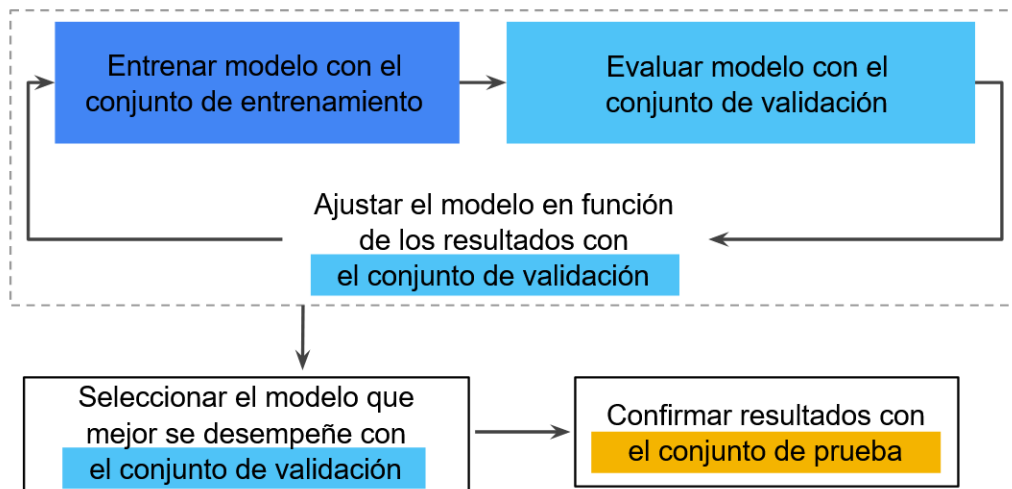


Figura 2: Proceso de entrenamiento para un modelo de aprendizaje supervisado [9].

Estos modelos utilizan una gran cantidad de datos, para poder trabajar con estos se tienen, por lo general, los siguientes siete pasos [8]:

1. Limpieza de datos.
2. Integración de datos.
3. Selección de datos.
4. Transformación de datos.
5. Minería de datos.
6. Evaluación de patrones.
7. Reporte.

La forma más simple de trabajar con machine learning es utilizar una Regresión lineal, la cual asume una linealidad entre los datos entregados [10], aunque también hay nuevos métodos, más complejos, como lo son los árboles de decisiones y las redes neuronales artificiales.

Las redes neuronales son utilizadas para tratar una gran cantidad de datos y formar relaciones más complejas que una regresión lineal, estas tienen la capacidad de reconocer patrones, estimar parámetros y realizar clasificaciones [9]. El funcionamiento a grandes rasgos es la utilización de distintas capas dentro de la red, la primera tiene los datos de entrada, considerados variables independientes, luego viene una capa oculta, la cual va

creando relaciones entre los datos, los resultados de esta capa pueden entrar a una capa de transformación no lineal la cual contiene una función de activación capaz de crear relaciones no lineales entre los datos y así llegando a obtener relaciones entre la entrada y la salida de la red neuronal como se puede detallar en la Figura 3.

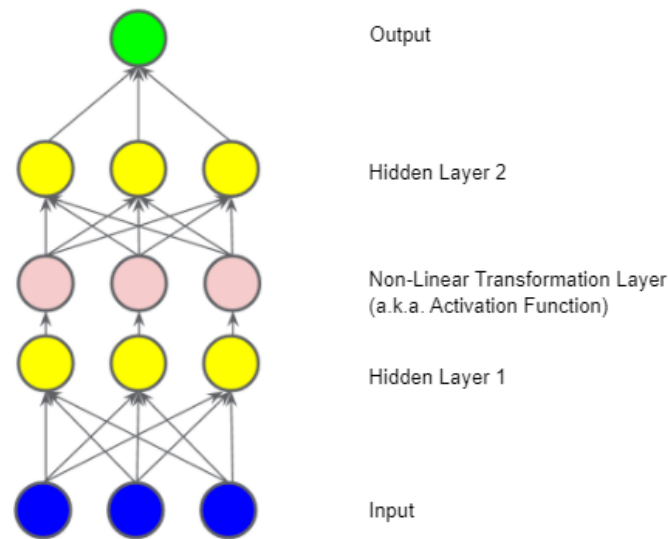


Figura 3: Esquema de red neuronal en machine learning [9].

Estas redes neuronales pueden llegar a tener un gran grado de complejidad para lograr detectar y segmentar objetos, para entender este concepto más en detalle se presenta la siguiente imagen.

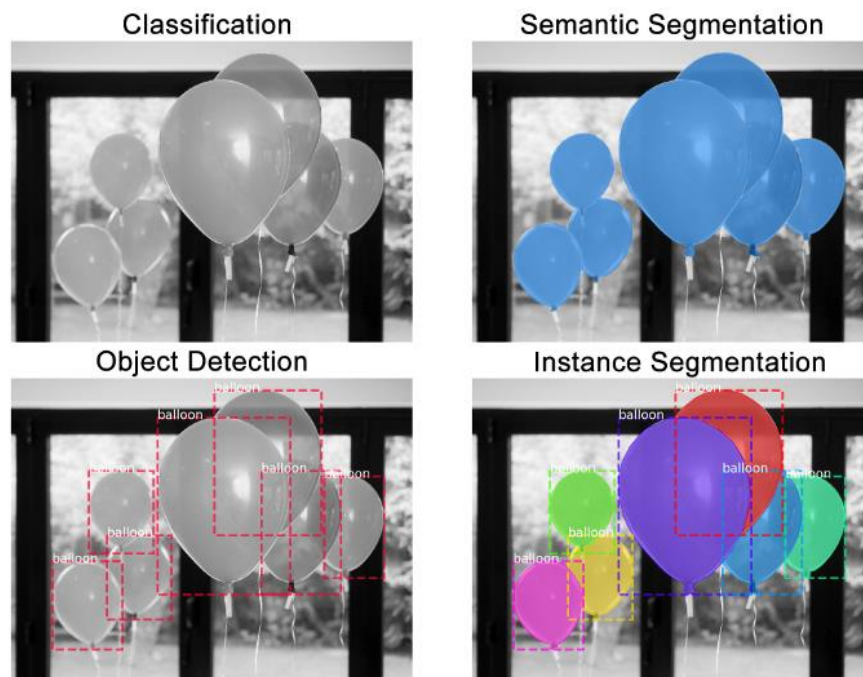


Figura 4: diferencias entre detección y segmentación.

De la Figura 4 es posible observar cuatro posibles objetivos en el análisis de imágenes.

- Clasificación: Existe un globo.
- Segmentación semántica: Estos son los píxeles pertenecientes a los globos.
- Detección de objetos: Hay 7 globos en esta imagen y estas son sus ubicaciones.
- Segmentación de objetos: Hay 7 globos en estas ubicaciones y estos son los píxeles de los globos.

Con estos conceptos claros en la
Tabla 1 se muestran distintos modelos basados en redes neuronales.

Tabla 1: Modelos basados en redes neuronales.

Modelo	Objetivo	Ventajas	Desventajas
Yolo v4	Detectar objetos	Mayor precisión en detección	Un poco más lento que Yolo v5
Yolo v5	Detectar objetos	Es la detección más rápida	Pierde precisión
Faster R-CNN	Detectar objetos	Pequeños tiempos en detección	No es capaz de segmentar objetos
Mask R-CNN	Detectar y segmentar objetos	El único modelo capaz de segmentar objetos	Es el modelo más lento

Uno de los modelos más utilizados es Faster R-CNN, este modelo tiene como objetivo entregar una etiqueta del objeto que encuentra en la imagen y un recuadro dentro del cual está contenido el objeto como se puede visualizar en la Figura 5, el funcionamiento de este modelo se explicara en la siguiente sección. En el trabajo de (Shuoquan Wei, 2019), se busca identificar puntos calientes en módulos PV analizando imágenes en el espectro infrarrojo tomadas desde un UAV (Vehículo aéreo no tripulado), para lograr esta tarea se utiliza el modelo Faster R-CNN, con el cual se logra la identificación de cada módulo dentro de una imagen, pero al no existir una segmentación el análisis no se puede realizar en una imagen única para cada módulo, si no que se debe realizar dentro de la imagen original, lo cual hace que el método sea menos efectivo en comparación con métodos más avanzados como lo es Mask R-CNN.

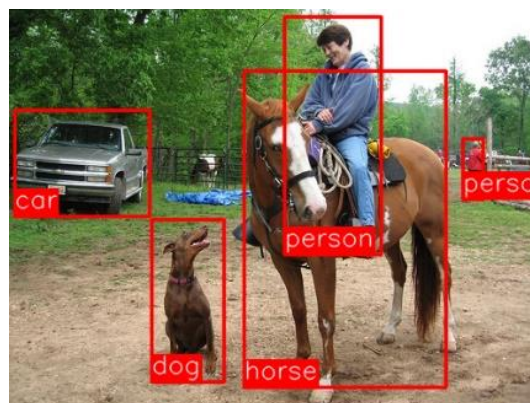


Figura 5: Resultado de Faster R-CNN [11].

Existen diversos trabajos en donde se busca detectar y segmentar módulos fotovoltaicos mediante modelos que utilizan Mask R-CNN, esta técnica es una mejora a Faster R-CNN consiguiendo además de una etiqueta y un recuadro delimitador, crear una máscara o segmentación del objeto que se está buscando dentro de la imagen como se puede ver en la Figura 6, esta técnica se puede considerar como la más avanzada dentro de la detección de objetos en imágenes y su funcionamiento será explicado con mayor detalle en la siguiente sección. En el trabajo de Díaz [12] se logra identificar y segmentar módulos PV en imágenes dentro del espectro infrarrojo con un “F₁ score” de 0.98 lo que habla de la robustez del modelo Mask R-CNN, Lukas Bommers [13] identifica y segmenta módulos PV a partir de videos infrarrojos realizados por un UAV (Vehículo aéreo no tripulado), cabe destacar que el video utilizado es separado en “Frames” para su detección y segmentación mediante el modelo Mask R-CNN y finalmente se analizan las fallas que pueda tener el panel fotovoltaico, además debido a las similitudes de este trabajo con el que se quiere realizar se tendrán en cuenta los requerimientos que deben cumplir los videos o fotografías para poder utilizar el modelo:

1. Cada fila de la planta PV es escaneada individualmente
2. La cámara se mueve monótonamente a lo largo de la fila, es decir, no hay movimientos hacia atrás significativos
3. La fila que se esté analizando se debe visualizar completamente y siempre ser la primera fila desde debajo de la imagen.
4. La fila debe quedar aproximadamente horizontal o vertical en cada imagen o “frame”.

Otro punto a tener en consideración es el trabajo realizado por Abdulla [14], el cual identifica globos en imágenes RGB utilizando una pequeña variación de Mask R-CNN, esta pequeña variación es reemplazar la sección de “ROI Align” por una función de TensorFlow llamada “crop and resize” la cual se aproxima de buena manera al resultado y es más simple que la planteada por los creadores de Mask R-CNN, este cambio puede ser tomado en cuenta para el trabajo que se pretende realizar, ya que las figuras que se van a detectar (Módulos

PV) tienen geometrías bastante simples y podría disminuir el costo computacional del modelo.

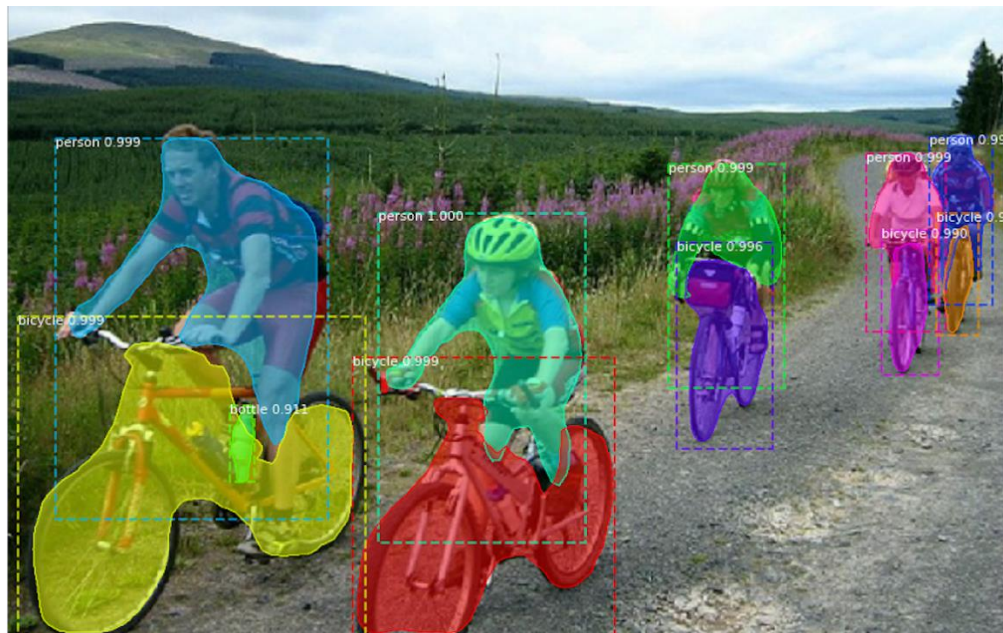


Figura 6: Resultado de la detección y segmentación utilizando Mask R-CNN.

3. Modelo de Mask R-CNN

El modelo de Mask R-CNN [15] o “Mask regional convolutional neural network” es una técnica de segmentación de objetos en imágenes mediante técnicas de machine learning, es capaz de reconocer objetos, crear un recuadro alrededor y además, una máscara en el área del objeto. Este modelo es una mejora de Faster R-CNN por lo que de forma introductoria se mostrará el funcionamiento de este último para luego entender Mask R-CNN.

3.1. Faster R-CNN

Uno de los mejores métodos de machine learning para identificar objetos dentro de una imagen es Faster R-CNN, el cual entrega como resultados una etiqueta del objeto que se desea encontrar dentro de una imagen junto con un recuadro alrededor de este, existen tres capas o “layers” principales dentro de esta familia de modelos (CNN) [16], estas capas son las siguientes:

- Capa convolucional o “convolutional layer”: Esta capa ayuda a extraer la imagen de entrada como un mapa de características o “feature map” mediante el uso de filtros y núcleos o “kernels”.
- Capa de agrupación o “pooling layer”: Esta capa ayuda a reducir la muestra de los mapas de características al resumir la presencia de características o “features” en parches del mapa de características.
- Capa totalmente conectada o “fully connected layer”: estas capas conectan cada neurona de una capa con cada neurona de otra capa.

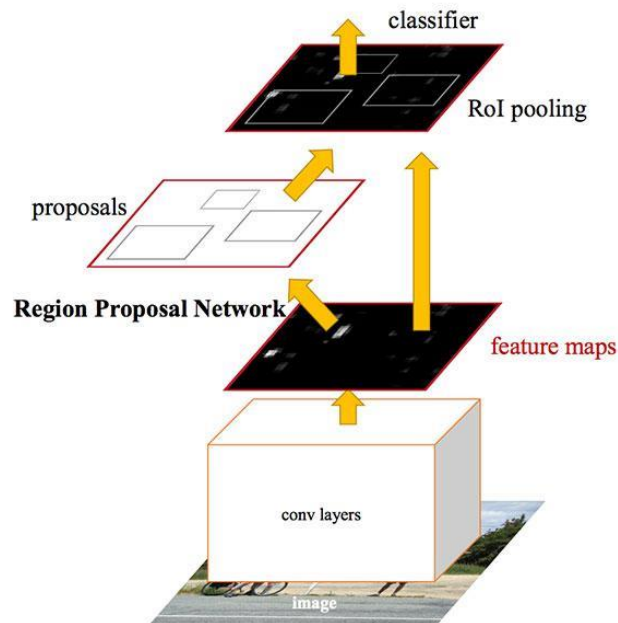


Figura 7: Funcionamiento de Faster R-CNN

El funcionamiento de Faster R-CNN se puede comprender con el mapa conceptual correspondiente a la Figura 7 en donde el primer paso corresponde a ingresar una imagen como por ejemplo la imagen correspondiente a la parte A) de la Figura 8 a una red neuronal convolucional (ResNet, VGG, Inception, etc), esta red entrega un mapa de características como el que se puede ver en la Figura 8 parte B), esta imagen es lo que la máquina entiende como características para lograr identificar un objeto.

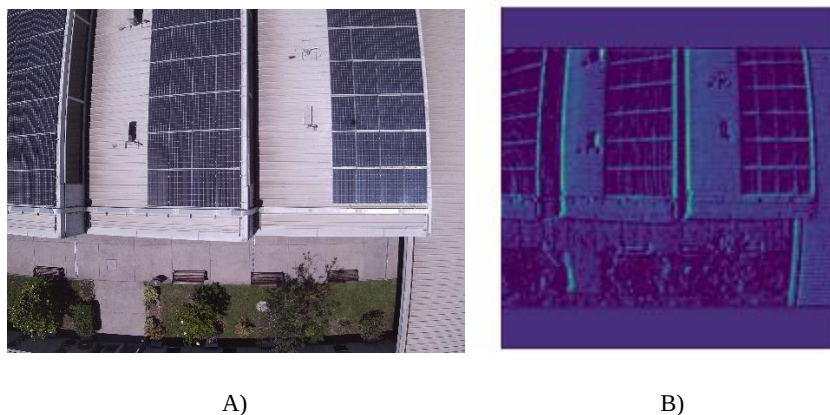


Figura 8: Ejemplo de imagen en conjunto de validación y su mapa de características.

El mapa de características obtenido es entregado a “Region Proposal Network” la cual entrega regiones de interés, estas corresponden a donde puede existir el objeto dentro de la

imagen, para este caso corresponden a lugares dentro de la imagen en donde se pueda encontrar un módulo fotovoltaico. Cada una de estas regiones tiene un tamaño diferente, “ROI pooling” (agrupación de Regiones de Interés) se encarga de equiparar los tamaños de todas las regiones de interés como se puede ver en la Figura 9, cabe destacar que en este ejemplo fue limitada la cantidad de regiones de interés a 50 para poder tener una correcta visualización. Toda esta información o regiones de interés entran a un clasificador compuesto por dos “Full Convolutional layers” o capas completamente convolucionales, finalmente de estas dos capas se obtiene la clasificación del objeto y el cuadro delimitador o “bounding-box”, como se puede ver en la Figura 10.

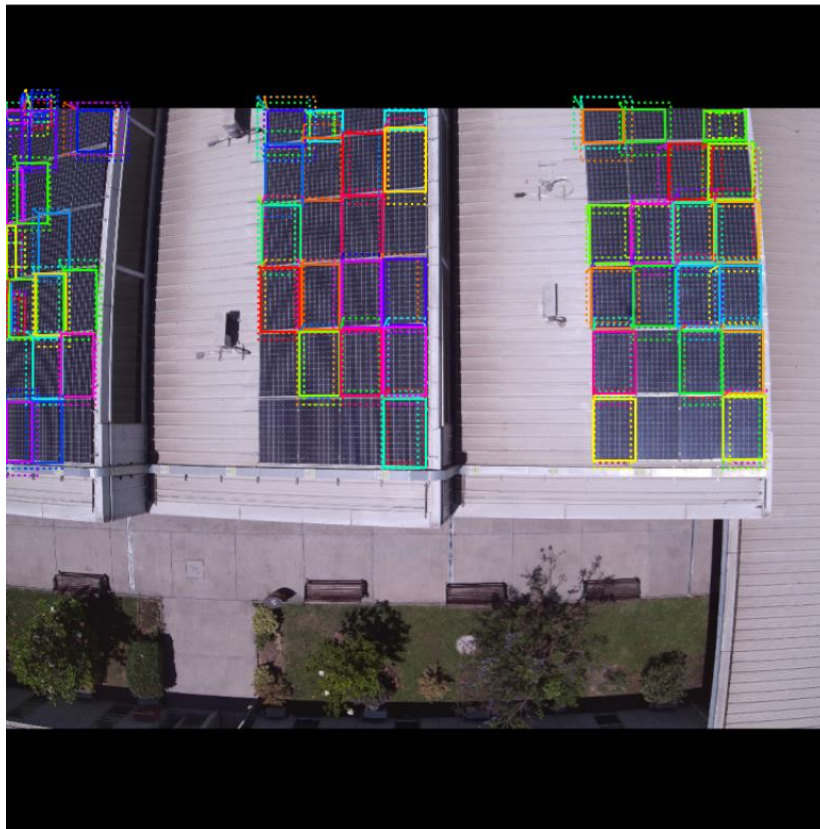


Figura 9: Regiones de interés para imagen de ejemplo.



Figura 10: Imagen resultante de Faster R-CNN.

3.2. Mask R-CNN

A partir de la herramienta Faster R-CNN en 2017 se publica Mask R-CNN [15], la técnica más efectiva actualmente para segmentar objetos en una imagen, ya que, es capaz de identificar un objeto, crear un cuadro delimitador y una máscara en el área del objeto. Su funcionamiento se muestra en el esquema de la Figura 11.

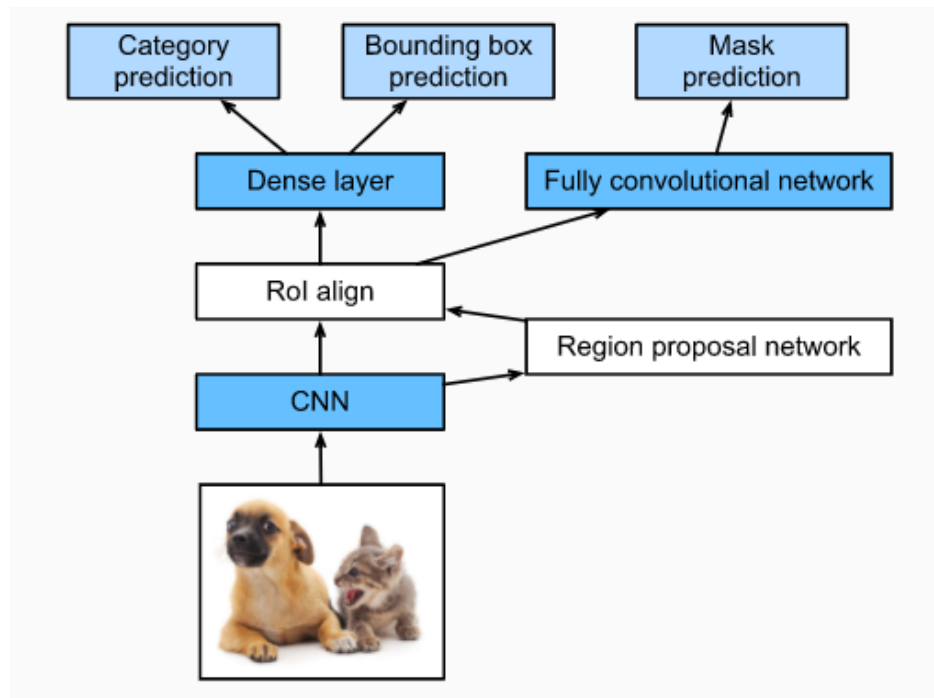


Figura 11: Mapa conceptual de una Mask R-CNN.

Como se mencionó anteriormente Mask R-CNN se basa en el funcionamiento de Faster R-CNN y realiza dos grandes cambios, el primero es cambiar las “ROI Pooling” por “ROI Align” (Alineación de Regiones de Interés), esto se realizó, ya que, según los autores de Mask R-CNN al realizar el cálculo de mapa de características con “ROI Pooling”, este mapa no se alineaba con las regiones de la imagen original, esto se debe al hecho de que las máscaras de píxeles deben ser más precisas que las máscaras del cuadro delimitador, por su parte “ROI Pooling” redondea los límites de la región de interés lo que conduce a una desalineación, en cambio, en “ROI Align” no existen estos redondeos y así se puede obtener una idea más exacta de lo que existe en cada píxel para poder crear la máscara del objeto que se busca segmentar en la imagen.

El segundo cambio importante es que se le agrega una salida al “ROI Align” como se puede ver en la Figura 11, esta información ingresa a dos capas convolucionales que tienen por objetivo entregar una “máscara” que corresponde a la segmentación del objeto como se

puede ver en la Figura 12, en donde se entrega una etiqueta, un cuadro delimitador y esta nueva máscara en el área del objeto que en este caso son paneles fotovoltaicos.

Las ventajas de Mask R-CNN son [16]:

- Simplicidad: Es simple entrenar Mask R-CNN
- Desempeño: Mask R-CNN supera a todos los modelos existentes en cada tarea.
- Flexibilidad: Mask R-CNN es fácil de generalizar a diferentes tareas.

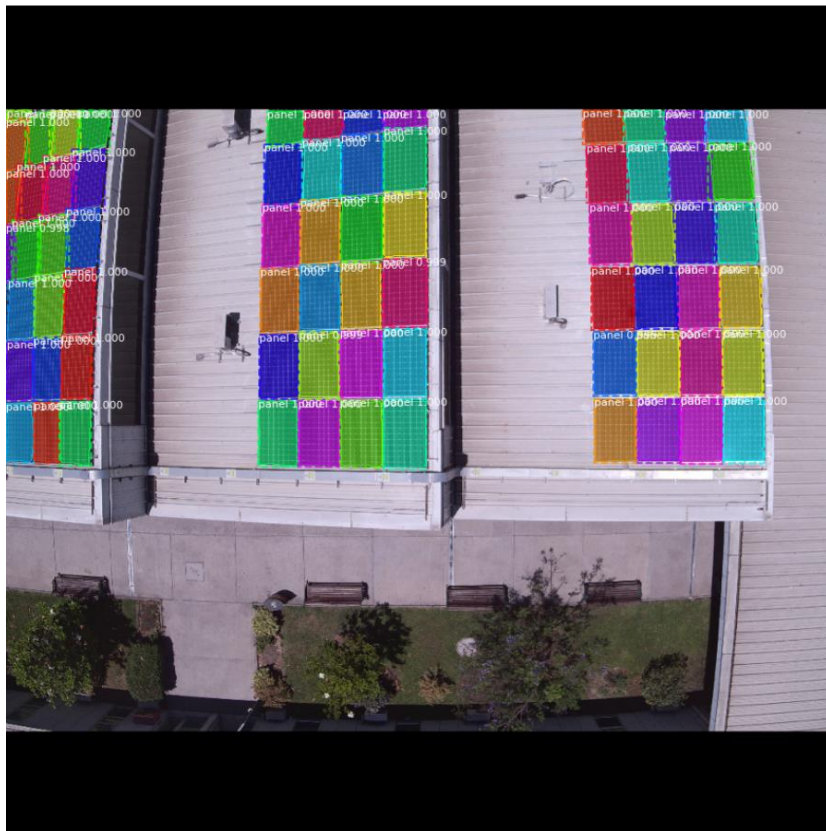


Figura 12: Resultado de Mask R-CNN.

3.3. Alteraciones al modelo original de Mask R-CNN

Este trabajo parte desde el modelo base desarrollado por Abdulla [14] en donde se realizan dos principales cambios al postulado original de Mask R-CNN [15], este modelo

originalmente utiliza como columna vertebral una red neuronal, comúnmente ResNet101 o ResNet50, pero los autores originales de Mask R-CNN crearon una mejora llamada Red Piramidal de características o “Feature Pyramid Network” la cual puede representar mejor los objetos en múltiples escalas. FPN mejora la pirámide de extracción de características estándar al agregar una segunda pirámide que toma las características desde el nivel más alto de la primera pirámide y las pasa a las capas inferiores de la segunda, al hacerlo permite que las características de todos los niveles tengan acceso a ambas.

La segunda alteración es el reemplazo de “ROI Align” por la función “crop and resize” de la librería Tensorflow de Python, este cambio se realiza más que nada por la simplicidad del modelo, ya que, al buscar segmentar objetos de geometría simple, paneles solares en el caso de este trabajo, la función de Tensorflow se aproxima lo suficiente a “ROI Align” con un costo computacional menor.

3.4. Configuraciones del modelo

El modelo obtenido fue entrenado con una configuración específica de hiperparámetros que se detallan a continuación.

- Tasa de aprendizaje o “Learning rate” de 0.001, este hiperparámetro es un escalar usado para entrenar un modelo a través de la técnica del descenso de gradiente. Durante cada iteración, el algoritmo de descenso de gradiente multiplica la tasa de aprendizaje por el gradiente.
- Tamaño de lote o “Batch size” de 1 imagen, es decir, la cantidad de ejemplos, en este caso imágenes, utilizados en una iteración en el entrenamiento del modelo.
- Épocas de entrenamiento o “epochs”, se utilizaron en este caso 30 épocas, las cuales se pueden entender como una fase de entrenamiento completa sobre todo el conjunto de datos de modo que cada ejemplo se haya visto una vez. Por lo tanto, una época representa $N/\text{iteraciones de entrenamiento de tamaño de lote}$, donde N es el número total de ejemplos.

- Pasos por época, en esta configuración se utilizaron 100 pasos por época, los cuales corresponden a una evaluación hacia adelante y hacia atrás de un lote.
- Número máximo de detecciones finales, este hiperparametro se estableció en 300, así el modelo es capaz de segmentar hasta 300 módulos fotovoltaicos dentro de una misma imagen.

Además, como columna vertebral del modelo o “Backbone” se utilizó la red neuronal ResNet101 la cual tiene por objetivo extraer desde la imagen de entrada un mapa de características. Dentro de esta red neuronal y a lo largo de todo el modelo es necesario incluir capas de transformación no lineal, las cuales dentro poseen una función de activación que tienen por objetivo crear una relación más compleja entre los datos de entrada, en este trabajo se utiliza la función de activación ReLU, su comportamiento se puede observar en la Figura 13. Para obtener mayor información sobre los hiperparámetros revisar [17], junto con lo anterior para lograr un mejor entendimiento de lo que son las funciones de activación y en particular la función ReLU consultar [18] y [19].

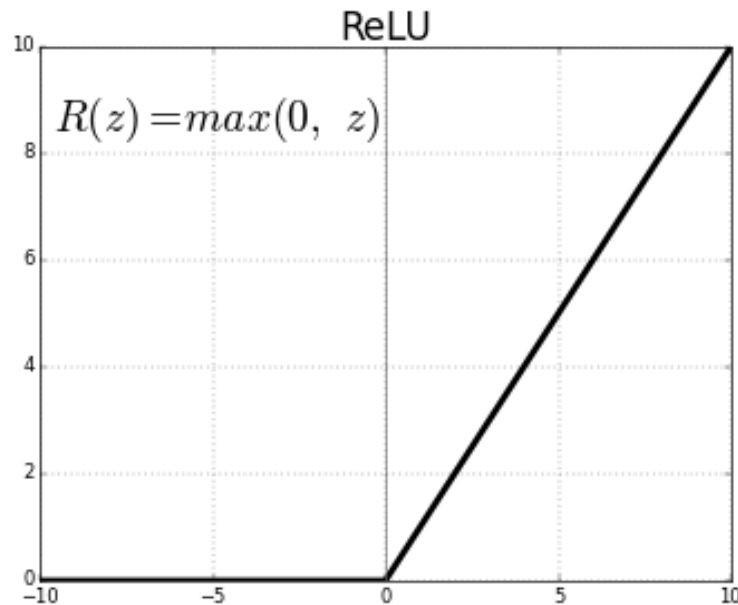


Figura 13: Comportamiento de función ReLU [19].

3.5. Especificaciones de la máquina

Uno de los puntos importantes para tener en cuenta es el tiempo que se demora en realizar el entrenamiento del modelo y el tiempo que se demora en aplicar el mismo. Uno de los factores más importantes a tener en cuenta es la utilización o no de una tarjeta gráfica para realizar todos los procesos, para que Tensorflow reconozca la tarjeta gráfica y sea utilizable es necesario instalar ciertos softwares específicos del modelo [20] como por ejemplo CUDA de NVIDIA, en Anexos 1 se detallan distintos softwares y librerías de Python instaladas junto con sus respectivas versiones.

Junto a la tarjeta gráfica existen distintos elementos que componen un computador y todos cumplen distintas funciones durante el entrenamiento del modelo, las especificaciones se detallan a continuación.

- Procesador AMD Ryzen 7 5800H.
- Tarjeta gráfica NVIDIA RTX 3060, 6GB de VRAM.
- Memoria RAM de 16 GB DDR4 3200MHz
- Sistema Operativo Windows 10, 64 bits.

Con estas especificaciones se realiza un entrenamiento en un total de 7 horas aproximadamente considerando las configuraciones detalladas anteriormente, si se buscara replicar estos resultados o realizar un nuevo entrenamiento con un conjunto de ejemplos distinto y se utilizara una configuración sin una tarjeta gráfica hay que tener en cuenta que el entrenamiento se realizará en alrededor de 18 días, es decir, 432 horas aproximadamente. Otro punto para tener en cuenta es la necesidad de utilizar una tarjeta gráfica con al menos 6GB de VRAM para poder realizar el entrenamiento, de caso contrario es bastante probable que durante este proceso el modelo entregue un error de “OOM” (out of memory), es decir, no será capaz de continuar el entrenamiento.

3.6. Requerimientos de entrada

Al realizar múltiples entrenamientos con el conjunto de ejemplos y seleccionando el que tuviese el mejor desempeño, se lograron identificar distintos requerimientos para lograr un uso óptimo del modelo desarrollado, esto no quiere decir que en el caso de no cumplirse estas indicaciones el modelo no sea capaz de segmentar módulos fotovoltaicos, si no que, es posible que dentro de una imagen no se reconozcan todos los módulos o las máscaras creadas no cubrirán toda el área efectiva del panel fotovoltaico. Los requerimientos establecidos son los siguientes.

1. La fila o “string” que se esté analizando se debe visualizar completamente, es decir, no contenga módulos cortados porque esto afectara a la entrega de resultados.
2. La fila o “string” debe quedar en una posición aproximadamente vertical u horizontal.
3. En una imagen es preferible que solo se visualice una única fila o “string” de módulos PV para lograr mantener un orden en la segmentación y no exista un sobreprocesamiento de data dentro de este modelo y en los modelos siguientes en trabajos futuros.
4. Se obtendrán mejores resultados a medida que la inclinación de los módulos se acerque a 0° , es decir, que el lente de la cámara esté perpendicular a los paneles fotovoltaicos.

Para que quede una mejor idea de lo que se está buscando como imagen ideal se muestra un ejemplo en la Figura 14, luego en secciones posteriores se mostrarán los resultados obtenidos a partir de esta imagen en particular.



Figura 14: Ejemplo de imagen ideal como input del modelo.

4. Conjunto de ejemplos o dataset

Como se mencionó en secciones anteriores, para poder realizar el ciclo de entrenamiento de un modelo basado en Mask R-CNN es necesario disponer de un conjunto de ejemplos. Para el presente trabajo se utilizaron 110 imágenes las cuales debieron ser segmentadas manualmente para entregarle al modelo información sobre en qué píxeles dentro de la imagen existe un panel fotovoltaico y cuantos de estos módulos PV existen dentro de esta.

4.1. Obtención de imágenes

El conjunto de 110 imágenes fue obtenido desde distintas plantas fotovoltaicas operativas en la Región Metropolitana de Chile, en específico, dos plantas instaladas en techo y una planta en terreno, junto con estas imágenes también existen algunos ejemplos aislados obtenidos desde diversas fuentes [21]. Las imágenes obtenidas de plantas fotovoltaicas fueron adquiridas utilizando los siguientes equipos.

4.1.1. Dron DJI Matrice 200

Este vehículo aéreo no tripulado tiene las siguientes características generales [22]:

- Dimensiones (desplegado): 887x880x378 [mm].
- Peso: 4.53 [kg] aproximadamente.
- Transmisión: OcuSync para un mejor alcance (2 a 5 [km]).
- Sensores: Delanteros, superior e inferior.
- Protección agua/polvo: IP43.
- Compatible con cámaras: Serie XT2, Z30, X5s, X7, PSDK.
- Máxima velocidad angular: Pitch 300 [°/s], Yaw 150 [°/s].



Figura 15: Imagen referencial de dron utilizado [22].

4.1.2. Cámara ZENMUSE XT2

Esta cámara presenta las siguientes características generales [23]:

- Dimensiones: Con lente de 25mm, 123.7x112.6x127.1 [mm].
- Rango controlable: Vertical $+30^\circ$ hasta -90° , horizontal $\pm 320^\circ$.
- Rango mecánico: Vertical $+45^\circ$ hasta -130° , horizontal $\pm 330^\circ$, giro -90° hasta $+60^\circ$.
- Velocidad máxima controlable: Vertical 90 [°/s], horizontal 90 [°/s].
- Resolución cámara térmica: 640x512 y 336x256 píxeles.
- Resolución cámara RGB: 3840x2160 y 1920x1080 píxeles.



Figura 16: Imagen referencial de cámara utilizada [24].

4.2. Conjunto de pre-entrenamiento

Un modelo de Mask R-CNN debería utilizar al menos miles de imágenes para realizar un correcto entrenamiento y poder segmentar un objeto dentro de una imagen lo que sería una gran demanda de ejemplos. Para facilitar los entrenamientos y poder reducirlos considerablemente existe un conjunto de pre-entrenamiento denominado COCO Dataset [25], el cual cumple con las siguientes características:

- Segmentación de objetos.
- Reconocimiento en contexto.
- Contiene alrededor de 330000 imágenes, en donde más de 200000 están etiquetadas.
- Hay 1500000 de objetos segmentados.
- Existen 80 categorías de objetos en el dataset.

Este conjunto de imágenes contiene máscaras de distintos objetos, para poder ser utilizadas en distintos entrenamientos, como se puede observar en la Figura 17 en dónde se logran identificar trenes, personas, bicicletas, automóviles, buses, textos y mochilas. De esta forma un modelo puede comenzar su entrenamiento a partir de un modelo pre-entrenado con el COCO Dataset y luego entrenar con el conjunto propio de imágenes ahorrando una gran cantidad de tiempo y esfuerzo en obtener un conjunto de ejemplos excesivamente grande, en este trabajo al utilizar este tipo de pre-entrenamiento se realiza un entrenamiento con 110 imágenes.



Figura 17: Imagen perteneciente a COCO Dataset [25].

4.3. Segmentación manual para imágenes de entrenamiento

Para poder realizar el entrenamiento del modelo es necesario tener imágenes en donde los paneles fotovoltaicos ya estén segmentados para que la máquina pueda aprender a identificarlos, para poder realizar esta tarea se pueden utilizar distintos programas, Lukas Bommes [13] usa un programa creado por los mismos colaboradores de la investigación, en cambio en el trabajo realizado por Abdulla [14] para segmentar manualmente sus imágenes utiliza el programa VGG Image Annotator (VIA) [26] el cual se puede utilizar directamente desde el navegador, cabe mencionar que existen otros programas que pueden satisfacer la necesidad de segmentar una imagen, tales como, LabelMe, el cual es un poco más lento que VIA, RectLabel, solo disponible para Mac, entre otros. Para el presente trabajo se opta por el software VGG Image Annotator por su simplicidad y la opción de guardar las anotaciones en el formato JSON [27] el cual es de fácil manipulación y tiene una gran compatibilidad para realizar el entrenamiento deseado.

4.4. Agrupación de ejemplos

Como se ha mencionado es necesario crear dos subconjuntos del conjunto de ejemplos original, un conjunto de entrenamiento y un conjunto de validación para poder realizar el entrenamiento deseado y su posterior validación analizando los resultados obtenidos.

- Conjunto de entrenamiento de 90 imágenes representando un 82% del total de ejemplos.
- Conjunto de validación de 20 imágenes representando un 18% del total de ejemplos.

Existe un fenómeno llamado “overfitting” o sobreajuste, el cual ocurre cuando se crea un modelo que se ajusta demasiado con el conjunto de entrenamiento causando que el modelo sea incapaz de hacer predicciones correctas sobre nuevos datos o imágenes en este caso. Para evitar esto y crear un modelo con una mayor robustez para poder realizar inferencias se realizaron cuatro entrenamientos del modelo con el mismo dataset pero ordenado de manera distinta de la siguiente manera.

- Primer orden de dataset: Este orden intenta que las imágenes dentro del conjunto de entrenamiento tengan “características” similares a las del conjunto de validación, por ejemplo, se intenta que en ambos conjuntos existan imágenes con objetos de geometrías similares o de color similar, este orden intenta respetar los lineamientos entregados en el curso de introducción a machine learning de Google [9].
- Segundo, tercer y cuarto orden: Para validar el primer orden de conjuntos se crean tres órdenes de forma aleatoria con la función ALEATORIO() de Excel [28], así en cada conjunto de entrenamiento y validación existen 90 y 20 imágenes respectivamente, elegidas de forma aleatoria.

5. Resultados y análisis

Luego de una serie de entrenamientos ajustando hiperparámetros del modelo este es capaz de segmentar módulos fotovoltaicos en ambientes complejos obteniendo resultados satisfactorios en el análisis de métricas que se presentaran en este capítulo, pero antes de ahondar en estas métricas es necesario establecer el formato en el cual serán entregados los resultados.

5.1. Formato de resultados

Existen tres distintos formatos en los que el modelo puede entregar resultados desde un mismo input, estos se detallan a continuación acompañados de un ejemplo utilizando como imagen de entrada la Figura 14.

- Segmentación original: Esta corresponde al utilizar el modelo de la forma más elemental, es decir, sin realizar ninguna variación. Al ejecutar el programa se entrega una imagen en donde existe una etiqueta del panel fotovoltaico, junto con una probabilidad de que sea un módulo PV, un recuadro alrededor del objeto y una máscara en su área como se puede ver en la Figura 18. Cabe destacar que para obtener este tipo de resultados es necesario ejecutar el programa contenido en el archivo “separar_paneles.py” y su contenido se encuentra en Anexos 2.



Figura 18: Resultado de segmentación original de Mask R-CNN.

- Segmentación y aplicación de filtro “Color splash”: Esta variación consiste en segmentar una imagen y luego aplicar un filtro, en donde el fondo de la imagen queda en escala de grises y solo se le entrega color al objeto que se está segmentando como se puede ver en la Figura 19, lo llamativo de esta función es que también es posible segmentar y aplicar filtros a videos, por lo que se pueden segmentar videos frame por frame. Para aplicar este tipo de segmentación es necesario introducir la siguiente línea en el CMD de Windows o en alguna ventana de comandos del sistema operativo que se esté utilizando, *python balloon.py --command=splash --weights="ubicación del archivo.h5 que contiene el modelo entrenado" --image="ubicación de la imagen o video que se busca segmentar"*.



Figura 19: Resultado de segmentación utilizando el comando "splash".

- Segmentación y separación de módulos: Esta aplicación se creó para poder enlazar los resultados de este modelo con un modelo siguiente el cual puede buscar pérdidas energéticas por soiling, cracking o sombras parciales. Este modelo necesita como datos de entrada una imagen en donde exista solo un módulo fotovoltaico, por lo que de la imagen ya segmentada como en la Figura 18 es necesario separar cada módulo en imágenes por separado, para mantener el orden en la entrega de resultados se optó por crear una carpeta con el formato imagen_AÑOMESDIATHORAMINUTOSEGUNDO y así evitar la posibilidad de crear duplicados y seguir un orden de segmentación, dentro de la carpeta existirán tantas imágenes como módulos fotovoltaicos encuentre el programa dentro de la imagen de entrada como se muestra en la Figura 20. Cabe destacar que para utilizar este método es necesario ejecutar el programa contenido en el archivo “separar_paneles.py”.

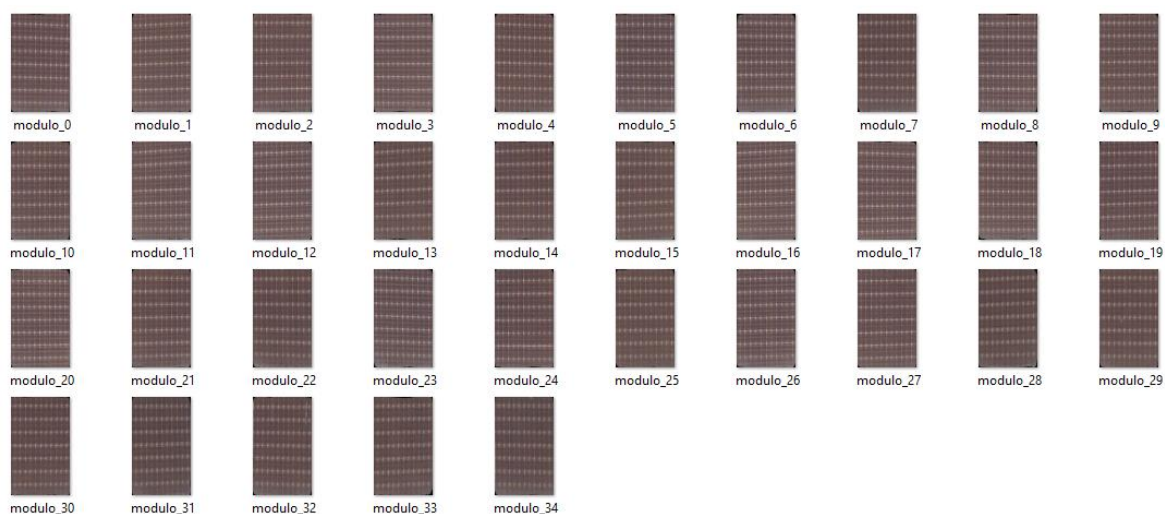


Figura 20: Resultado con módulos en imágenes separadas.

5.2. Métricas de desempeño

Se plantea usar tres métricas para medir el desempeño del modelo, la Precisión y la puntuación F1 [29] están basadas en la cantidad de paneles que logra segmentar el modelo dentro del conjunto de validación, en cambio la métrica “loss” mide la cantidad de área que no fue segmentada de los paneles.

Antes de realizar cálculos para la obtención de estas métricas es necesario primero introducir el término “matriz de confusión” donde uno de los ejes representa el valor real de la etiqueta, en este caso lo que realmente es un panel y el otro representa el valor predicho del modelo, en otras palabras, lo que el modelo aprendió y cree que es un panel, un esquema de esta matriz se puede observar en la Figura 21.

		Predicted condition	
Total population = P + N		Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

Figura 21: Matriz de confusión [29].

En la Figura 21 se puede observar que existen cuatro resultados posibles, en verde se encuentra el estado verdadero positivo (TP) en donde el modelo logró predecir de manera correcta la clase positiva, es decir, el modelo fue capaz de segmentar un objeto que realmente era un panel solar, también está el estado verdadero negativo (TN), en donde el modelo predijo la clase negativa de forma correcta, lo que no es un panel no fue marcado como panel. Los otros dos posibles estados se encuentran en rojo y corresponden a errores, el primero es el falso positivo (FP) en donde el modelo predice la clase positiva cuando el ejemplo era una clase negativa, segmenta un objeto como panel cuando este no lo es y finalmente se encuentra el estado falso negativo (FN) cuando el modelo predice la clase negativa siendo un ejemplo perteneciente a la clase positiva, entonces el modelo no segmenta un panel que realmente existe en la imagen.

Teniendo en cuenta los resultados expuestos en la matriz de confusión se puede calcular la precisión y puntuación F1 de la siguiente manera respectivamente.

$$Presición = \frac{TP}{TP+FP} \quad (1)$$

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP+FN)} \quad (2)$$

Para el cálculo de la tercera métrica denominada “loss” o pérdida se puede observar de mejor manera en el archivo “model.py”, pero en simples palabras es la resta del área entregada como ejemplo y el área predicha por parte del modelo.

Como se mencionó en la sección 4.4 se realizaron cuatro entrenamientos, uno para cada orden del conjunto de ejemplos, por lo que se entregaran las métricas obtenidas para cada orden y poder otorgarle una mayor robustez al modelo realizado.

5.2.1. Orden impuesto

Para el primer orden, el cual corresponde al que se intentó de mantener una similitud entre imágenes de ambos conjuntos, se obtuvo la siguiente matriz de confusión en función de la totalidad de módulos existentes dentro de las imágenes del conjunto de validación.

		Predicción	
		Panel	No Panel
Realidad	Panel	1262 (TP)	70 (FN)
	No Panel	5 (FP)	0 (TN)

Figura 22: Matriz de confusión utilizando el primer orden del conjunto de ejemplos.

Así utilizando la ecuación (1), (2) y los resultados entregados en la Figura 22 se obtiene una Precisión del 99.6% junto con una puntuación F1 de 0.97.

Además, durante el entrenamiento se obtiene el siguiente comportamiento de la métrica “loss” mostrado en la Figura 23, el modelo obtuvo el menor “loss” en la época 30 correspondiente a 15%.

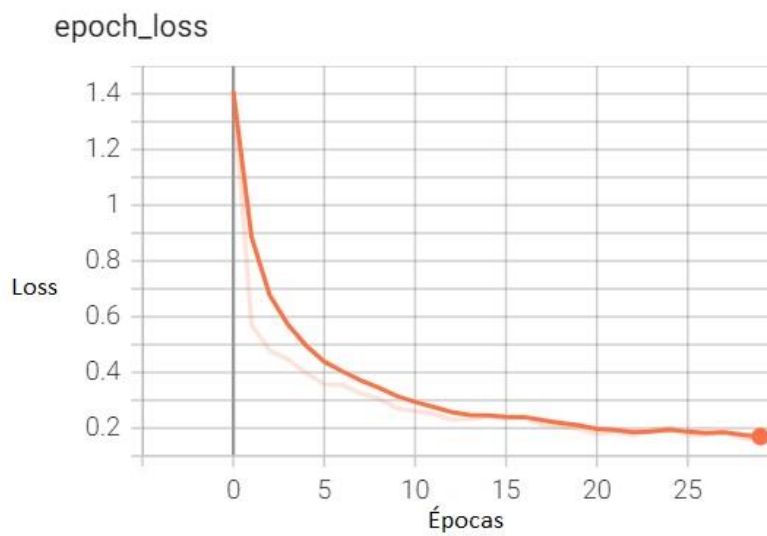


Figura 23: Gráfico de loss vs épocas para el primer orden.

5.2.2. Orden aleatorio

En los tres órdenes restantes se dividió el conjunto de ejemplos de forma aleatoria como se explicó anteriormente, así para el primero orden se obtiene la matriz de confusión mostrada en la Figura 24 y se muestra el comportamiento de la métrica “loss” a lo largo de las épocas en la Figura 25, obteniendo una Precisión de 99.8%, una puntuación F1 de 0.98 y el menor “loss” se alcanza en la época 27 correspondiente a 17%.

		Predicción	
		Panel	No Panel
Realidad	Panel	1186	56
	No Panel	3	0

Figura 24: Matriz de confusión utilizando el segundo orden del conjunto de ejemplos.

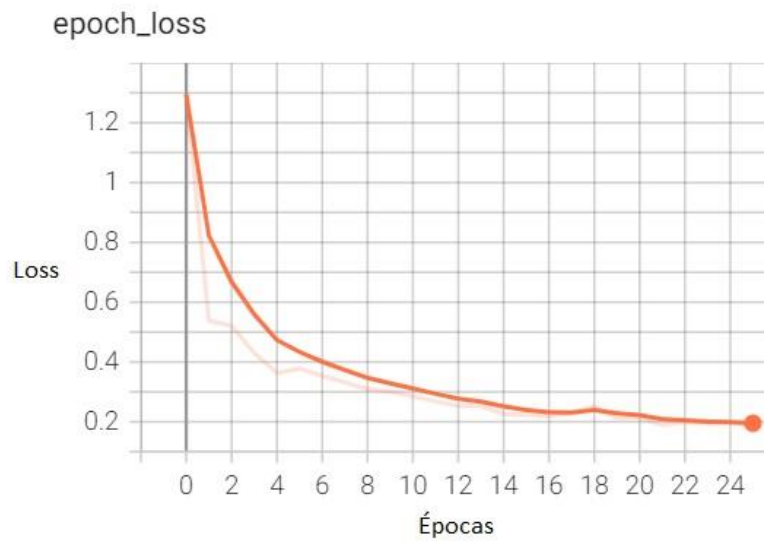


Figura 25: Gráfico de loss vs épocas para el segundo orden.

Para el tercer orden se obtiene la matriz de confusión que se muestra en la Figura 26 y el comportamiento de la métrica “loss” se muestra en la Figura 27, obteniendo así una Precisión de 100%, una puntuación F1 de 0.98 y el menor “loss” se alcanza en la época 30 correspondiente al 18%.

		Predicción	
		Panel	No Panel
Realidad	Panel	1283	46
	No Panel	0	0

Figura 26: Matriz de confusión utilizando el tercer orden del conjunto de ejemplos.

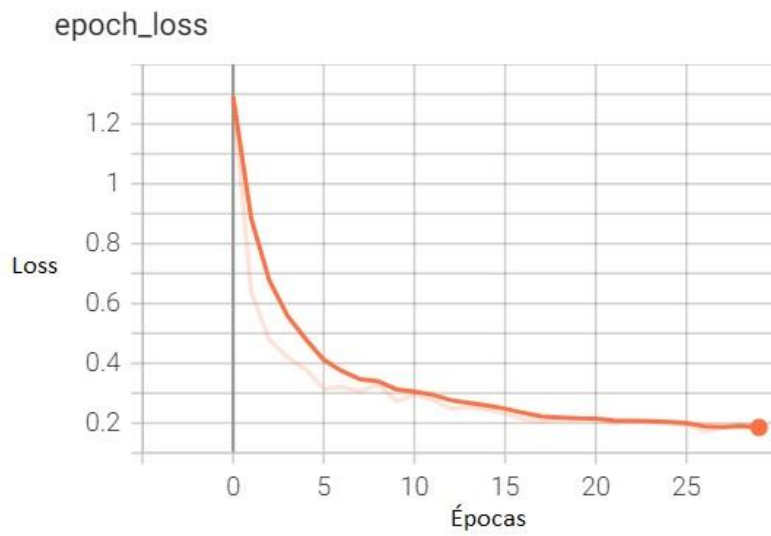


Figura 27: Gráfico de loss vs épocas para el tercer orden.

Para el cuarto orden se obtiene la matriz de confusión que se muestra en la Figura 28 y el comportamiento de la métrica “loss” se muestra en la Figura 29, obteniendo así una Precisión de 99.9%, una puntuación F1 de 0.99 y el menor “loss” se alcanza en la época 30 correspondiente al 18%.

		Predicción	
		Panel	No Panel
Realidad	Panel	1334	34
	No Panel	1	0

Figura 28: Matriz de confusión utilizando el cuarto orden del conjunto de ejemplos.

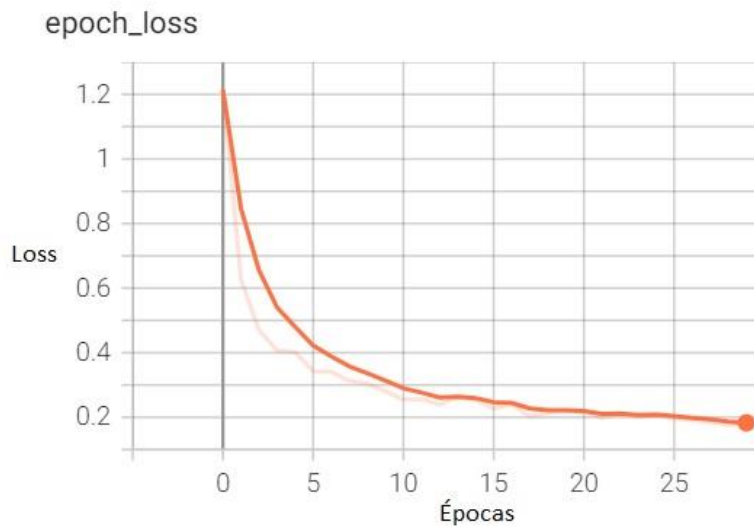


Figura 29: Gráfico de loss vs épocas para el cuarto orden.

5.3. Comparación

Con las tres métricas obtenidas para cada orden es posible formar la siguiente tabla resumen.

Tabla 2: Métricas obtenidas para cada orden.

Orden	Precisión	F1	Loss
N°1 Impuesto	99,60%	0,97	15%
N°2 Aleatorio	99,80%	0,98	17%
N°3 Aleatorio	100%	0,98	18%
N°4 Aleatorio	99,90%	0,99	18%

De la Tabla 2 se puede observar que el orden N°4 fue el que obtuvo una mayor puntuación F1 del 0.99, lo que implica que el modelo detectó prácticamente todos los módulos fotovoltaicos del conjunto de validación, pero obtuvo un porcentaje de “loss” del 18% siendo el mayor de todos. Al observar esta última métrica de “loss” es posible notar como los tres órdenes aleatorios se mantienen con una pérdida de área entre el 17 y 18%, en cambio para el orden impuesto baja al 15% por lo que el modelo entrenado con el primer orden es capaz de detectar mayor área de los paneles segmentados lo cual es un resultado bastante deseado.

Al observar el comportamiento de la métrica “loss” en los cuatro gráficos entregados se puede notar que todos presentan una tendencia similar demostrando la capacidad del modelo de converger a una solución a medida que avanzan las épocas de entrenamiento.

Un punto para tener en cuenta son qué imágenes del conjunto de ejemplos terminó en el conjunto de validación para cada orden, ya que estas poseen características distintas, por ejemplo, las imágenes pertenecientes a la planta fotovoltaica instalada en terreno presentan un ángulo de inclinación muy elevado por lo que no cumplen con los requerimientos de entrada presentados en la sección 3.6 por lo que es de esperarse que el modelo no sea capaz de tener un buen desempeño al detectar y segmentar paneles fotovoltaicos dentro de estas imágenes. Al realizar una inspección en los grupos de validación se observa que en el primer orden realizado es donde se encuentran más de estas imágenes por lo que podría explicar el menor desempeño dentro de las métricas Precisión y F1 en comparación con los órdenes aleatorios.

Tomando en cuenta lo anteriormente expuesto se recomienda utilizar el modelo entrenado con el primer orden, ya que sus métricas de Precisión y F1 se ven disminuidas por el tipo de imágenes que tenía el conjunto de validación, pero si se espera utilizar el modelo con los requerimientos impuestos en la sección 3.6 se espera una mejora sustancial en la detección y segmentación, la ganancia que se obtiene al utilizar este modelo por sobre los modelos obtenidos con los órdenes aleatorios es el 2% más de área segmentada en cada panel fotovoltaico segmentado.

Finalmente, se pueden comparar la puntuación F1 obtenida para el entrenamiento del modelo realizado con el primer orden del conjunto de ejemplos la cual fue de 0.97 con la puntuación obtenida para los trabajos de Vega Díaz [12] y Bommers [13] las cuales fueron 0.98 y 0.99 respectivamente, al observar los valores son lo suficientemente cercanos para poder demostrar junto con los demás argumentos expuestos la robustez que tiene el modelo para poder identificar y segmentar paneles fotovoltaicos en imágenes RGB con ambientes o fondos complejos.

6. Conclusiones

Se desarrolla un modelo que logra detectar y segmentar módulos fotovoltaicos en un arreglo o “string” mediante técnicas de machine learning. En base al estudio realizado se concluye que la técnica más óptima para el modelo es la utilización de Mask R-CNN, ya que otras técnicas de detección como Faster R-CNN o YOLO solo son capaces de detectar objetos entregando una etiqueta y un cuadro delimitador, en cambio Mask R-CNN es capaz de entregar una etiqueta, un cuadro delimitador y una máscara en el área del objeto, es por esto que se considera la técnica más avanzada en la detección de objetos en imágenes y en concreto esta máscara es el output necesario para luego poder extraer el panel de la imagen y así realizar un posterior análisis panel por panel en trabajos futuros.

Para que el modelo sea capaz de detectar y segmentar módulos PV fue necesario realizar un entrenamiento, por lo que se crea un conjunto de ejemplos, en este caso 110 imágenes obtenidas principalmente de dos plantas fotovoltaicas en techo y una en terreno dentro de la Región Metropolitana, este conjunto para ser catalogado como ejemplos tuvo que ser etiquetado con el programa VGG Image Annotator [26], es decir manualmente se segmentó lo que es un panel dentro de la imagen. Este conjunto de ejemplos contiene ambientes o fondos complejos, en otras palabras, contiene objetos similares en color o en forma que puedan confundir al modelo logrando una mayor robustez y una mejor capacidad de predicción por parte de este. Además, el conjunto de ejemplos se ordenó de cuatro maneras distintas para realizar cuatro distintos entrenamientos y así corroborar que no se esté sobreajustando el modelo al conjunto de entrenamiento, lo que hubiese provocado una baja capacidad del modelo para segmentar módulos PV en imágenes futuras.

Finalmente, ajustando hiperparámetros y configuraciones se logra entrenar un modelo capaz de predecir o segmentar hasta 300 paneles fotovoltaicos en imágenes con ambientes complejos con una precisión del 99.6%, una puntuación F1 de 0.97 y un parámetro de pérdida o “loss” del 15%, comparando estas métricas con estudios relacionados se concluye que el modelo entrenado logra los resultados esperados para la segmentación de módulos en arreglos fotovoltaicos.

7. Trabajos futuros

Se propone en primer lugar realizar un estudio sobre los límites del modelo, analizar las condiciones críticas en las imágenes como máxima altura, máxima velocidad del viento o velocidad relativa del dron al momento de tomar la imagen, máximo ángulo de inclinación, entre otros, para garantizar una buena detección y segmentación por parte del modelo.

Para continuar con el análisis de imágenes es necesario obtener un nuevo dataset de imágenes de módulos fotovoltaicos en operación junto con la cantidad de energía eléctrica generada por cada panel para poder realizar el entrenamiento del modelo de pérdida energética desarrollado en trabajos anteriores [6].

Finalmente, es posible utilizar este modelo de Mask R-CNN como ejemplo o realizar una adaptación en su código para desarrollar un modelo capaz de realizar una segmentación, detección y análisis de fallas en módulos PV dentro de imágenes infrarrojas, la adaptación en este caso es importante ya que las características de entrada para una imagen RGB son distintas que para una imagen dentro del espectro infrarrojo por lo que se debe realizar un estudio más detallado.

8. Bibliografía

- [1] V. Zapata *et al.*, «Climate change impacts on the energy system: a model comparison», *Environ. Res. Lett.*, vol. 17, n.º 3, p. 034036, feb. 2022, doi: 10.1088/1748-9326/ac5141.
- [2] Rothenberg, A, «How long do solar panel last. Wholesale Solar.», 2018.
- [3] M. Köntges, S. Kurtz, C. Packard, U. Jahn, K. A. Berger, y K. Kato, *Performance and reliability of photovoltaic systems: subtask 3.2: Review of failures of photovoltaic modules: IEA PVPS task 13: external final report IEA-PVPS*. Sankt Ursen: International Energy Agency, Photovoltaic Power Systems Programme, 2014.
- [4] R. R. Cordero *et al.*, «Effects of soiling on photovoltaic (PV) modules in the Atacama Desert», *Sci. Rep.*, vol. 8, n.º 1, p. 13943, dic. 2018, doi: 10.1038/s41598-018-32291-8.
- [5] M. Mussard y M. Amara, «Performance of solar photovoltaic modules under arid climatic conditions: A review», *Sol. Energy*, vol. 174, pp. 409-421, nov. 2018, doi: 10.1016/j.solener.2018.08.071.
- [6] Robinson Cavieres, Rodrigo Barraza, Danilo Estay, José Bilbao, y Patricio Valdivia, «Automatic soiling and partial shading assessment on PV modules through RGB images analysis», 2020.
- [7] F. Grimaccia, S. Leva, y A. Niccolai, «PV plant digital mapping for modules' defects detection by unmanned aerial vehicles», *IET Renew. Power Gener.*, vol. 11, n.º 10, pp. 1221-1228, ago. 2017, doi: 10.1049/iet-rpg.2016.1041.
- [8] M. Khoshlessan, B. Fahimi, y M. Kiani, «A comparison between Machine learning algorithms for the application of micro-grids Energy management», en *2020 IEEE International Conference on Industrial Technology (ICIT)*, feb. 2020, pp. 805-809. doi: 10.1109/ICIT45562.2020.9067203.
- [9] «Introduction to Machine Learning | Google Developers». <https://developers.google.com/machine-learning/crash-course/ml-intro> (accedido 2 de agosto de 2022).
- [10] M. Kayri, İ. Kayri, y M. Gencoglu, «The performance comparison of Multiple Linear Regression, Random Forest and Artificial Neural Network by using photovoltaic and atmospheric data», jun. 2017, pp. 1-4. doi: 10.1109/EMES.2017.7980368.
- [11] S. Bagalkar, «Deep Learning for Object Detection and Localization using R-CNN», *Alegion*, 23 de octubre de 2019. <https://medium.com/alegion/deep-learning-for-object-detection-and-localization-using-r-cnn-e88f85ea7c16> (accedido 2 de agosto de 2022).
- [12] J. J. Vega Díaz, M. Vlaminc, D. Lefkaditis, S. A. Orjuela Vargas, y H. Luong, «Solar Panel Detection within Complex Backgrounds Using Thermal Images Acquired by UAVs», *Sensors*, vol. 20, n.º 21, p. 6219, oct. 2020, doi: 10.3390/s20216219.
- [13] L. Bommers, T. Pickel, C. Buerhop-Lutz, J. Hauch, C. Brabec, y I. M. Peters, «Computer vision tool for detection, mapping, and fault classification of photovoltaics modules in aerial IR videos», *Prog. Photovolt. Res. Appl.*, vol. 29, n.º 12, pp. 1236-1251, dic. 2021, doi: 10.1002/pip.3448.
- [14] W. Abdulla, «Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow», *Medium*, 10 de diciembre de 2018. <https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46> (accedido 2 de agosto de 2022).

- [15] K. He, G. Gkioxari, P. Dollár, y R. Girshick, «Mask R-CNN». arXiv, 24 de enero de 2018. Accedido: 2 de agosto de 2022. [En línea]. Disponible en: <http://arxiv.org/abs/1703.06870>
- [16] E. Odemakinde, «Everything about Mask R-CNN: A Beginner's Guide», *viso.ai*, 19 de marzo de 2022. <https://viso.ai/deep-learning/mask-r-cnn/> (accedido 2 de agosto de 2022).
- [17] «Machine Learning Glossary», *Google Developers*. <https://developers.google.com/machine-learning/glossary> (accedido 3 de agosto de 2022).
- [18] C. Versloot, «Machine learning articles». 3 de agosto de 2022. Accedido: 3 de agosto de 2022. [En línea]. Disponible en: <https://github.com/christianversloot/machine-learning-articles/blob/3995782892d6f34b70c139265acdfa1c7b9ee07e/relu-sigmoid-and-tanh-todays-most-used-activation-functions.md>
- [19] «Redes Neuronales». https://ml4a.github.io/ml4a/es/neural_networks/ (accedido 3 de agosto de 2022).
- [20] «Compatibilidad con GPU», *TensorFlow*. <https://www.tensorflow.org/install/gpu?hl=es-419> (accedido 3 de agosto de 2022).
- [21] H. A. Photography, *Broadland District Council Eco Centre*. 2012. Accedido: 3 de agosto de 2022. [Photo]. Disponible en: <https://www.flickr.com/photos/hexcam/8006845746/>
- [22] «Matrice 200 Series - DJI», *DJI Official*. <https://www.dji.com/matrice-200-series> (accedido 3 de agosto de 2022).
- [23] «Zenmuse XT2 - DJI», *DJI Official*. <https://www.dji.com/zenmuse-xt2> (accedido 3 de agosto de 2022).
- [24] «Zenmuse Xt2 | ATyges ®», *ATyges*. <https://atyges.es/tienda/zenmuse-xt2/> (accedido 3 de agosto de 2022).
- [25] «COCO - Common Objects in Context». <https://cocodataset.org/#explore> (accedido 3 de agosto de 2022).
- [26] A. Dutta y A. Zisserman, «The VIA Annotation Software for Images, Audio and Video», en *Proceedings of the 27th ACM International Conference on Multimedia*, Nice France, oct. 2019, pp. 2276-2279. doi: 10.1145/3343031.3350535.
- [27] «JSON». <https://www.json.org/json-es.html> (accedido 3 de agosto de 2022).
- [28] «ALEATORIO (función ALEATORIO)». <https://support.microsoft.com/es-es/office/aleatorio-funci%C3%B3n-aleatorio-4cbfa695-8869-4788-8d90-021ea9f5be73> (accedido 3 de agosto de 2022).
- [29] «F-score», *Wikipedia*. 18 de julio de 2022. Accedido: 3 de agosto de 2022. [En línea]. Disponible en: <https://en.wikipedia.org/w/index.php?title=F-score&oldid=1099088638>
- [30] J. M. Malof, L. M. Collins, y K. Bradbury, «A deep convolutional neural network, with pre-training, for solar photovoltaic array detection in aerial imagery», en *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Fort Worth, TX, jul. 2017, pp. 874-877. doi: 10.1109/IGARSS.2017.8127092.
- [31] A. Moradi Sizkouhi, M. Aghaei, y S. M. Esmailifar, «A deep convolutional encoder-decoder architecture for autonomous fault detection of PV plants using multi-copters», *Sol. Energy*, vol. 223, pp. 217-228, jul. 2021, doi: 10.1016/j.solener.2021.05.029.

- [32] Valentín Romero, «Metodología Para Diagnóstico de Paneles Solares Fotovoltaicos a Través de Técnicas de Machine Learning», jul. 2022.
- [33] «How Mask R-CNN Works?», *ArcGIS API for Python*. <https://developers.arcgis.com/python/guide/how-maskrcnn-works/> (accedido 3 de agosto de 2022).
- [34] «Empiece a utilizar TensorBoard | TensorFlow». https://www.tensorflow.org/tensorboard/get_started?hl=es-419 (accedido 3 de agosto de 2022).

Anexos

Anexo 1: Softwares y librerías de Python

Softwares:

- CUDA 11.2.
- cuDNN 8.1.
- Visual C++ 2015 build tools.
- Anaconda con Python 3.6.

Librerías en Python:

- Tensorflow 2.6.0.
- Numpy 1.19.5.
- Scipy 1.5.4
- Pillow 8.4.0
- Cython 0.29.30
- Matplotlib 3.3.4
- Scikit-image 0.16.2
- Opencv-python 4.6.0.66
- H5py
- Imageio 0.4.0
- IPython[all] 7.16.1
- Os
- Time

Anexo 2: Código en lenguaje Python utilizado

```
# In[1]:
```

```
import os
```

```
import sys
```

```

import random
import math
import re
import time
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
print("Num GPUs Available: ", tf.config.list_physical_devices('GPU'))
# Root directory of the project
ROOT_DIR =
os.path.abspath("C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-master")
import cv2
import PIL
import datetime
# Import Mask RCNN
sys.path.append(ROOT_DIR) # To find local version of the library
from scripts import corners
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log
# from samples.balloon import balloon
from scripts import balloon
from mrcnn.config import Config
get_ipython().run_line_magic('matplotlib', 'inline')
import skimage
# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR,
"logs/balloon20220715T1345/mask_rcnn_balloon_0030.h5")

```

```

# Path to Balloon trained weights
# You can download this file from the Releases page
# https://github.com/matterport/Mask_RCNN/releases
BALLON_WEIGHTS_PATH =
"C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-
master/logs/balloon20220715T1345/mask_rcnn_balloon_0030.h5" # TODO: update this
path

# In[2]:
# ## Configurations
class BalloonConfig(Config):
    """Configuration for training on the toy dataset.
    Derives from the base Config class and overrides some values.
    """
    # Give the configuration a recognizable name
    NAME = "balloon"
    # We use a GPU with 12GB memory, which can fit two images.
    # Adjust down if you use a smaller GPU.
    IMAGES_PER_GPU = 1
    # Number of classes (including background)
    NUM_CLASSES = 1 + 1 # Background + balloon
    # Number of training steps per epoch
    STEPS_PER_EPOCH = 100
    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9
    # Maximum number of ground truth instances to use in one image
    MAX_GT_INSTANCES = 300
    # # Max number of final detections
    DETECTION_MAX_INSTANCES = 300
class InferenceConfig(BalloonConfig):
    # Set batch size to 1 since we'll be running inference on
    # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

```

```

# Maximum number of ground truth instances to use in one image
MAX_GT_INSTANCES = 300

# # Max number of final detections
DETECTION_MAX_INSTANCES = 300

config = InferenceConfig()
# config = balloon.BalloonConfig()
# config=Config
# BALLOON_DIR = os.path.join(ROOT_DIR, "datasets/balloon")
BALLOON_DIR =
os.path.join("C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-
master/datasets/paneles_2")

# # ## Notebook Preferences
# In[4]:

# Device to load the neural network on.
# Useful if you're training a model on the same
# machine, in which case use CPU and leave the
# GPU for training.
DEVICE = "/GPU:0" # /cpu:0 or /gpu:0
# DEVICE = "/job:localhost/replica:0/task:0/device:GPU:1"
# Inspect the model in training or inference modes
# values: 'inference' or 'training'
# TODO: code for 'training' test mode not ready yet
TEST_MODE = "inference"

# In[5]:
def get_ax(rows=1, cols=1, size=16):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Adjust the size attribute to control how big to render images
    """

```

```

_, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
return ax

### Load Validation Dataset

# In[6]:
# Load validation dataset
dataset = balloon.BalloonDataset()
dataset.load_balloon(BALLOON_DIR, "val")

# Must call before using the dataset
dataset.prepare()
dataset.class_names=['BG','panel']
print("Images: {} \nClasses: {}".format(len(dataset.image_ids), dataset.class_names))
### Load Model

# In[7]:
# Create model in inference mode
with tf.device(DEVICE):
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
                               config=config)
# model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
#                             config=config)

# In[8]:
# Set path to balloon weights file
# Download file from the Releases page and set its path
# https://github.com/matterport/Mask_RCNN/releases
# weights_path =
"C:/Users/claude/OneDrive/Escritorio/usm/memoria3.6/Segmentacion_Robinson/logs/balloon
20220518T2213/mask_rcnn_balloon_0008.h5"
weights_path = "C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-
master/logs/balloon20220715T1345/mask_rcnn_balloon_0030.h5"
# Or, load the last model you trained
# weights_path = model.find_last()

```

```

# Load weights
print("Loading weights ", weights_path)
model.load_weights(weights_path, by_name=True)
# ## Run Detection

# In[9]:
#image_id = random.choice(dataset.image_ids)
#image, image_meta, gt_class_id, gt_bbox, gt_mask =\
    #modellib.load_image_gt(dataset, config, image_id, use_mini_mask=False)
#info = dataset.image_info[image_id]
path = "C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-master/datasets/paneles_2/val/DJI_0403.jpg"
#filename = random.choice(os.listdir(path))
#path = os.path.join(path, filename)
# image = cv2.imread(path,1)
image = skimage.io.imread(path)
#image = cv2.imread(path)
#image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Run object detection
results = model.detect([image], verbose=0)
# Display results
ax = get_ax(1)
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                           ['BG','panel'], r['scores'], ax=ax,
                           title="Predictions")

# In[10]:
mask=r['masks']
mask=mask.astype(int)
os.makedirs("C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-master/resultados/imagen_{:%Y%m%dT%H%M%S}".format(datetime.datetime.now()))
donde_guardar="C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-master/resultados/imagen_{:%Y%m%dT%H%M%S}".format(datetime.datetime.now())

```

```

for i in range(mask.shape[2]):
    imagen=image.copy()
    puntos=[]

    for j in range(imagen.shape[2]):
        imagen[:,j]=imagen[:,j]*mask[:,i]

    aux = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    x1,y1,x2,y2,x3,y3,x4,y4=corners.puntos(aux)
    cv2.imshow("Image1", imagen)
    pts1 = np.float32([[y1, x1], [y2, x2], [y3, x3], [y4, x4]])
    pts2 = np.float32([[0, 0], [500, 0], [0, 300], [500, 300]])
    # pts2 = np.float32([[0, 0], [300, 0], [0, 200], [300, 200]])
    print(pts1,pts2)
    matrix = cv2.getPerspectiveTransform(pts1, pts2)
    result = cv2.warpPerspective(imagen, matrix, (500, 300))
    # result = cv2.warpPerspective(imagen, matrix, (300, 200))

    mirrored=cv2.rotate(result,cv2.ROTATE_90_CLOCKWISE)
    rotated = cv2.flip(mirrored, 1)
    # print(rotated)
    file_name = "modulo_{}.png".format(i)
    # skimage.io.imsave(file_name, rotated)
    #
    donde_guardar="C:/Users/claude/OneDrive/Escritorio/usm/memoria_limpia/Mask_RCNN-master/resultados"
    cv2.imshow("Image", imagen)
    #cv2.imwrite('/home/robinson/Documentos/mascaras/'+str(i)+'mascara.jpg',imagen)
    cv2.imshow("Perspective transformation", rotated)
    cv2.imwrite(os.path.join(donde_guardar,file_name),rotated)

    # cv2.waitKey(0)
    # cv2.destroyAllWindows()

```