UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Ingeniería en Software

DESARROLLO DE UN MODELO DE RED NEURONAL PROFUNDA PARA DETECCIÓN DE METEORITOS DEL SISTEMA INTEGRADO ALLSKYCAMS EN ESTADOS UNIDOS.

Trabajo de grado previo a la obtención del título de Ingeniero de Software

Autor:

Dennis Andrés Chicaiza Acosta

Director:

PhD. Iván García

Ibarra – Ecuador 2022

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	100308932-1		
APELLIDOS Y NOMBRES:	CHICAIZA ACOSTA DENNIS ANDRÉS		
DIRECCIÓN:	CARANQUI, AV. ESPINOSA DE LOS MONTEROS Y PADRE PIO DE PIETRELCINA 1-97		
EMAIL:	dachicaizaa@utn.edu.ec saitoinosaka@gmail.com		
TELÉFONO FIJO:	-	**TELÉFONO MÓVIL:**	0984910710

DATOS DE LA OBRA	
TÍTULO:	"DESARROLLO DE UN MODELO DE RED NEURONAL PROFUNDA PARA DETECCIÓN DE METEORITOS DEL SISTEMA INTEGRADO ALLSKYCAMS EN ESTADOS UNIDOS"
AUTOR:	CHICAIZA ACOSTA DENNIS ANDRÉS
FECHA: DD/MM/AAAA	05/09/2022
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERIA EN SOFTWARE
ASESOR /DIRECTOR:	PHD. IVÁN GARCÍA

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 5 días del mes de septiembre de 2022

EL AUTOR:

Nombre: Dennis Andrés Chicaiza Acosta
Cédula: 100308932-1

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE GRADO

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DEL ASESOR

Certifico que la Tesis previa a la obtención del título de Ingeniera en Software con el tema: "DESARROLLO DE UN MODELO DE RED NEURONAL PROFUNDA PARA DETECCIÓN DE METEORITOS DEL SISTEMA INTEGRADO ALLSKYCAMS EN ESTADOS UNIDOS" ha sido desarrollada y terminada en su totalidad por el Sr. Dennis Andrés Chicaiza Acosta, con cédula de identidad Nro. 100308932-1 bajo mi supervisión para lo cual firmo en constancia.

1002292603 IVAN DANILO GARCIA SANTILLAN

Firmado digitalmente por 1002292603 IVAN DANILO GARCIA SANTILLAN
Fecha: 2022.09.05 17:21:10 -05'00'

_____

Ph. D. Iván García

DIRECTOR DE TESIS

DEDICATORIA

El presente trabajo está dedicado a mi madre Liliana Acosta, a mi padre Fredy Chicaiza y a mi hermano Dylan Chicaiza. Su apoyo en este último empujón fue decisivo.

Dennis Chicaiza

AGRADECIMIENTOS

A mis progenitores Liliana Acosta y Fredy Chicaiza por haber sido mi soporte económico. Por haberme criado en un ambiente familiar, acogedor y con buenos principios. Es por ello por lo que este trabajo de grado se realizó sobre pilares muy fuertes emocionalmente hablando.

A mi hermano Dylan Chicaiza que con su vivaz existencia me enseñó que en la vida hay que sonreír a cualquier circunstancia y no tomarse las cosas tan en serio. Implícitamente, él me enseñó muchísimo y por eso lo aprecio y amo.

Agradecer a mi tía Cecilia del Pilar Lily por su humildad y aprecio. Gracias a su apoyo incondicional, pude crear mi propio destino a partir de una oportunidad inusual y muy valiosa.

A mi amigo Ángel Carmona, que conocí gracias a mi tía y que siempre contará conmigo para lo que sea. Lo aprecio mucho. Me hizo confiar de nuevo en la humanidad y allá donde yo vaya transmitiré sus valores.

A mi amigo Mike Hankey, que congeniamos desde el primer momento y teniendo metas afines haremos grandes cosas. Gracias por darme la oportunidad de realizar semejante proyecto.
A la música, que, siendo mi mayor pasión e inspiración en la vida, hizo llevadero todo este proceso.

A mis amigos y compañeros de carrera que discutimos temas muy interesantes llevándonos a grandes e importantes conclusiones tanto profesional como personales.
Por último, agradecerme a mí mismo. Por mi valía y constancia a la hora de tomar decisiones arriesgadas. Siempre priorizando lo que es mejor para mismo. Por haber sido considerablemente abierto a los consejos y comentarios de todas las personas que han pasado por mi vida.

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Resumen

AllSkyCam es una compañía internacional de productos de hardware y software en el área de la astronomía. Su principal objetivo es el estudio de meteoros y meteoritos. En 2016 empiezan a desarrollar lo que ahora se comercializa como AllSkyCam6 y desde entonces, se ha ido depurando y mejorando las diferentes funcionalidades. Debido al almacenamiento masivo de imágenes y clips de video por cada estación alrededor del mundo, la principal problemática que atraviesa AllSkyCam, es el almacenamiento masivo de falsos meteoros que perjudican el estudio de los datos relevantes. Para solventar esta problemática, se propuso un modelo de red neuronal convolucional sobre la API de detección de objetos de Tensorflow 2, utilizando el proceso Knowledge Discovery in Databases (KDD) como metodología. El modelo, denominado *AllSkyNet*, logra una detección de 99,36% para el problema de clasificación y un 92,70% para el problema de regresión.

Palabras clave: Visión por computador, Detección de objetos, Redes neuronales convolucionales, aprendizaje profundo, Python, Tensorflow.

Abstract

AllSkyCam is an international company of hardware and software products in astronomy. Its main objective is the study of meteors and meteorites. In 2016 they began to develop what is now marketed as AllSkyCam6 and since then, it has been debugging and improving the different functionalities. Due to the mass storage of images and video clips by each operator stations around the world, the main problem that passes through AllSkyCam is the mass storage of fake meteors that damage the study of the relevant data. To solve this problem, a convolutional neural network model was proposed on the Tensorflow 2 Object Detection API, using the Knowledge Discovery in Databases (KDD) process as methodology. The model, called *AllSkyNet*, achieves detection of 99.36% for the classification problem and 92.70% for the regression problem.

Keywords: Computer vision, Object detection, Convolutional Neural Network, Deep Learning, Python, Tensorflow.

Introducción

Antecedentes

Mike Hankey es un astrónomo amateur, dueño de la empresa https://www.mikehankey.com y https://allskycams.com a demás de director de operaciones de la American Meteor Society (AMS). Desde 2009 su sueño por construir un sistema que involucre software y hardware para el estudio tanto profesional como amateur de astros y estrellas lo ha llevado, a partir de primitivas tecnologías y herramientas ir poco a poco logrando su meta. Pretende que el estudio del cielo nocturno esté al alcance de todos sin importar sus conocimientos o su situación económica. Dentro de una red retroalimentada por los mismos usuarios, está desarrollando la plataforma de la American Meteor Society y su visión es cubrir los Estados Unidos de América con cámaras AllSkyCams.

Situación actual

El sistema integrado AllSkyCams es utilizado para estudios astronómicos tanto amateur como profesional. El actual sistema de reconocimiento de meteoros genera imágenes inconsistentes que se traducen en datos basura perjudicando el estudio de los meteoros. Esto se debe a que el sistema sólo hace uso de librerías de visión por computador que reconocen movimiento y no logran clasificar o identificar la diferencia de un meteorito con otros objetos como un avión o un ave.

Planteamiento del problema

Mediante la matriz Vester se ha identificado que AllSkyCams se enfrenta a un problema que desencadena en graves efectos para su modelo de negocio. Ya que a partir del algoritmo actual solo es posible el reconocimiento de movimiento en el cielo. El potencial que se obtendría con especialistas en el área de la inteligencia artificial, y, sobre todo, en desarrollo de modelos de redes neuronales profundas seria inmensurable. No obstante, los resultados arrojados por el actual sistema de reconocimiento son en gran parte contenido que no quieren ser analizados ya sean aviones, aves e incluso la luna y las nubes. Es por ello por lo que se debe limpiar estos datos manualmente, revisando cada clip de video o fotografía.

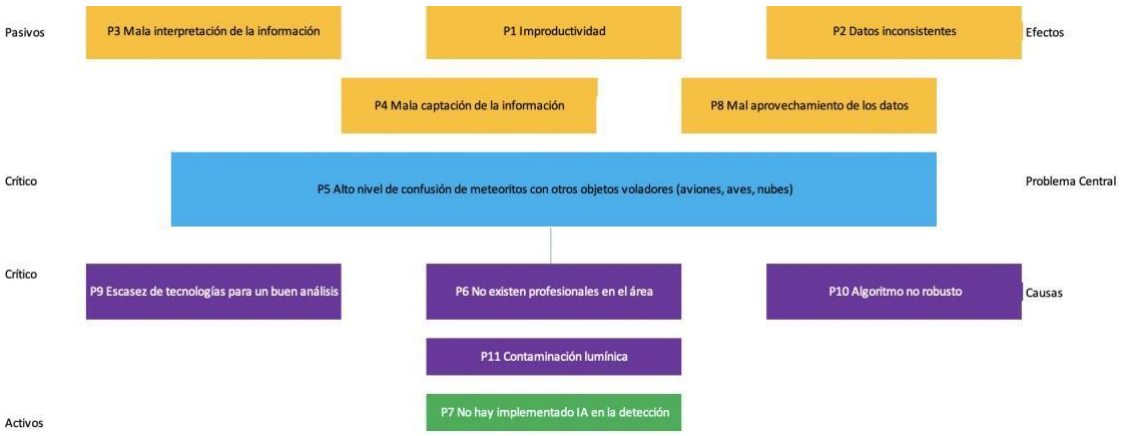


Fig. 1. Árbol de problemas.

Fuente: Propia

Objetivos

Objetivo General

Desarrollar un modelo de red neuronal profunda para detección de meteoritos del sistema integrado AllSkyCams en Estados Unidos.

Objetivos Específicos

- Elaborar un marco teórico concerniente a la detección de meteoritos basado en visión artificial y redes neuronales profundas con el lenguaje Python.
- Desarrollar una red neuronal profunda para detección de meteoritos utilizando Python y el framework Tensorflow.
- Validar cuantitativamente a través de métricas y gráficas el modelo de red neuronal profunda propuesta.

Alcance

El alcance del presente trabajo de grado es llegar a identificar la mejor técnica para desarrollar el modelo de red neuronal y cuantificar el porcentaje de precisión a la hora de detectar meteoritos. Dependiendo de la calidad del dataset proporcionado por la base de datos de la American Meteor Society (AMS). La profundidad del trabajo de grado llegará hasta la capacidad de determinar cuándo un objeto en movimiento corresponde a la clase meteorito o no. Si un meteorito es detectado, el resultado debe ser *meteorite_true* y el porcentaje de acierto de este resultado. Si por el contrario no existe nada en la imagen o se detecta otro

tipo de objeto que no comparte las mismas características que un meteorito, el resultado es *meteorite_false* y el porcentaje de acierto de este resultado. El modelo entrenado será validado en un sistema integrado AllSkyCams en tiempo real y proporcionará en tiempo real las predicciones.

Los datos históricos con los que se van a contar para construir el dataset provienen de sistemas integrados AllSkyCams situados en Monkton, Maryland y empieza desde diciembre de 2018 hasta finales de enero de 2021 en formato jpg proveídos por Mike Hankey.

A continuación, se desarrollará la arquitectura óptima de red neuronal convolucional para entrenamiento con imágenes en dos dimensiones mediante las siguientes librerías de Python:

- OpenCV: ayudará en el pre procesamiento, redimensión de las imágenes de entrada. OpenCV es el referente en librerías de código abierto para visión artificial, procesamiento de imágenes y redes neuronales. Recientemente ha incluido la funcionalidad de procesamiento por GPU para mejorar el rendimiento de aplicaciones de alto coste computacional. El desarrollador original del proyecto es Intel (Pérez, 2019).
- Numpy: ayudará en la transformación de los datos de imágenes a arrays multidimensionales que luego Tensorflow transformará en su propio formato de tensores (Pypi.org, 2022).
- Os: este módulo proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo. Será de gran ayuda para crear los directorios y manejar de manera versátil las rutas del modelo, anotaciones, archivo de configuración (Pypi.org, 2022).
- Tensorflow: mediante su API de detección de objetos de Tensorflow 2, permitirá el entrenamiento de una red neuronal convolucional (Tensorflow.org, 2022).
- TensorBoard: esta herramienta permitirá desplegar las gráficas de entrenamiento y validación (Tensorflow.org, 2022).
- Protobuf: es un tipo de archivo serializado binariamente con el protocolo Buffers que permite la compresión de las grandes magnitudes de datos que se trabajan en el área del big data y precisamente en el Deep learning. Es una tecnología de Google (Pypi.org, 2022).
- CUDA: es un entorno de desarrollo de software basado en una plataforma de código abierto y una interfaz de programación para crear aplicaciones basadas en diferentes tipos de unidades de procesamiento de gráficos (GPU) (Tensorflow.org, 2022).

- cudNN: es un driver que permite el uso de CUDA con Tensorflow usando las GPUs compatibles (Tensorflow.org, 2022).

Obteniendo un modelo de red neuronal profunda, se guardará como archivo en formato .pb conteniendo matrices multidimensionales para proceder a las predicciones. Cuando el entrenamiento haya finalizado y se haya guardado el modelo, se generará las gráficas de entrenamiento (accuracy/loss).

Se desarrollará según el diagrama de la metodología propuesta (KDD). Donde primero se debe etiquetar cada meteorito en las imágenes para obtener las coordenadas del bounding box en archivos xml. Todo esto gracias a Colabeler un software gratuito de etiquetado y posterior corrección de cualquier coordenada errónea en Roboflow un software gratuito online.

Con el Data Warehouse listo se debe hacer un procesamiento de este. Para ello se ha utilizado una herramienta cloud llamada Roboflow con la que el preprocesamiento es sencillo y versátil.

Para transformar los datos de entrada en información que la red neuronal pueda entender y aprender, se hizo uso de Roboflow igualmente ya que permite exportar los archivos necesarios denominados TFRecord para proceder con el entrenamiento.

A partir de un modelo pre entrenado en este caso MobileNet en su versión 2, se entrenará la red neuronal para que genere los pesos, checkpoints y el modelo en sí que luego se usará para las predicciones.

Los datos de validación serán específicamente el 20% del total de datos que se tenga disponible. Se evaluará la precisión del modelo y se generará la matriz de confusión que permitirá́ analizar la calidad y arquitectura del modelo. A continuación, se cargará el modelo en el sistema integrado AllSkyCams, ayudándose del bounding box generado por el sistema actual. Se analizará los frames que recolecta el sistema AllSkyCams y se realiza la predicción de los datos de entrada. La clasificación dentro del alcance de este proyecto será́ de una clase, por lo que solamente reconocerá si un objeto es un meteorito. La precisión a la cual se apunta en porcentaje es de un 85% como mínimo. En la imagen a analizar y predecir, podrán existir uno o varios objetos, ya que los bounding box tratarán a cada objeto por separado. El límite de múltiples análisis depende de la capacidad del hardware en el que se esté ejecutando el modelo.

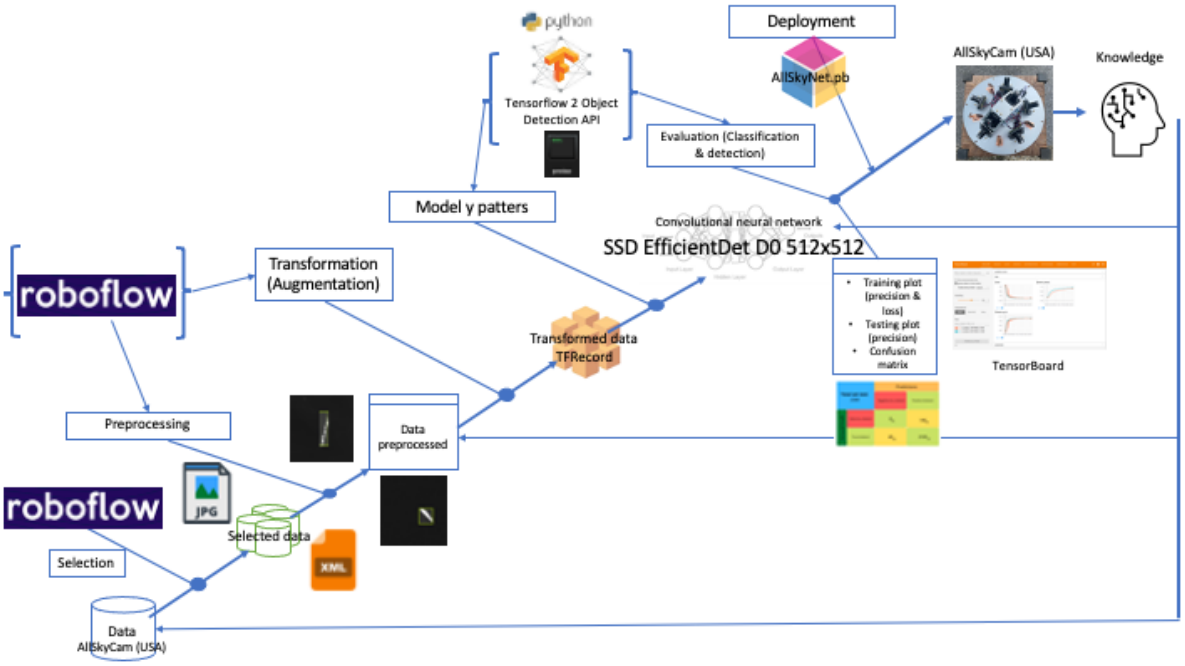


Fig. 2. Proceso Knowledge Discovery in Databases (KDD).

Fuente: Propia

Justificación

El presente proyecto se plantea con justificación tecnológica y como solución a un problema en un sistema hardware - software que se comercializa principalmente en los Estados Unidos de América y Europa. Se pretende mejorar los servicios proporcionados a sus clientes que usualmente son escuelas, institutos, universidades, centros de investigación, astrónomos profesionales y astrónomos amateurs.

La viabilidad de este proyecto es alta ya que se tiene contacto directo y comprometido con el propietario de la compañía. Así mismo, de los datos y herramientas que se requieren para cumplir con éxito la propuesta.

La implantación de este software en el sistema integrado AllSkyCams es importante porque permitirá mejorar el análisis de meteoros tanto para astrofotografía como para estudios de cuerpos celestes. El presente proyecto se apoya en el Objetivo de Desarrollo Sostenible número 9 relacionado con la Industria, Innovación e Infraestructura ya que la visión de la empresa AllSkyCams, es lograr que los artefactos tecnológicos sean accesibles «económicamente hablando» para la mayoría de las personas en el mundo, dándoles la oportunidad de cultivar mejores conocimientos sobre astronomía en las escuelas, institutos, universidades e incluso en hogares, sin limitarse solamente a centros de investigación.

El presente proyecto se encuentra dentro del área de la carrera; desarrollo, aplicación de software y cyber security (seguridad cibernética). En la sublínea de la inteligencia artificial, aprendizaje profundo, aplicándose al área de la astronomía y contribuyendo al área científica nacional e internacional.

La presente investigación será aplicada. Esto debido a que se utilizarán conocimientos en el área de visión artificial para la creación de un sistema que permita realizar análisis de meteoros.

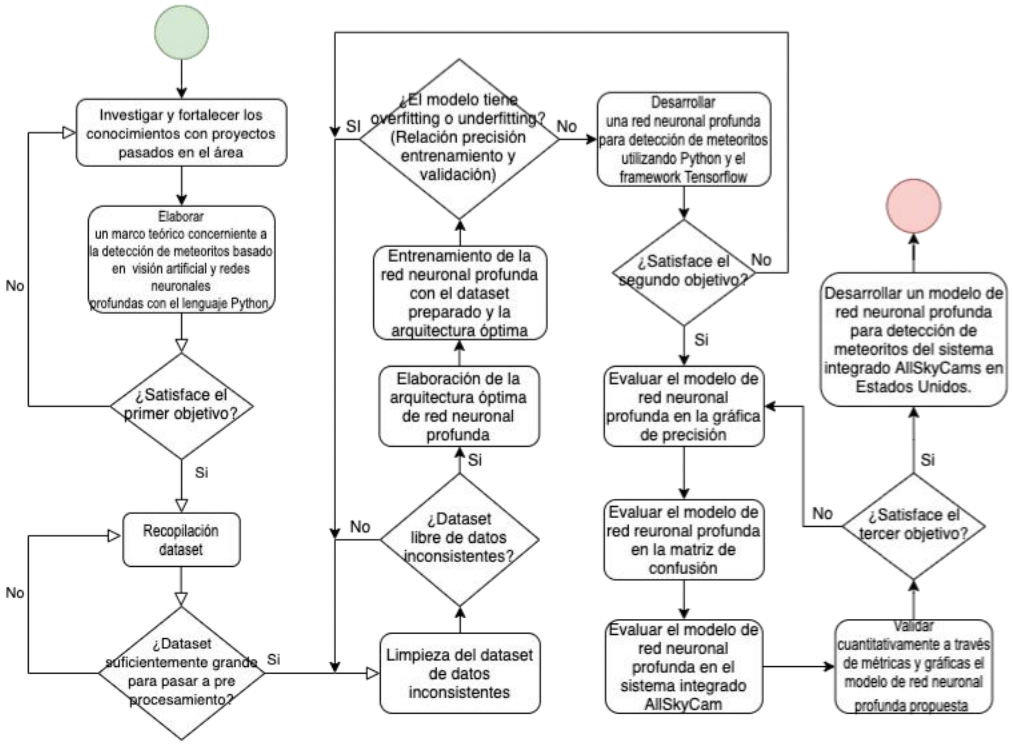


Fig. 3. Proceso metodología.

Fuente: Propia

Actualmente la investigación astronómica es amplia pero el seguimiento de meteoritos es un campo que aún está empezando a ser investigado. De todas formas, se expone a continuación (en orden local, regional, nacional e internacional) trabajos relacionados que, aunque traten tópicos similares, no exponen la solución planteada en la presente tesis.

CAPÍTULO I

Marco Teórico

En el siguiente capítulo se exponen todas las investigaciones relevantes para poder sustentar la teoría y práctica del presente trabajo de grado.

En los últimos años, se han recopilado cantidades masivas de datos en varios campos, incluida la seguridad cibernética, la informática médica, las redes sociales e incluso el campo científico astronómico. Esto ha provocado la búsqueda de nuevas formas de analizar estas grandes cantidades de datos para extraer información relevante. En este capítulo, se tratará con modelos de Deep Learning pre entrenados y sus respectivas arquitecturas. A demás de tener un acercamiento al proceso que se toma como metodología KDD, los hiperparámetros y trabajos pasados que han resuelto problemas similares.

(Liu et al., 2017), (Wang et al., 2021), (Smithson et al., 2016), entre otros; concuerdan que los métodos de detección de objetos basados en aprendizaje profundo son capaces de extraer características de alto nivel en múltiples escenarios y niveles del objeto a detectar, proporcionando una increíble mejora en cuanto a precisión y generalización de la detección frente a métodos anteriores. Las características de alto nivel se consiguen cuando las capas de la red son cada vez más profundas. En cierta capa, podría aprender a reconocer diferentes tipos de bordes en los patrones de imagen. En tanto que, en la siguiente capa, podría aprender una característica más compleja del objeto en la imagen. A esto se le denomina característica de alto nivel.

Las redes neuronales convolucionales suelen utilizarse para tareas difíciles relacionadas con el reconocimiento de patrones en imágenes. Su arquitectura usualmente es simple pero precisa, favoreciendo al rápido análisis por sus simplificadas operaciones (O'Shea & Nash, 2015).

1.1 Meteoros y AllSkyCams

Durante los últimos años, la astronomía, como muchos otros campos, se ha enfrentado al problema del procesamiento automático de grandes cantidades de datos. Esto ha sido soportado principalmente por instrumentos, que escanean grandes áreas del cielo con el objetivo de catalogar todo lo que encuentran (Smithson et al., 2016). Se han diseñado

diferentes sistemas para escanear grandes áreas del cielo, como el Sloan Digital Sky Survey, el Telescopio de Levantamiento Panorámico y el Sistema de Respuesta Rápida (Pan-STARRS), el Dark Energy Survey (DES) (Smithson et al., 2016) y recientemente el AllSky6 (Hankey et al., 2020). Para (Carrasco-Davis et al., 2019), la astronomía se enfrenta al desafío de flujos de datos cada vez más grandes producidos por grandes telescopios de prospección. Siendo la prospección es la exploración de la existencia de algo en este caso, eventos astronómicos.

Es de gran importancia detectar los eventos astronómicos lo más rápido posible. De este modo, poder investigar los fenómenos en detalle, evitando información basura, y sin haber transcurrido prolongados periodos de tiempo. Históricamente, los astrónomos han utilizado el lente ojo de pez para detectar eventos astronómicos (Smithson et al., 2016). Esto debido a que el lente ojo de pez, es capaz de abarcar más área en el cielo. Este tipo de lente será usado en las 7 cámaras dispuestas en el AllSky7 (modelo beta con el que se trabajará en el presente proyecto) para el reconocimiento de meteoros.

Vale mencionar que mientras se hacía estudio del estado del arte de la presente investigación, se pudo reconocer la existencia de terminología que explica la diferencia entre astro, meteoroide, meteoro y meteorito. Aunque todos conforman un mismo objeto, es la circunstancia la que da un nombre u otro.

Un astro es un cuerpo rocoso que orbita el sol. A lo que un meteoroide es un pedazo de un astro que fue desprendido por la colisión entre cuerpos celestes en el espacio. El meteoro se produce cuando un meteoroide se acerca demasiado a la atmosfera de la Tierra produciendo una combustión entre los elementos del meteoroide y la atmosfera. En este proceso se crea un rayo luminoso comúnmente denominado “estrella fugaz”. Cuando parte de este meteoroide no combustiona en la atmosfera y cae en la tierra es cuando se le denomina meteorito y los científicos pueden estudiar sus propiedades y por ende las propiedades de nuestro sistema solar (Vanderbilt Museum, 2020).

La American Meteor Society es una organización científica sin fines de lucro fundada en 1911 con el propósito de motivar y apoyar las actividades de investigación de astrónomos aficionados y profesionales (Hankey et al., 2020).

El producto actual se llama AllSky6 y está disponible comercialmente en el sitio web www.allskycams.com. Aunque en el presente trabajo se hará uso de la versión más actual, el AllSky7 que aún no se encuentra disponible al público.

Los sistemas AllSky6 por (Hankey et al., 2020), han estado funcionando en una capacidad beta durante más de 2 años y se han sometido a múltiples ciclos de mejora que han dado como resultado un sistema de hardware y software maduro y estable. Satisfaciendo los requerimientos más inmediatos de sus clientes, además de tener una interfaz amigable.

Durante un período de varios años, la American Meteor Society, Ltd. (AMS) ha desarrollado un sistema de hardware y software especializado para capturar, reducir, resolver y almacenar permanentemente datos de meteoros y meteoritos. Gracias a la alta precisión del software actual, se puede obtener coordenadas de trayectoria de los meteoros y su caída aproximada. Para (Hankey et al., 2020), la parte de la trayectoria de meteoros es el proceso más difícil de automatizar y verificar con un mínimo de intervención humana. La industria científica se ha visto muy interesada en usar los beneficios que trae las redes neuronales profundas sobre todo en el campo de visión artificial (R. Zhao et al., 2020) y AllSkyCams, no ha sido la excepción.

El software de detección y clasificación de (Hankey et al., 2020) se ejecuta en cada uno de los clips de vídeo en los que se detectó movimiento para rastrear y clasificar los objetos del clip. Los eventos similares a meteoros se marcan y guardan. Como parte de este proceso, los fotogramas de video HD y SD correspondientes al evento se extraen de los archivos de video sin procesar, se recortan y se guardan en la cola de revisión. Es con estos datos almacenados temporalmente con los que se trabajará.

Tanto las imágenes como los clips en HD se almacenan por 24 horas si fueron capturados en el día. En el caso de ser imágenes y clips capturados en la noche, son almacenados por 48 horas. Para las imágenes y clip en SD, el tiempo es mayor almacenándose por 7 días a los capturados en el día y por 30 días a los capturados en la noche. Luego, los meteoros capturados serán almacenados para siempre y los falsos positivos, son movidos a un directorio donde permanecerán por 30 días más. Después se eliminarán para siempre.

Cada objeto en movimiento que se encuentra en el video se clasifica según varias características del objeto. Dos de los componentes más importantes para la detección de meteoros son la velocidad angular y la intensidad de los píxeles. Sin embargo, existen 12 o más pruebas o filtros adicionales que se aplican a los objetos para determinar si son meteoros o no. Algunos de estos incluyen, la duración del evento, el cambio de intensidad a lo largo del tiempo, el número de espacios o cuadros faltantes, si el objeto se mueve en línea recta, la dirección de viaje constante y la distribución de puntos. También hay filtros negativos que,

cuando se aplican, anularán los meteoros. Por ejemplo, se programó un módulo específicamente para identificar los faros de los automóviles, ya que a veces se marcaban como meteoros, así también se han programado filtros para detectar y clasificar aves (Hankey et al., 2020).

Actualmente, la detección y clasificación no es 100% precisa, pero la interfaz de usuario permite eliminar fácilmente las falsas capturas o falsos positivos en términos de IA (Hankey et al., 2020). Aunque lo que se pretende, es hacer uso de estos datos. En vez de eliminarlos, usar el mismo proceso para clasificar estos resultados y mejorar la precisión de la red neuronal futura. Por tantos inconvenientes y factores que impiden una detección y clasificación precisa, a finales de 2021, Dennis Chicaiza propone a Mike Hankey, propietario de https://allskycams.com/ apostar por métodos de inteligencia artificial para solventar la principal problemática de recolectar y guardar falsos positivos como aves, aviones, helicópteros, nubes o faros y poder centrarse en los datos que realmente enriquecen al servicio y producto ofrecido por AllSkyCam6 y el futuro AllSkyCam7.

El sistema AllSkyCams es un sistema integrado para el estudio visual de eventos astronómicos en el cielo. Su hardware incorpora cámaras de alta resolución que cubren los 360 grados incluidos el zenit. Estas cámaras se conectan con un minicomputador que posee 8 GB de RAM, 64 GB de disco sólido (SSD) y un disco duro (HDD) de 1 TB, suficiente para las tareas de grabación, detección y clasificación de objetos por movimientos. Se tiene como preferencia el sistema operativo Linux en su distribución Ubuntu versión 21. Este minicomputador se conecta localmente a la red de las 7 cámaras y a la red de AllSkyCams global que conecta a todas las estaciones en el mundo.

La parte del software corresponde a software desarrollado por Mike Hankey y Vincent Perlerin. Principalmente desarrollado en Python, JavaScript, HTML y C. Su código fuente se encuentra almacenado en el siguiente repositorio: https://github.com/mikehankey/amscams

Algunos de los clientes y asociados principales de https://allskycams.com/ son:

- http://montanalearning.org/
- https://www.imo.net/
- https://www.amsmeteors.org/

Una tarea constante en astronomía es la capacidad de encontrar fuentes astronómicas de calidad. Esto es importante porque constituye la base sobre la que se construye un catálogo de imágenes de meteoros y meteoritos dentro de la comunidad de astrónomos aficionados y

profesionales de la AMS o cualquier otra organización astronómica. Un fenómeno interesante dentro de las empresas es que no siempre están interesadas en la técnica más precisa para solventar un problema, sino en la más apropiada para obtener los resultados esperados. Por eso es que un equilibrio entre la precisión y el costo computacional es indispensable, favoreciendo al producto final (Affonso et al., 2017).

1.2 Visión Artificial

1.2.1 Visión artificial

La visión artificial es una ciencia en auge dentro de la Inteligencia Artificial que incorpora métodos para analizar y comprender datos bidimensionales (como imágenes o frames en videoclips) del entorno y que posteriormente generan información numérica lógica a través de un computador, proporcionando conocimiento al investigador.

La capacidad de aprender automáticamente las características de los datos de entrada sin procesar se está volviendo imprescindible para las aplicaciones de visión artificial y aprendizaje automático debido a la gran cantidad de datos generados diariamente (Jyothi et al., 2018).

Las técnicas de aprendizaje profundo y visión artificial han superado a muchos métodos convencionales en visión por computadora aumentado los intereses de la investigación debido a su capacidad para superar los inconvenientes de los algoritmos tradicionales que dependen de un gran número de características diseñadas por el programador (Ordóñez & Roggen, 2016). Inicialmente, en el área de la visión artificial, se utilizó la red neuronal convolucional profunda para la clasificación de imágenes. El propósito de la clasificación es agrupar videos o imágenes en categorías semánticas. En la visión por computadora y el aprendizaje automático, la clasificación es un problema desafiante debido a la variabilidad intraclase y la similitud entre clases (Jyothi et al., 2018).

Es de recalcar que, si bien existe una amplia variedad de métodos; hasta la fecha, no existe un modelo genérico en sistemas de visión artificial que pueda usarse para cualquier problema de detección y clasificación de imágenes (Manassés, 2018).

El aprendizaje profundo ha logrado un gran éxito en muchas áreas de investigación. En particular, se han logrado avances considerables en el campo de la visión por computadora (Bai et al., 2020). Reconocer objetos a escalas muy diferentes es un gran desafío en la visión por computadora (Lin et al., 2017).

1.2.2 Detección de objetos

Los métodos tradicionales de clasificación de objetos astronómicos se basan en el preprocesamiento de una secuencia de imágenes (calibración) seguida de extracción de características (medición) (Carrasco-Davis et al., 2019). Las redes neuronales convolucionales han surgido como el algoritmo maestro en visión por computadora en los últimos años (Chollet, 2017).

Para obtener una comprensión completa de la imagen, no solo se debe concentrar en clasificar diferentes imágenes, sino también procurar estimar con precisión las ubicaciones de los objetos contenidos en cada imagen (Z. Q. Zhao et al., 2019).

La detección de objetos genéricos tiene como objetivo localizar y clasificar los objetos existentes en cualquier imagen y etiquetarlos con cuadros delimitadores rectangulares (bounding box) para mostrar la precisión de la existencia. Los marcos de los métodos de detección de objetos genéricos se pueden clasificar principalmente en dos tipos. Uno sigue el canal de detección de objetos tradicional. El otro considera la detección de objetos como un problema de regresión y clasificación, adoptando un marco de trabajo (framework) unificado para lograr resultados finales (categorías y ubicaciones). Los métodos basados en regresión y clasificación incluyen principalmente MultiBox, AttentionNet, YOLO y SSD (Z. Q. Zhao et al., 2019).

El exponencial progreso en Deep Learning y las mejoras en las capacidades de los dispositivos, incluida la potencia informática, la capacidad de memoria, el consumo de energía, la resolución del sensor de imagen y la óptica, han mejorado el rendimiento y la rentabilidad de acelerar aún más la difusión de las aplicaciones basadas en la visión. En comparación con las técnicas tradicionales de visión por computador, el Deep Learning permite a los ingenieros de CV lograr una mayor precisión en tareas como clasificación de imágenes, segmentación semántica y detección de objetos (O'Mahony et al., 2020).

La detección es uno de los subdominios más conocidos de la visión por computadora. Busca localizar y clasificar con precisión objetos específicos en una imagen. En las tareas de detección, se escanea la imagen para descubrir ciertas características que se asemejan a las características aprendidas por la red neuronal. Debido a su gran capacidad para capturar la información geométrica, como la ubicación de los objetos, las redes neuronales profundas se

han utilizado ampliamente para la detección y han mostrado un rendimiento sobresaliente (Liu et al., 2017).

- **Algoritmos de visión artificial para la detección de objetos**

  Se recomienda analizar las características principales asociadas a algoritmos especializados de visión artificial. Dicho análisis debe llevarse a cabo de forma bibliográfica y práctica para corroborar el rendimiento de las diferentes opciones existentes. La correcta elección de los componentes adecuados (arquitectura e hiperparámetros) para el entrenamiento y posterior implementación del modelo en el AllSkyCam7 determina el éxito del desarrollo del modelo.

  - SSD (Single Shot Multibox o Single Shot Detection)

    En este tipo de algoritmos, en lugar de buscar objetos en toda la imagen haciendo deslizar una ventana de manera secuencial, se divide la imagen mediante una rejilla y toda la imagen es transferida hacia una capa oculta de la red convolucional, así que los objetos de menor tamaño en la rejilla no transfieren características del objeto a la última capa. De esta manera se consume menos memoria al procesar las características que son casi imperceptibles. Existe otro algoritmo denominado YOLO que tiene dificultades para generalizar objetos pequeños, lo que se debe a las fuertes restricciones espaciales impuestas a las predicciones del cuadro delimitador o bounding box (Z. Q. Zhao et al., 2019). Se descartó desde el principio este algoritmo por la revisión de la literatura.

    Con el propósito de abordar el problema de detección de meteoros, se propuso el Detector Single Shot MultiBox (SSD), que se inspiró en los métodos predecesores MultiBox, RPN (Region Proposal Network) y la representación multiescala (Z. Q. Zhao et al., 2019). En vez de hacer uso de las cuadrículas fijas adoptadas en YOLO, el SSD aprovecha un conjunto de cuadros predeterminados con diferentes proporciones y escalas a partir de un mapa de características específico para hacer el espacio de salida de los cuadros delimitadores mucho más simple. Se debe mencionar que los problemas multiclase, llevan más tiempo y es difícil generar vectores de soporte para separar las clases múltiples (Jyothi et al., 2018).

    (Morera et al., 2020) y (Z. Q. Zhao et al., 2019) exponen que SSD, para manejar objetos con varios tamaños, fusiona predicciones de múltiples mapas de características con diferentes resoluciones, generando puntajes de confianza

relacionados con la presencia de cualquier objeto y su categoría. Además, (Morera et al., 2020) añade que SSD produce ajustes en las cajas delimitadoras (bounding boxes) para que se ajuste mejor con los objetos detectados. Siendo para este y (Horak & Sablatnig, 2019) una buena opción a la hora de desarrollar un modelo para aplicaciones ligeras ya que no repite el muestreo de características para cada cuadro delimitador (como el modelo Faster R-CNN) sino que lo hace una sola vez.

El entrenamiento de SSD necesita una colección de imágenes de entrada y sus correspondientes cajas de datos reales (ground truth bounding box) para cada objeto de clase contenido en ellas.

Las pirámides de características construidas sobre pirámides de imágenes en SSD (para abreviar, las llamamos pirámides de imágenes con características) forman la base de una solución estándar. De manera intuitiva, esta propiedad permite que un modelo detecte objetos en una amplia gama de escalas escaneando el modelo tanto en posiciones como en niveles de pirámide (Lin et al., 2017).

La arquitectura SSD está basada en CNN y para detectar las clases objetivo de objetos sigue dos etapas:

- Extrae los mapas de características y
- Aplica filtros convolucionales para detectar los objetos.

El Single Shot Multibox Detector (SSD), es uno de los primeros intentos de utilizar una jerarquía de características piramidales de ConvNets como si fuera una pirámide de imágenes con características. La pirámide de estilo SSD reutiliza los mapas de características de múltiples escalas de diferentes capas calculadas en el pase directo y, por lo tanto, no tendría ningún costo (Lin et al., 2017).

El modelo de arquitectura SSD, para (Z. Q. Zhao et al., 2019) y Liu (citado por Pérez, 2019) supera en rendimiento a Faster RCNN con los dataset de evaluación COCO y PASCAL VOC. De la misma manera, es 3 veces más rápido en sus tiempos de ejecución.

El uso de este tipo de modelos sobre grandes volúmenes de imágenes y datos en general provenientes de instrumentos astronómicos punteros, tendrá un gran

impacto positivo en la detección y clasificación de las incógnitas de nuestro limitado universo visible (Smithson et al., 2016).

Como sugiere el nombre (SSD), este método utiliza solo una red neuronal profunda para resolver el problema de detección. Otra ventaja del SSD es su simplicidad, que está relacionada con la velocidad de este método. La precisión de esta meta arquitectura es competitiva en relación con los otros métodos de última generación como Tiny-Yolo o Mask-RCNN.

(Zhang et al. 2021) divide en 2 categorías los tipos de detección.

- Descriptores globales que describen la imagen completa a partir de un solo vector con las características.
- Descriptores regionales que se centran en las regiones de interés (bounding box) de la imagen y la describe individualmente.

El problema crucial en las tareas de reconocimiento de imágenes es que no se conoce el tamaño del objeto en una imagen. Esto causa un costo computacional más grande ya que la rejilla de SSD debe evaluar cada posición en la imagen. La solución más utilizada para aplicaciones reales es utilizar una combinación de la denominada red pre entrenada, que se ejecuta en un potente hardware informático. Debido al problema del costo computacional, se implementó modelos pre entrenados con los que se puede trabajar libremente, entrenando con datasets personalizados en arquitecturas poderosas como SSD logrando tiempos computaciones significativas en aplicaciones de tiempo real (Horak & Sablatnig, 2019).

Implementar el método propuesto es de gran ayuda cuando las escenas presentan cambios debido al clima, perspectiva, estación del año usualmente visto en el reconocimiento de objetos al aire libre (Zhang et al., 2021).

1.2.3 Arquitecturas para la detección de objetos

Los métodos empleados con mayor éxito son crear una red neuronal de cero y entrenarla, aunque a un coste más grande computacional, o transferir conocimiento de una red neuronal pre entrenada (Shin et al., 2016).

Las redes de detección de objetos en imágenes proporcionan características de imagen genéricas que pueden transferirse a otros problemas de visión por computadora (Zoph, Vasudevan, et al., 2018). Esta transferencia también se puede realizar de un modelo ya entrenado (comúnmente llamado pre entrenado) a un modelo nuevo. De este modo, el nuevo modelo no empieza de cero y trae consigo el conocimiento de un modelo que según el objetivo del desarrollo se elegirá entre tantos que existen hoy en día.

- **SSD with EfficientDet-b0 + BiFPN**

  Se presenta la arquitectura usada para el entrenamiento del modelo. La versión más liviana de la familia de los EfficientDet. Su ventaja principal es que el extractor de características utiliza una técnica denominada BiFPN (Weighted bidireccional feature pyramid network).

  En la resolución de problemas de visión artificial, se conoce que las imágenes están representadas por diferentes características y en diferentes niveles. Es por eso por lo que uno de los componentes esenciales en la detección de objetos es el extractor de características (feature extractor). Este extractor es capaz de analizar patrones en el color, en las texturas y en las figuras de las imágenes (Tiwari et al., 2013). A continuación (Pérez, 2019) extrae en los siguientes principales niveles, a partir de las descripciones de (Thakur et al., 2017) que presenta las características por niveles, los extractores de características que son capaces de analizar.

  - **Características de nivel 1 y 2:** referente a la intensidad de color, contraste, saturación, entre otros; de un píxel en específico.
  - **Características de nivel 3:** las características de nivel 3 son el resultado de aplicar un algoritmo de visión específico a las características de nivel 1 y 2.
  - **Vectores de características:** contenedores de característica de pixeles. Los mismos son creados con el objeto de reducir tiempo de cómputo, o presentar datos de entrada compatibles con ciertos algoritmos.

- **BiFPN**

  Fusiona las características en múltiples escalas ayudando a agregar características a diferentes resoluciones. El diseño del extractor en la arquitectura seleccionada es el siguiente:

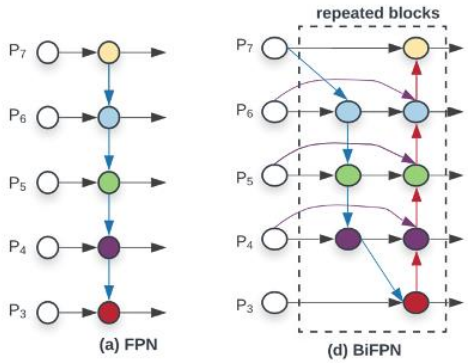


Fig. 4. Diseño del extractor de características de BiFPN.

Fuente: (Tan et al., 2020)

A la izquierda se puede observar un extractor de características convencional y a la derecha, el extractor con el que se trabajará gracias a su mejora tanto en precisión como en eficiencia (Tan et al., 2020).

Después de conocer el extractor de características, se presenta la arquitectura completa de EfficientDet D0.

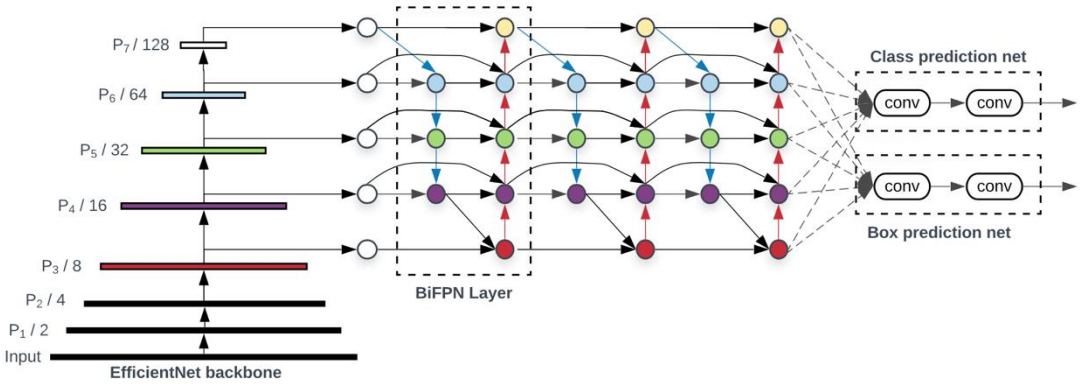


Fig. 5. Arquitectura de EfficientDet.

Fuente: (Tan et al., 2020)

Donde claramente se puede observar como la entrada de la red es una estructura piramidal que transformará los datos de entrada a diferentes escalas, siendo estos los valores de entrada a los nodos de la primera capa oculta de la red. A continuación, pasa por tantas capas ocultas como clases se tiene y es dada por la siguiente ecuación.

$$D_{box} = D_{class} = 3 + [\phi/3]$$

Donde el coeficiente ϕ es 0 debido a que la versión usada será EfficientDet 0 por lo que el total de capas ocultas es de 3.

Resolución de las imágenes de entrada: es dada por la siguiente ecuación, y por la versión que se esté usando (D0, D1, …, D6, D7).

$$R_{input} = 512 + \phi \cdot 128$$

Donde el coeficiente ϕ es 0 debido a que la versión usada será EfficientDet 0 y por lo tanto la resolución de la imagen de entrada será de 512 píxeles por 512 píxeles (Tan et al., 2020). Se expone una tabla con las características que tiene cada versión de la familia EfficientDet.

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Fig. 6. Configuraciones de escalado para EfficientDet D0-D7.
Fuente: (Tan et al., 2020)

Lo que se consigue con esta arquitectura es una mejora considerable en la eficiencia del modelo sin perder precisión gracias a su reducido número de parámetros 3.9 millones y de los FLOPs (cálculos de punto flotante) 2.5 billones. Llegando a un mAP de 33.8, reduciendo en 28 veces la cantidad de FLOPs que en YOLOv3 (Tan et al., 2020). La reducción de estos parámetros entrenables (FLOPs) favorece a que el modelo no se sobreajuste (Jaramillo & Bustamante, 2020).

Una comparativa entre modelos se presenta a continuación para observar el rendimiento que es capaz de dar la familia de los EfficientDet. Esto se traduce en una reducción de tiempos de ejecución notable. Estos valores pueden verse incluso reducidos si la selección de hiperparámetros mejora la optimización de la arquitectura y el modelo.

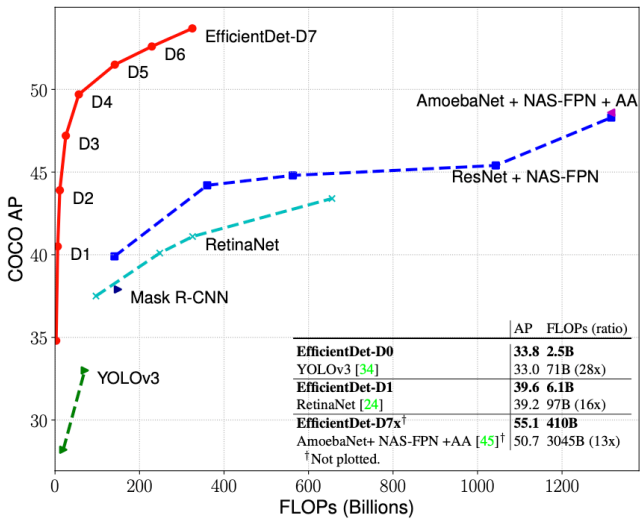


Fig. 7. Relación FLOPs de la arquitectura y la precisión con la evaluación COCO.

Fuente: (Tan et al., 2020)

1.3 Deep Learning

1.3.1 Redes neuronales profundas

Una de las tecnologías más llamativas actualmente es el aprendizaje profundo, logrando grandes éxitos en áreas distinguidas como la visión por computadora y la robótica. En los últimos cinco años, investigadores en estas áreas han apostado cada vez más por las redes neuronales profundas al abordar problemas de reconocimiento de objetos (Zhang et al., 2021).

Un paradigma informático inspirado en el funcionamiento del cerebro humano. Al igual que el cerebro humano, está compuesto por muchas células o "neuronas", cada una de las cuales realiza una operación simple e interactúan entre sí para tomar una decisión (O'Mahony et al., 2020).

La investigación en el campo del análisis de imágenes utilizando redes neuronales se ha ralentizado un poco en los últimos tiempos. Esto se debe en parte a la creencia incorrecta en torno al nivel de complejidad y conocimiento necesarios para comenzar a modelar estos algoritmos de aprendizaje automático magníficamente poderosos (O'Shea & Nash, 2015).

La idea básica de una red neuronal es la utilización de un conjunto de unidades computacionales (parámetros) muy sencillas, aunque abundantes (millones), que "mediante

operaciones matemáticas básicas son capaces de aprender funciones complejas" (Zhang et al., 2021, p. 5) y ofrecer una salida a partir de unos datos de entrada. Actualmente, la estrategia más generalizada que se tiene para ello es disponer las neuronas en capas de procesamiento, de forma que haya una capa de entrada, una de salida y un cierto número de capas intermedias, llamadas capas ocultas (Garbisu Arocha, 2016).

Tanto (Mittal et al., 2020) como (Smithson et al., 2016) coinciden en que es necesario una cantidad relativamente grande de datos para entrenar modelos y técnicas como minería de datos o árboles K-D. Y que estos datos multidimensionales solo han estado disponibles en los últimos años. A demás (Zhang et al., 2021) aporta que el dataset para el entrenamiento debe ser de buena calidad para un mejor rendimiento de los modelos en producción.

El aprendizaje profundo se ha estudiado ampliamente en el campo de la visión por computadora y, como consecuencia, ha surgido una gran cantidad de enfoques relacionados (Jia, 2017). Siendo su objetivo estudiar algoritmos informáticos flexibles que sean capaces de aprender. Este aprendizaje siempre se basa en algún tipo de experiencia basada en datos o instrucciones de entrada conocidos con anterioridad (Karanam et al., 2020).

La razón principal de ser llamadas redes neuronales profundas es que la red puede extraer las características abstractas más complejas de las imágenes cuando se profundiza el nivel de la red (He & Li, 2019).

Las arquitecturas DL (Deep Learning) resuelven problemas que requieren funciones complejas y muy variables. En estos problemas se trabaja con conjuntos de datos muy grandes y, en su mayoría, no etiquetados (Affonso et al., 2017). Los desarrolladores de modelos de Deep Learning deben tener cierto nivel de experticia para poder adecuar los parámetros y diseño de arquitectura al objetivo del modelo.

El aprendizaje profundo (DL) se utiliza en el dominio del procesamiento de imágenes digitales para resolver problemas difíciles (coloración, clasificación, segmentación y detección de objetos). Los métodos de DL, como las redes neuronales convolucionales (CNN), mejoran principalmente el rendimiento de la predicción mediante el uso de macrodatos (datos a gran escala) y abundantes recursos informáticos, y han superado los límites de lo que era posible (O'Mahony et al., 2020).

Es de mencionar que los abundantes recursos de software y hardware permiten programar y entrenar redes de manera conveniente y eficiente. Zhang et al. (2021) expone: "el uso de la

unidad de procesamiento de gráficos (GPU) hace que entrenar una red sea más rápido que si se usara solamente una unidad de procesamiento computacional CPU" (p.5). (NVIDIA, 2014) ha demostrado que el incremento mínimo de velocidad de un procesamiento paralelo mediante GPU (Utilizando CUDA) frente a uno por CPU es del 200%. Por todas las ventajas que conlleva el uso de GPU para el entrenamiento de modelos de red neuronal profundos, se ha hecho la inversión de la herramienta de Google Colab Pro mientras se desarrolla la presente investigación.

El área del procesamiento de imágenes digitales ha experimentado muchos cambios dramáticos recientemente y en un período muy corto de tiempo. Tanto es así, que nos ha llevado a cuestionarnos si las técnicas de CV que estaban de moda antes de la explosión de la IA siguen siendo relevantes (O'Mahony et al., 2020).

(Zhang et al., 2021) expone que numerosos artículos concuerdan en que las redes neuronales profundas, especialmente las redes neuronales convolucionales, presentan un rendimiento superior en comparación a los métodos tradicionales.

Aunque (O'Mahony et al., 2020) destaca algunos casos en los que las técnicas tradicionales de CV son útiles y que todavía hay algo que ganar con los años de esfuerzo dedicados a su desarrollo, incluso en la era de la inteligencia basada en datos.

Las redes neuronales convolucionales profundas (DCNN) han llevado el rendimiento de los sistemas de visión por computadora a alturas vertiginosas en una amplia gama de problemas de alto nivel, incluida la clasificación de imágenes y detección de objetos (Chen et al., 2018).

Las redes de aprendizaje profundo están clasificadas en dos grandes categorías: las de análisis por región y las de flujo directo. Dentro de esta clasificación, se usará SSD (Single Shot Multibox) como se expuso en la sección anterior. La ventaja del algoritmo de flujo directo ante uno por regiones es su veloz procesamiento, aunque pierde los objetos más pequeños de las imágenes (Pérez, 2019).

Para terminar esta sección hay que mencionar que las redes neuronales profundas, también se han utilizado en los campos más exóticos como la astronomía y la paleontología para la detección de enanas marrones y el descubrimiento de fósiles, respectivamente.

1.3.2 Redes neuronales convolucionales (CNN)

Para (Ordóñez & Roggen, 2016) y (Smithson et al., 2016) las redes neuronales convolucionales (CNN) son un tipo de DNN (red neuronal profunda) con la capacidad de actuar como extractores de características. Apilando varios operadores convolucionales para crear una jerarquía de características cada vez más abstractas. Dichos modelos pueden aprender automáticamente múltiples capas de jerarquías de características mediante estos operadores convolucionales, de la capa de entrada pasan a la salida a la siguiente capa los pesos y sesgos obtenidos. Este proceso también es denominado "aprendizaje de representación" (Lukic et al., 2020), que es lo que SSD propone en su arquitectura.

El éxito de las técnicas de DL, y más específicamente de la CNN, se debe principalmente a su rendimiento predictivo superior en comparación con otras técnicas de ML (Machine Learning). Por lo general, las tareas de procesamiento de imágenes abordadas con éxito por CNN son problemas de reconocimiento de imágenes y de escritura a mano (Affonso et al., 2017).

Las redes neuronales convolucionales (CNN) son análogas a las ANN tradicionales en el sentido de que están compuestas por neuronas que se auto optimizan mediante el aprendizaje (O'Shea & Nash, 2015). La única diferencia notable entre las CNN y las ANN tradicionales es que las CNN se utilizan principalmente en el campo del reconocimiento de patrones dentro de imágenes.

(Lukic et al., 2020) expone que la razón para que el rendimiento de las CNNs sea tan alto, es el uso de filtros en sus capas convolucional y concuerda con (Manassés, 2018) y (R. Zhao et al., 2020) en que son capaces de aprender el extractor de características y el clasificador, al mismo tiempo.

Se ha encontrado que las CNN son altamente efectivas y también son el más comúnmente utilizado en diversas aplicaciones de visión por computadora (Jia, 2017). La arquitectura de CNN brinda la oportunidad de optimizar conjuntamente varias tareas relacionadas (por ejemplo, Fast R-CNN o SSD combinan la clasificación y la regresión de bounding box) (Z. Q. Zhao et al., 2019).

Las CNN son una clase especial de red neuronal desarrollada para la aplicación de detección y clasificación de imágenes que comprende conjuntos de capas para diferentes propósitos. Las primeras capas extraen atributos locales de las imágenes de entrada, luego las capas más profundas extraen atributos globales. El propósito principal de esta organización

jerárquica es proporcionar que la señal de entrada pase a través de las capas sucesivas, de modo que la posición de los atributos extraídos tenga menos influencia en la salida final de la red (Luís & Medeiros, 2019).

Para (Dong et al., 2016) varios factores son de importancia en el proceso de crear un modelo de red neuronal profunda:

- La implementación de entrenamiento eficiente en GPUs modernas y potentes,
- la implementación de la unidad lineal rectificada (ReLU) que hace que la convergencia sea mucho más rápida mientras todavía presenta buena calidad, y
- el fácil acceso a una gran cantidad de datos (como ImageNet) para entrenar modelos más grandes.

En primer lugar, estos modelos suelen entrenarse con una gran cantidad de datos de muy alta dimensión, como imágenes, videos y datos de texto (Garcia & Alegre, 2019). Las CNN se han estudiado ampliamente para el reconocimiento de lugares. Una desventaja es que se necesita una gran cantidad de datos etiquetados al entrenar los modelos de CNN. Recientemente se ha podido utilizar las CCNN en problemas relacionados con la astronomía ya que hace pocos años atrás era impensable dada la magnitud de datos necesarios.

Para (Xin & Wang, 2019) los pasos para entrenar un modelo de red neuronal convolucional son los siguientes:

- Entrada: La entrada es una colección de N imágenes; cada etiqueta de imagen es una de las etiquetas de clasificación K. Este conjunto se denomina conjunto de entrenamiento.
- Aprendizaje: utilizar el conjunto de entrenamiento para aprender exactamente cómo es cada clase. Este paso generalmente se denomina clasificador de entrenamiento o aprendizaje de un modelo.
- Evaluación: El clasificador se utiliza para predecir las etiquetas de clasificación de imágenes que no ha visto y para evaluar la calidad de los clasificadores. Comparamos las etiquetas predichas por el clasificador con las etiquetas reales de la imagen.

Las redes neuronales convolucionales (CNN) son uno de los enfoques de aprendizaje profundo más notables donde se entrenan múltiples capas de manera robusta (Jia, 2017). Siendo las capas más profundas las que toman características de más bajo nivel a partir de la información de entrada. En las capas más externas características como los bordes o los

colores son examinadas. Por otro lado, en las capas más profundas se analizan características semánticas de alto nivel (Zhang et al., 2021) .

Una de las diferencias clave es que las neuronas que forman las capas dentro de la CNN están compuestas por neuronas organizadas en tres dimensiones, la dimensión espacial de la entrada (altura y ancho) y la profundidad. La profundidad no se refiere al número total de capas dentro de la red neuronal, sino a la tercera dimensión referente al número de canales de colores. (O'Shea & Nash, 2015).

Dentro de los modelos entrenados con otros modelos pre entrenados y con un conjunto de datos lo suficientemente grandes existen capas intermedias y completamente conectadas que servirán de representadores de características para las futuras predicciones en producción (Zhang et al., 2021). Desde los vectores de imagen sin procesar de entrada hasta la salida final de la puntuación de la clase, toda la red seguirá expresando una única función de puntuación (el peso). La última capa contendrá funciones de pérdida asociadas con las clases, y aun se aplicarán todos los trucos habituales que en las ANN tradicionales (O'Shea & Nash, 2015).

El objetivo de una capa convolucional es aplicar operaciones convolucionales en los datos recibidos de la capa anterior para encontrar patrones a menor escala que puedan ayudar a la predicción en el futuro. Al apilar múltiples capas convolucionales, el modelo puede construir sucesivamente características más complejas que pueden operar como un conjunto de funciones en secuencia, codificadas por capas, que cambian la representación de los datos para facilitar la predicción (Garcia & Alegre, 2019).

La principal diferencia de CNN es que puede extraer características de los píxeles de la imagen sin procesar, mientras que los algoritmos de clasificación más tradicionales esperaban un conjunto de características que describían una imagen como entrada (Affonso et al., 2017).

Las CNN están especialmente diseñadas para problemas de clasificación supervisados (Manassés, 2018). El aprendizaje supervisado es el aprendizaje a través de insumos pre etiquetados, que actúan como objetivos. Para cada ejemplo de entrenamiento habrá un conjunto de valores de entrada (vectores) y uno o más valores de salida designados asociados. El objetivo de esta forma de entrenamiento es reducir el error de clasificación general de los modelos, mediante el cálculo correcto del valor de salida del ejemplo de entrenamiento por entrenamiento (O'Shea & Nash, 2015). De esta manera, es capaz de

generalizar las predicciones de prueba que no tuvo acceso durante el proceso de entrenamiento (Garcia & Alegre, 2019).

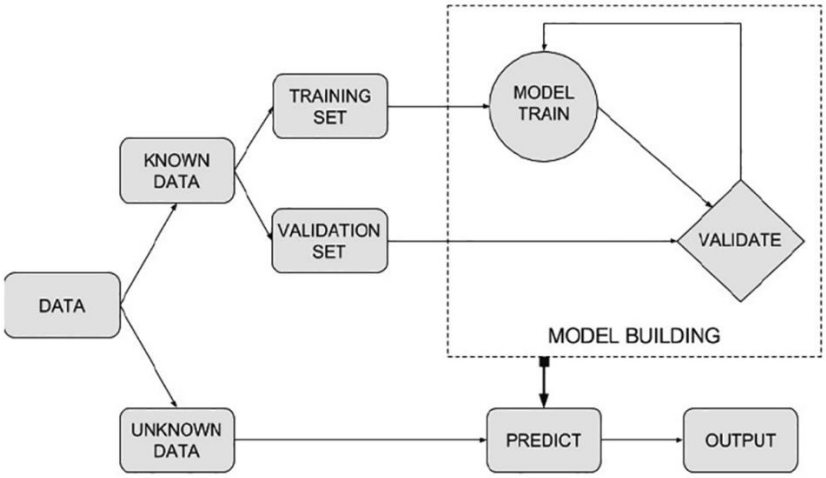


Fig. 8. Estructura del aprendizaje supervisado.
Fuente: (Karanam et al., 2020)

- **Artefactos de las Redes Neuronales Convolucionales**

La red neuronal de convolución (red neuronal celular) se ha convertido en la técnica más eficaz para resolver una gran cantidad de tareas de aprendizaje automático (R. Zhao et al., 2020). También denominado, ConvNet consta de capas de detección de características y clasificación. Una ConvNet se compone de varias capas. Algunas de ellas son, capas convolucionales, capas de agrupación máxima (max pooling) o agrupación media y capas completamente conectadas (fully connected) (Krishna et al., 2018).

Los artefactos que usualmente manejan las redes neuronales convolucionales son:

- **Capa de entrada:** esta capa obtiene matrices tridimensionales representando imágenes de dos dimensiones. Los datos de imagen (píxeles) se convierten en neuronas de tres dimensiones, ancho por alto por número de canales (O'Shea & Nash, 2015).

- **Convolución**: En CNN, un mosaico común es de 3x3 o 5x5 píxeles. Del mapa de características de entrada se extrae y se agrega mediante filtros de convolución. Los filtros se deslizan sobre la cuadrícula de entrada de la función

de izquierda a derecha y de arriba a abajo, un píxel a la vez, extrayendo cada mapa respectivo. O en el caso de SSD, mediante la malla a diferente escala. La CNN "aprende" los valores óptimos para las matrices de filtro durante el entrenamiento, lo que le permite extraer características relevantes cada vez (Karanam et al., 2020).

- **Función de activación:** La función de activación recibe un valor de entrada y se encarga de calcular la suma ponderada entre las entradas y los sesgos, a partir de esto se decide si el nodo o neurona se activará o no, devolviendo una salida que corresponde a la decisión o el peso. A la suma ponderada se le considera la operación multiplicación de cada valor de entrada por su peso y al final sumándose cada una de ellas (Nwankpa et al., 2018).

  Habitualmente los valores de salida están normalizados en rangos entre [0,1] (sigmoid o de una sola clasificación) o [-1,1] (tanh para clasificación multi clase). Pero, la función ReLU es la función de activación más utilizada para las capas ocultas, ya que se utiliza en casi todas las redes neuronales convolucionales o de Deep Learning. La clave de ReLU está en que todos los valores negativos se vuelven cero, y eso significa que cualquier entrada negativa dada a la función de activación de ReLU convierte el valor en cero inmediatamente. Esto puede ayudar mucho en la simplificación computacional ya que todos los valores iguales a 0 son inmediatamente descartados (dichas neuronas son irrelevantes) (Nwankpa et al., 2018).

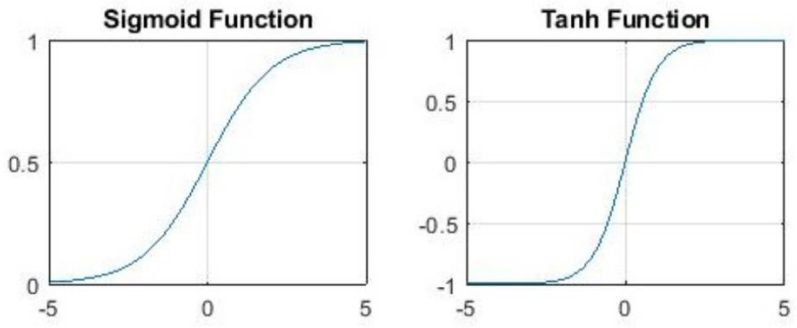


Fig. 9. Función de activación Sigmoid y Tanh.
Fuente: (Nwankpa et al., 2018)

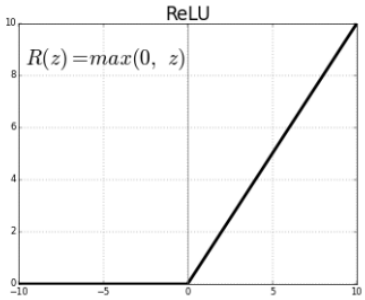


Fig. 10. Función de activación ReLU.

Fuente: (Farrukh, 2020)

La nueva propuesta por (Zoph, Le, et al., 2018) investigadores de Google, propone una mejora en la función RELU denominada SWISH. Diseñada para cualquier tipo de arquitectura de red neuronal, pero sobre todo para LTSM. La función SWISH tiene una mejora de casi 1% para las arquitecturas livianas como la familia MobileNet y 0,6% para modelos más robustos como la familia ResNet con el conjunto de datos ImageNet de Google.

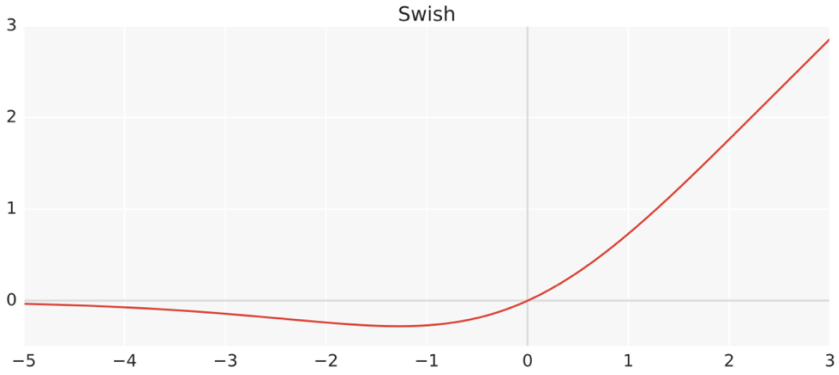


Fig. 11. Función de activación SWISH.

Fuente: (Zoph, Le, et al., 2018)

Una rápida comparativa entre SWISH y RELU se expone a continuación para un entrenamiento a 600 mil iteraciones bajo la arquitectura Inception-RenNet-v2 donde el accuracy se ve incrementado considerablemente al usar SWISH.

Fig. 12. Comparativa SWISH y RELU.

Fuente: (Zoph, Le, et al., 2018)

- **Pooling:** La agrupación o pooling es el siguiente paso después de ReLU o SWISH al finalizar las capas ocultas. También conocida como capa de muestreo descendente. Reduce las dimensiones espaciales, pero no su profundidad (Karanam et al., 2020).

- **Fully connected layers:** Al final de una red neuronal convolucional, se encuentran una o más capas completamente conectadas. Su tarea es realizar una clasificación basada en la característica que recopilan las convoluciones entre todas las capas ocultas. En su mayoría, las neuronas están completamente conectadas hasta el final. Esta última capa está completamente conectada y contiene una función de activación SoftMax o Sigmoid que da un valor de probabilidad entre -1 y 1 o 0 y 1 respectivamente para cada una de las etiquetas de clasificación del modelo que se intenta predecir (Karanam et al., 2020).

1.4 Tensorflow

1.4.1 Framework Tensorflow

Tensorflow es una plataforma de código abierto para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores impulsar un aprendizaje automático

innovador y, a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología de ML (https://www.tensorflow.org).

Compila y entrena modelos de ML con facilidad mediante API intuitivas y de alto nivel como la API para Detección de Objetos de Tensorflow en su segunda versión, con ejecución inmediata, que permite una interacción de modelos inmediata y una depuración fácil.

1.4.2 API Detección de Objetos

Esta herramienta es muy versátil y poderosa a la hora de crear modelos personalizados con las tecnologías de Google, Tensorflow. Existen tutoriales en su sitio web con los que se ha guiado para hacer la parte práctica del presente proyecto. Actualmente existen tutoriales para las versiones de Tensorflow 1.x y una más actual para la versión 2.x.

1.4.3 Modelos para detección ZOO Tensorflow 2

Los modelos para detección de ZOO son una colección de modelos de detección entrenados previamente en el conjunto de datos COCO 2017. Estos modelos pueden ser útiles para la detección de los objetos ya diferenciados en el dataset de COCO 2017 pero también pueden usarse como base para un post entrenamiento con un dataset personalizado aprovechando sus arquitecturas.

1.5 Metodología de desarrollo de software

Las metodologías de desarrollo de software tienen varios conjuntos de reglas y pautas para el proceso de investigación, planificación, diseño, desarrollo, prueba y mantenimiento de productos de software (Saranya et al., 2017).

Las personas que trabajan para organizaciones profesionales de desarrollo de software descubren el gran aporte que otorga el trabajar con metodología definida (Gechman, 2013). Una metodología de desarrollo de software es una forma de gestionar un proyecto de desarrollo de software. Por lo general, esto aborda problemas como la selección de funciones para su inclusión en la versión actual, cuándo se lanzará cierta versión del software, quién trabaja en qué y qué pruebas se realizarán (Gechman, 2013). Ninguna metodología es la mejor para todas las situaciones por lo que se recomienda hacer un análisis de las diferentes

alternativas para poder trabajar bajo una metodología ajustada al proyecto. Todas las fases de una metodología en conjunto permiten reducir costes y tiempo cuando son aplicadas adecuadamente. Incrementando las posibilidades de éxito del proyecto.

1.5.1 Proceso Knowledge Discovery in Databases (KDD)

A continuación, se exponen las fases que tiene esta metodología en Deep Learning tratando los datos. Cuando utilizamos el aprendizaje en profundidad para estudiar la tecnología de reconocimiento de la calidad de la imagen, se encuentra que tanto para (He & Li, 2019) como para (Xin & Wang, 2019) se estudia principalmente desde tres aspectos: preprocesamiento de imágenes, extracción de características y reconocimiento.

- **Recolección**

  Se recolectan las imágenes que serán usadas como entrada en el entrenamiento del modelo.

- **Selección**

  Desafortunadamente, el enfoque DL requiere una anotación de muestras buenas y malas, que debe hacerse manualmente. Representa un trabajo humano agotador y que consume mucho tiempo (Horak & Sablatnig, 2019). Especialmente en eventos astronómicos, ya que suelen ser figuras muy sutiles e insignificantes. De los miles de muestras se debe ir seleccionando las mejores imágenes que contengan ejemplares del objeto a detectar y clasificar.

- **Preprocesamiento**

  El preprocesamiento de la imagen es principalmente para mejorar la imagen objetivo y debilitar la influencia del fondo en el reconocimiento. Entre otros, incluye procesamiento de grises, eliminación de ruido, detección de bordes y segmentación de imágenes enriqueciendo el dataset con nuevas formas, ubicaciones y direcciones. Un preprocesamiento razonable puede garantizar la calidad de la imagen, desempeñar un papel muy bueno en la extracción de características y el reconocimiento de imágenes posteriores, mejorando la precisión del reconocimiento de imágenes (He & Li, 2019).

- **Redimensionamiento**

Comprende una operación básica de cambio de dimensiones del archivo de imagen a procesar. Trabajar con archivos grandes no implica resultados mejores, a más de elevar los tiempos de procesamiento. Lo cual resulta crítico para aplicaciones que manejan cientos de datos a procesar (Pérez, 2019).

Las imágenes sin procesar se organizan en matrices de píxeles. (Jyothi et al., 2018) propusieron demostrar que el aumento en el número de fotogramas mejora el rendimiento en la clasificación hasta cierto punto. Con los trabajos en los que tuvieron que solventar una problemática similar se aconseja llegar a un punto medio del tamaño de imagen. Donde las características sean suficientemente claras para poder trabajar. En el sistema AllSkyCam7 se manejan 25 fotogramas por segundo cuando se registran videos en calidad SD (480 píxeles) y en calidad HD (720 píxeles) e imágenes de los posibles meteoros a 640 píxeles y 1080 píxeles.

Antes de la implementación de cualquier aplicación de visión artificial, se debe someter la imagen obtenida a un preprocesamiento que facilite la ejecución del algoritmo que se haya seleccionado (Pérez, 2019).

- **Transformación**

Las técnicas de aprendizaje profundo pueden aprovechar el aumento de imágenes, ya que genera más muestras de entrenamiento, lo que debería mejorar el rendimiento hasta cierto umbral (Lukic et al., 2020).

La extracción de características o transformación consiste en estudiar cómo extraer características más eficientes de la imagen original y lograr una relación de mapeo precisa entre la imagen y un determinado vector o espacio de características. De modo que podamos usar el método de mapeo para transformar las características de la alta dimensión de la imagen original en nuevas características del espacio de baja dimensión (He & Li, 2019).

- **Entrenamiento**

Con un alto uso de la computación y la memoria y una enorme cantidad de datos de entrenamiento leídos, es necesario utilizar el alto rendimiento de manera efectiva (R. Zhao et al., 2020). Herramientas cloud potentes están disponibles actualmente para este tipo de operaciones. Google Colab es una de ellas, donde el préstamo de servicios GPU están disponibles por un precio razonable.

En este proceso, con los TFRecord como entrada a la arquitectura elegida por el investigador. El modelo genera inicialmente unos pesos y bias para cada neurona de entrada de manera aleatoria para romper cualquier patrón simétrico entre las neuronas iniciales. Si las predicciones son demasiado diferentes de lo que deberían ser, el modelo modifica los parámetros del modelo significativamente, con la esperanza de que esto dé como resultado un mejor desempeño que antes. De lo contrario, cuando las predicciones son relativamente similares a las etiquetas esperadas, la configuración de los parámetros no cambia mucho (Garcia & Alegre, 2019).

En el entrenamiento con Colab se puede tener un feedback del estado del modelo en tiempo real. La capacidad de visualizar el modelo en tiempo real puede ayudar a los diseñadores a tomar decisiones conscientes sobre qué cambiar en el modelo mucho más rápido que si tuvieran que esperar hasta que se complete el entrenamiento para analizar si el resultado obtenido es bueno o no (Garcia & Alegre, 2019).

Cada vez que termina una iteración de entrenamiento, los valores de los pesos actualizados pasan por cada neurona de la red en sentido contrario para actualizar cada nodo de la red. A esta acción se le denomina Back Propagation.

Los hiperparámetros, ya visto en secciones anteriores, son clave. Un uso acertado de una función de activación para la última capa (actualización de los pesos) puede garantizar que los valores no se estanquen en mínimos locales que ocurre cuando el modelo cree haber llegado al mínimo óptimo, siendo solamente el mínimo de la iteración y no global.

- **Evaluación**

La evaluación se realiza con la misma API de detección de objetos de Tensorflow 2. Ya que consta de una función para medir la robustez del modelo ante datos que jamás ha visto con técnicas de evaluación en la visión artificial muy comunes como la clasificación correcta o incorrecta del objeto de estudio o la Intersección sobre la Unión (IOU).

Otras evaluaciones son posibles para Clasificación como la tasa de Precisión, Recall, F1 Score o pérdida en clasificación. Y para Regresión existe la Precision, Recall, tasa de pérdida de localización, regularización y pérdida total.

- **Despliegue**

Elegir dónde se desplegará es crucial, ya que existen procesadores y sistemas operativos legacy que no son capaces de ejecutar modelos de Tensorflow actuales. El sistema operativo que la AllSkyCams recomienda para su buen funcionamiento es cualquier distribución base Unix.

Es común desarrollar un servicio web que se consume por clientes web o móvil para evitar cualquier tipo de problemas con capacidad de procesamiento. En proyectos futuros se propondrá realizar la apertura a microservicios para todos los actuales servicios del AllSkyCams además del que se desarrolla en la presente investigación.

- **Conocimiento**

Al final de la metodología KDD, se obtiene el conocimiento necesario para refinar el tratamiento de los datos y poder continuar con la investigación. En este proceso se valida verdaderamente si el modelo está haciendo lo esperado o no.

1.6 Trabajos relacionados y selección de hiperparámetros

En las ciencias e investigación astronómicas actuales se ha analizado un creciente uso del Machine Learning en especial del Deep Learning como principal herramienta para la construcción de software que permita discernir e identificar ciertos eventos astronómicos. Algunos como la morfología de galaxias y su evolución en (Walmsley et al., 2022), (González et al., 2018), (Barchi et al., 2020) y (Jimenez et al., 2020). La identificación de cráteres y sus tamaños en la Luna en (Ali-Dib et al., 2020) y (Silburt et al., 2019) entre otros muchos casos de estudio.

La mayoría de los estudios citados han resuelto una problemática similar a la del presente trabajo por lo que se toman sus métodos y resultados como referencia para poder desarrollar el capítulo siguiente. Estos en su mayoría también comparten ciertas cualidades como, haber obtenido el dataset de telescopios potentes como el DECaLS (Dark Energy Camera Legacy Survey) en (Walmsley et al., 2022) o el Sloan Digital Sky Survey Data Release 7 (SDSS-DR7) en (Barchi et al., 2020). El proceso de entrenamiento se desarrolló bajo GPUs de punta como la Titan-X Nvidia GPU en (González et al., 2018), la NVIDIA Tesla K40c GPU en (Čokina et al., 2021) o la GeForce RTX 2060 6 GB RAM en (Raju & Das, 2021).

Otras características importantes que comparten fueron el uso del marco de trabajo Tensorflow y su API para detección de objetos además de la biblioteca Keras, salvo en (González et al., 2018) que utilizaron DarkNet. En cuanto a la cantidad de imágenes en sus datasets se pudo apreciar una cantidad generosa, ninguno de los estudios trabajó con menos de 5 mil ejemplares (Hosenie et al., 2021) siendo este caso particular ya que todos los demás van de los 50 mil ejemplares (Ali-Dib et al., 2020) a 900 mil ejemplares (González et al., 2018). En cuanto a la división de los dataset para entrenamiento, validación y test su mayoría lo hicieron en proporción 80/20 saltándose la validación. Se encontraron casos singulares como en (Hosenie et al., 2021) que divide el dataset en proporción de 50/20/30, en (Shilon et al., 2019) que lo hace en proporción de 62/38 o en (Shilon et al., 2019) que lo hace en proporción de 53/40/6. La mayoría de estos estudios clasifican más que detectan y lo hacen con un número de clases de entre 1 y 2 habiendo una excepción en (Shilon et al., 2019) que clasifica 40 diferentes tipos de galaxias.

A parte de (Čokina et al., 2021) que hace uso de redes convolucionales de tipo RNN y LSTM (ya que categoriza las curvas de luz en estrellas siendo un proceso más complejo). Todas las demás utilizan redes neuronales convolucionales simples.

La mayoría de los estudios propusieron entrenar el modelo de Deep Learning a partir de un modelo pre entrenado, entre los maso usados fueron Resnet-101, Faster R-CNN y GoogleNet en (Barchi et al., 2020), (Bird et al., 2021), (Jimenez et al., 2020), (Baek et al., 2021), (Raju & Das, 2021). Un modelo pre entrenado bien justificado para el estudio de fenómenos astronómicos aparece en (Walmsley et al., 2022) que es el EfficientNet D0. Sus autores exponen lo siguiente “el modelo EfficientNet D0 es usado para la detección múltiple y clasificación de objetos. Optimiza tanto la precisión como el costo computacional de predicción (FLOPS). Este balance es particularmente útil en investigación astrofísica con limitado acceso a recursos de GPU.”

En las únicas investigaciones (Walmsley et al., 2022) y (Ali-Dib et al., 2020) que se pudo obtener conocimiento del proceso conocido como “data augmentation” proponen giros randómicos horizontales y verticales, seguido de una rotación de un ángulo aleatorio en un rango (0, π) o a 90, 180 y 270 grados.

El tamaño de las imágenes de entrada varía mucho y seguramente dependen de la herramienta para recopilar estos datos. Se pueden apreciar en su mayoría casos en los que el tamaño de entrada ha sido de 64 por 64 píxeles y 100 por 100 píxeles. Cuanto más grande es la imagen se puede apreciar una reducción de la precisión del modelo como en (González

et al., 2018) que con un tamaño de entrada de 2048x1489 obtuvo una precisión de 80% mientras que en (Walmsley et al., 2022) con un tamaño de entrada de 224x224 se obtuvo una precisión de 99%, en (Čokina et al., 2021) con un tamaño de entrada de 100x100 se obtuvo una precisión de 98% o en (Armstrong & Fletcher, 2019) que con un tamaño de entrada de 64x64 se obtuvo una precisión de 99,92%. Las imágenes de entrada se recomiendan sean cuadradas de dimensión alto x ancho igual según los estudios revisados.

En cuanto al optimizador usado por lo estudios, el más utilizado fue Adam, habiendo casos aislados como en (Tanoglidis et al., 2021) que utilizaron Adadelta una variante de Adam o en (Tanoglidis et al., 2021) y (Armstrong & Fletcher, 2019) que utilizaron SGD.

En cuanto al hiperparámetro conocido como “batch size” que es el número de ejemplares que ve el modelo simultáneamente en cada iteración es muy variado, aunque tienden por usar 128 y 64 en (Armstrong & Fletcher, 2019), (Raju & Das, 2021) y (Tanoglidis et al., 2021) y (Tanoglidis et al., 2021). El resto de los estudios rondan los 8, 24, 32, y 256.

El hiperparámetro conocido como “Learning rate” fue más difícil de conseguir, pero los mejores resultados fueron con 0,001 ;0,0002; 0,0004 e incluso en (Armstrong & Fletcher, 2019) se aplica un “schedule” de “Learning rates” en el siguiente orden (0,001; 0,0005; 0,0001).

Por último, se tiene los hiperparámetros “epochs” e “iterations” que reflejan el tiempo que el modelo estará aprendiendo de los datos de entrada. En los estudios revisados algunos trabajan con “epochs” y otros con “iterations”. Ejemplo de esto es (cita 3) que entrenó su modelo en 20 mil iteraciones. Sin embargo, (cita 5) entrenó su modelo en 80 épocas. (cita 7) lo hizo en 60 mil iteraciones y otros como (cita 8), (cita 11) y (cita 12) lo hicieron en 100 épocas.

CAPÍTULO 2

Desarrollo de la propuesta

El modelo ha sido acuñado *AllSkyNet* por su cualidad de ser un modelo que detectará meteoros usando el sistema integrado AllSkyCam7. A continuación, se desarrollará la parte práctica siguiendo los lineamientos de (Saranya et al., 2017) para una metodología KDD.

En esta segunda parte del proyecto se expone el desarrollo de un modelo de Deep Learning construido con la API de detección de objetos de Tensorflow 2. Los archivos de ejecución para entrenamiento y luego puesta en producción se encuentran en el repositorio de Tensorflow (https://github.com/tensorflow/models) y el tutorial de Roboflow para entrenar tu propio modelo de detección de objetos (https://blog.roboflow.com/train-a-tensorflow2-object-detection-model/).

A continuación, se describen los requerimientos que el modelo debe cumplir en este capítulo como un análisis preliminar antes de empezar con el proceso descrito de la metodología KDD en el capítulo anterior.

2.1 Análisis de requerimientos

TABLA 1. Requerimientos del desarrollo

TABLA DE REQUERIMIENTOS		
Código	**Nombre**	**Prioridad**
RF-001	Redimensión de imagen	Alta
RF-002	Aumentación del dataset	Alta
RF-003	Detección de meteoros	Alta
RF-004	Validar el modelo	Alta
RNF-001	Arquitectura tecnológica	Alta

Fuente: Propia

2.1.1 Requerimientos funcionales

TABLA 2. Dimensión de imagen.

Código	**Nombre**	**Prioridad**
RF-001	Redimensión de imagen	Alta
Descripción	La arquitectura del modelo seleccionado tiene como requisito una entrada de imagen de 512x512 píxeles.	

Proceso	Las imágenes obtenidas de la empresa AllSkyCams se suben a la plataforma cloud Roboflow. En el proceso de redimensionamiento se le da el tamaño de 512x512 y se selecciona la opción de completado de relleno para no afectar a las características de las imágenes originales.
Notas	

Fuente: Propia

TABLA 3. Aumentación del dataset.

Código	**Nombre**	**Prioridad**
RF-002	Aumentación del dataset	Alta
Descripción	Para una mejor sensibilidad ante varios escenarios, se debe aplicar una aumentación de las imágenes.	
Proceso	Las imágenes sufrirán una transformación. A partir de las imágenes originales, se realizará una aumentación del dataset. Con rotaciones de 90 grados. Rotación aleatoria de entre -45 y 45 grados y un giro horizontal y vertical.	
Notas		

Fuente: Propia

TABLA 4. Detección de meteoros.

Código	**Nombre**	**Prioridad**
RF-003	Detección de meteoros	Alta
Descripción	El modelo entrenado debe obtener al menos un 85% de precisión al validar con la partición del dataset para la fase de test.	
Proceso	Se exporta el modelo en forma de grafo en formato (.ckpt) y (.pb). Se le pasa como parámetro al script que lo pondrá a prueba tanto en el problema de clasificación como en el problema de regresión.	
Notas		

Fuente: Propia

TABLA 5. Validar el modelo.

Código	Nombre	Prioridad
RF-004	Validar el modelo	Alta
Descripción	El modelo entrenado será validado mediante un conjunto de datos de meteoros y no meteoros.	
Proceso	Primero obteniendo los resultados para los meteoros y posteriormente evaluando con imágenes de casas, cielo nocturno, luna, alumbrado público, aviones y nubes se pondrá a prueba el modelo, debiendo tener baja predicción de meteoros en las imágenes mostradas.	
Notas		

Fuente: Propia

2.1.2 Requerimientos no funcionales

TABLA 6. Arquitectura tecnológica

Código	Nombre	Prioridad
RNF-001	Arquitectura tecnológica	Alta
Descripción	Definir la arquitectura del modelo pre entrenado base para el entrenamiento con la API de detección de objetos de Tensorflow 2. Además de definir y contratar tecnologías que facilitarán el trabajo (Roboflow y Google Colab).	
Proceso	El tratamiento de las imágenes se realizará en la herramienta cloud de Roboflow. El modelo será entrenado y validado con la API de detección de objetos de Tensorflow 2 utilizando la arquitectura del modelo pre entrenado Efficient Det D0 en la plataforma cloud de Google Colab.	
Notas		

Fuente: Propia

2.2 Diseño

Los pasos para preparar adecuadamente el dataset y entrenar con éxito el modelo fueron sacados principalmente del tutorial por Roboflow https://blog.roboflow.com/train-a-tensorflow2-object-detection-model/

El repositorio que contiene el modelo efficientdet_d0_coco17_tpu-32 que será la arquitectura tomada para entrenar el modelo se encuentra en el ya mencionado repositorio de Tensorflow 2 Detection Model Zoo http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d0_coco17_tpu-32.tar.gz.

Se ha elegido el efficientdet_d0_coco17_tpu-32 que tiene una velocidad de respuesta de 39 milisegundos y una precisión según la evaluación mean Average Precision (mAP) con el dataset COCO 2017 de 33,6%, debido a que es moderadamente liviano y con una precisión alta. Permitiendo un despliegue más natural en el AllSkyCams que es un sistema que aún tiene un cierto grado de retraso (delay).

Sobre esta arquitectura fue entrenado el modelo, usando una TPU con 37,8GB de memoria RAM proporcionada por Google Colab en su versión PRO + con las imágenes recopiladas de meteoros.

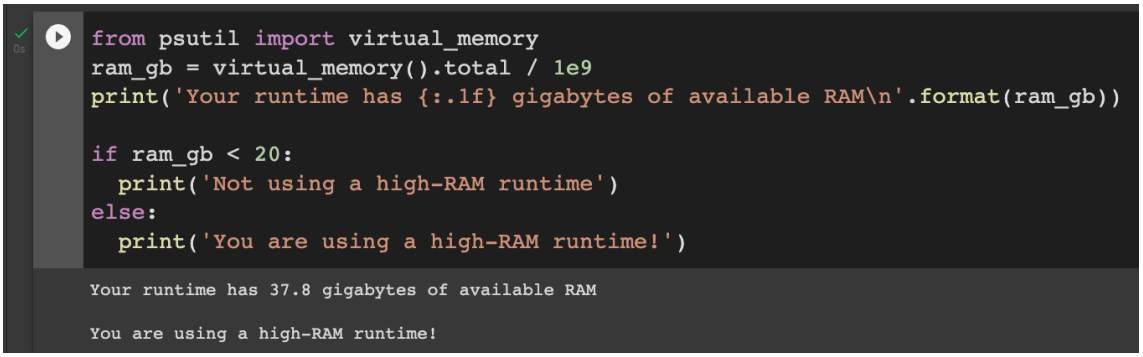


Fig. 13. Consulta de memoria RAM en Google Colab.
Fuente: (Google Colab, 2022)

```
import tensorflow as tf
try:
  resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(resolver)
  tf.tpu.experimental.initialize_tpu_system(resolver)
  print("All devices: ", tf.config.list_logical_devices('TPU'))
  strategy = tf.distribute.experimental.TPUStrategy(resolver)
except ValueError:
  strategy = tf.distribute.get_strategy()
```

```
INFO:tensorflow:Deallocate tpu buffers before initializing tpu system.
INFO:tensorflow:Deallocate tpu buffers before initializing tpu system.
INFO:tensorflow:Initializing the TPU system: grpc://10.110.118.202:8470
INFO:tensorflow:Initializing the TPU system: grpc://10.110.118.202:8470
INFO:tensorflow:Finished initializing TPU system.
INFO:tensorflow:Finished initializing TPU system.
WARNING:absl:`tf.distribute.experimental.TPUStrategy` is deprecated, please use  the non
All devices:  [LogicalDevice(name='/job:worker/replica:0/task:0/device:TPU:0', device_typ
INFO:tensorflow:Found TPU system:
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
```

Fig. 14. Consulta de tarjeta gráfica en Google Colab.

Fuente: (Google Colab, 2022)

Para (Riccio et al., 2020) los modelos de red neuronal convolucional profunda comprenden niveles de abstracción y acceso. En el esquema siguiente se presenta su propuesta y con ello contextualizar qué factores serán los que dependen del investigador.

Empezando por la White-box o caja accesible, donde el investigador puede comprender y ajustar a merced la arquitectura, hiperparámetros y funciones de activación con el fin de obtener los resultados deseados. El siguiente nivel es el nivel data-box donde aún es accesible y se hace una división del dataset pensando en la fase de entrenamiento y test. Por último, el nivel black-box donde se encuentran todos los cálculos, pesos y matrices multidimensionales que comprenden el modelo entrenado. Es en los dos primeros niveles donde el investigador juega un papel crucial, ya que, de su previo estudio del estado del arte, le permitirá seleccionar los hiperparámetros adecuados al objeto de estudio y detección.

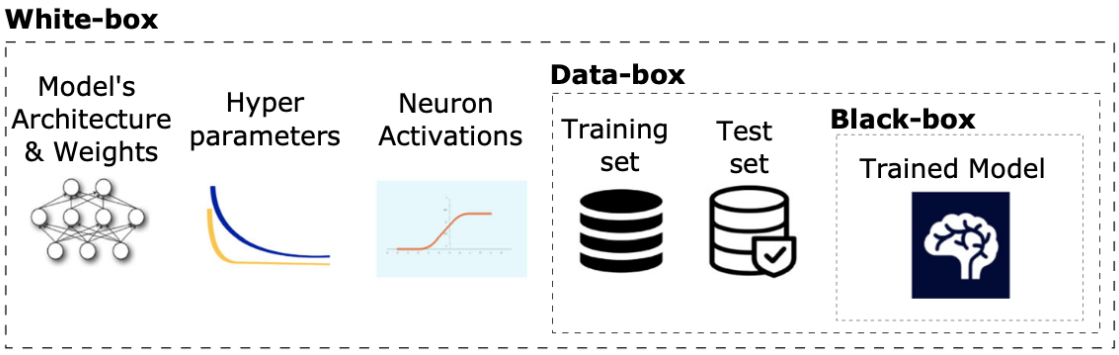


Fig. 15. Niveles de abstracción en el entrenamiento de un modelo de Deep learning.
Fuente: (Riccio et al., 2020)

Basado en el diagrama anterior se expone cada componente crucial para conseguir el entrenamiento de una red neuronal profunda para detección de meteoros con éxito.

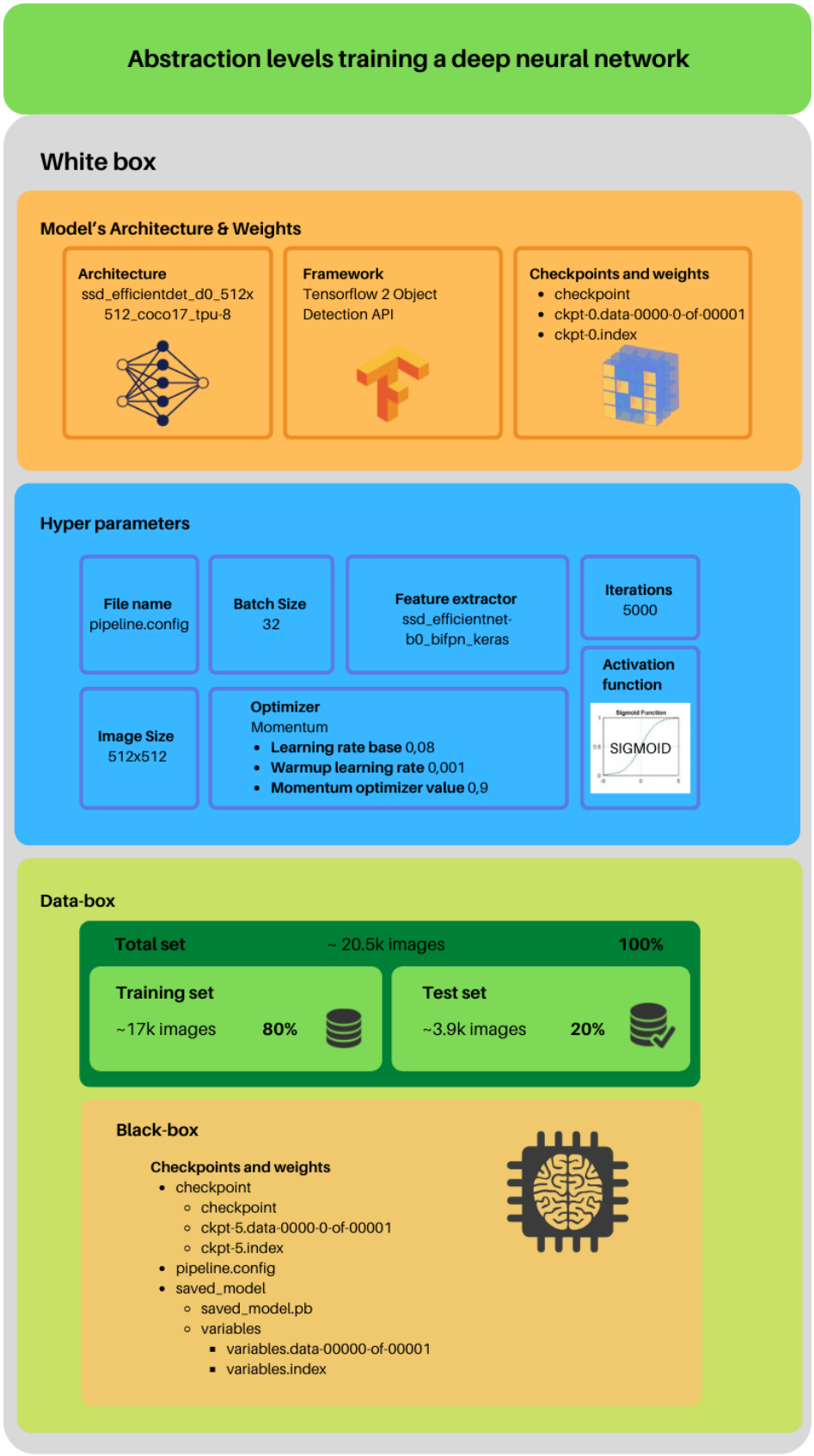


Fig. 16. Niveles de abstracción en el entrenamiento del AllSkyNet.

Fuente: Propia

Como se muestra en la literatura, el aprendizaje profundo generalmente se beneficia del entrenamiento con cantidades muy grandes de datos denominado técnicamente como big data (Dong et al., 2016). Es por esta razón que se ha tratado de abastecer al dataset con la mayor cantidad de ejemplares originales y hacer moderadamente aumentación. Son 9409 imágenes de meteoros recuperados por la unidad en funcionamiento AllSkyCams específicamente la estación AMS1. Se experimentó entrenar con diferentes tamaños 320, 512, 640 y deferentes arquitecturas ya que se puede aplicar diferentes tamaños gracias a su estructura SSD. Pero se concluyó que, su tamaño ideal para tener un balance entre precisión y rapidez puede ser cualquiera de los 3 analizados. Precisamente el tamaño de entrada que requiere la arquitectura ssd_efficientdet_d0_512x512_coco17_tpu-8 es 512. Por esto se procedió a redimensionar las imágenes a un tamaño de 512 x 512 píxeles sin alterar sus dimensiones, utilizando una función de redimensión reflejo en Roboflow que se explicará más adelante. Las imágenes de entrenamiento son un 80% del total equivalente a ~ 17 mil y las de prueba un 20% equivalente a ~ 3.9 mil debido a la aumentación que se extenderá más adelante.

El algoritmo de la CNN en acción para (Horak & Sablatnig, 2019) se puede enumerar en estos cuatro puntos:

- La imagen de entrada se preprocesa (cortes, redimensión).
- En cada imagen se aplica un clasificador CNN que calcula la puntuación de confianza para cada categoría definida.
- Las etiquetas clasificadas se almacenan solo si la puntuación de confianza es superior al umbral predefinido.
- Se dibujan rectángulos (bounding box) alrededor de los objetos con las puntuaciones de confianza más altas.

El número de imágenes de entrenamiento se incrementó artificialmente mediante el uso de transformaciones manteniendo las mismas etiquetas de las imágenes. Gracias a la herramienta RoboFlow en cloud, se aceleró este proceso considerablemente.

2.3 Metodología KDD

2.3.1 Recolección del dataset

Gracias a AllSkyCams, se pudo recolectar una cantidad decente de ejemplares para el entrenamiento y se eliminó cualquier imagen basura que interfiera en este proceso.

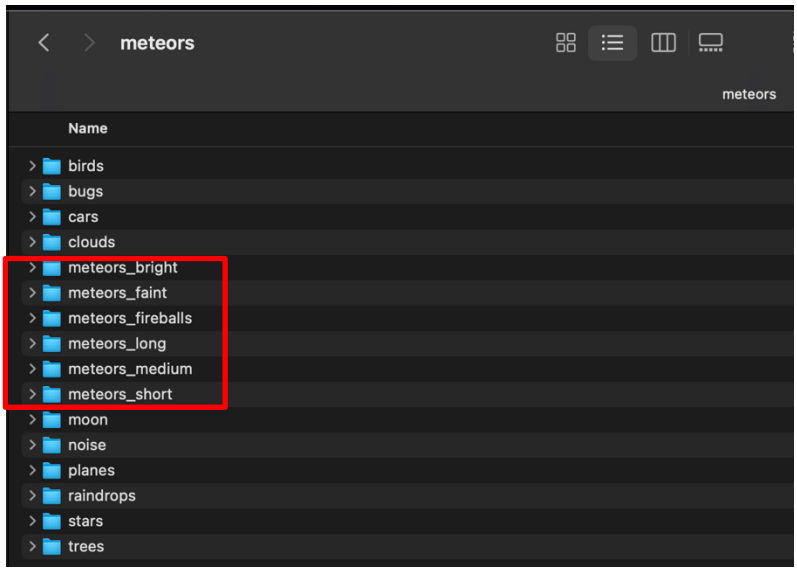


Fig. 17. Dataset de entrenamiento en crudo.

Fuente: Propia

Se procede a crear un proyecto y subir todas las imágenes a Roboflow para seguir todo el proceso de Selección, Preprocesamiento y Transformación allí.

Crear un proyecto en Roboflow es sencillo.

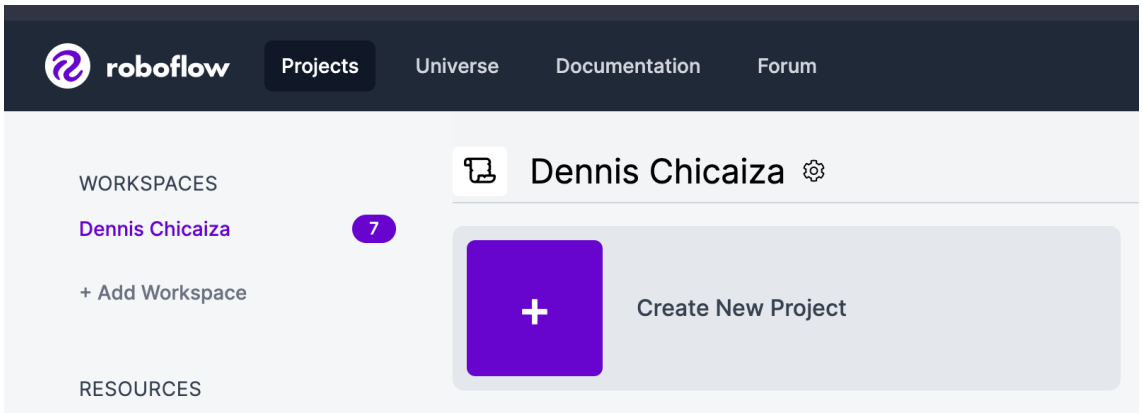


Fig. 18. Dataset de entrenamiento en crudo.

Fuente: (Roboflow, 2022)

Ahora se debe especificar el nombre, el tipo de proyecto y la etiqueta de predicción. Si se tiene más de un objeto, luego al crear los bounding box se puede crear más de una clase de predicción.

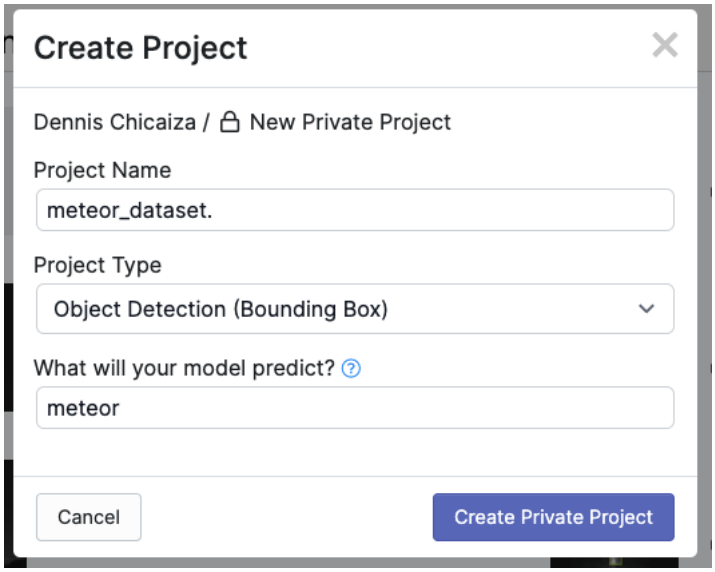


Fig. 19. Dataset de entrenamiento en crudo.

Fuente: (Roboflow, 2022)

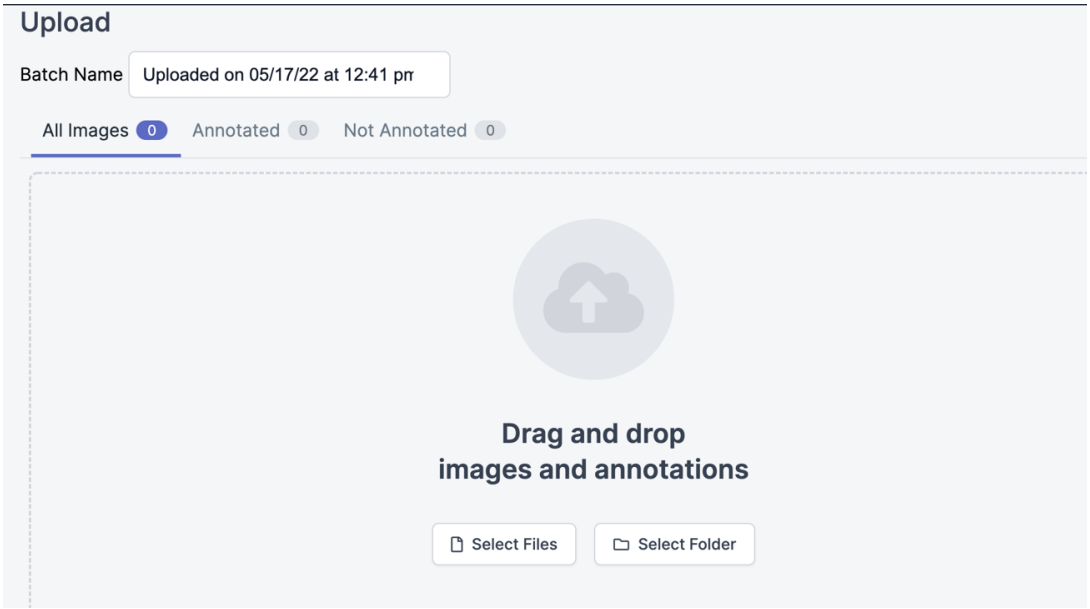


Fig. 20. Dataset de entrenamiento en crudo.

Fuente: (Roboflow, 2022)

2.3.2 Selección

Al subir las imágenes se puede crear tareas en un tablero Kanban para organizar la selección y etiquetado de las imágenes. A continuación, se muestra cómo en la fase de selección, de entre 14 mil ejemplares una tarea con 778 imágenes y una tarea con 497 imágenes fueron las que no pasaron de la fase de selección por sus precarias características y exceso de ruido en las imágenes. El resto fueron eliminadas mientras se iba seleccionando las mejores imágenes.

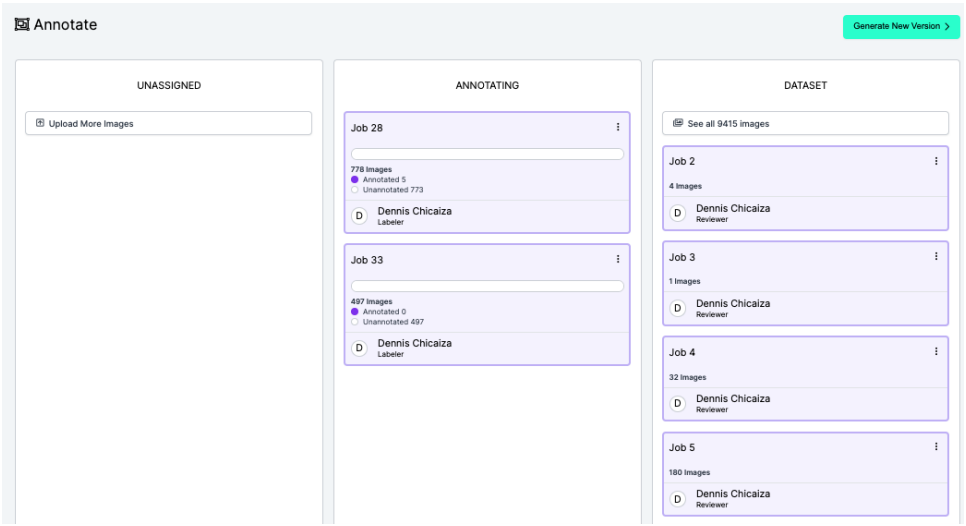


Fig. 21. Tablero Kanban Selección y Etiquetado de las imágenes.

Fuente: (Roboflow, 2022)

A continuación, se etiqueta cada imagen o se elimina del dataset.

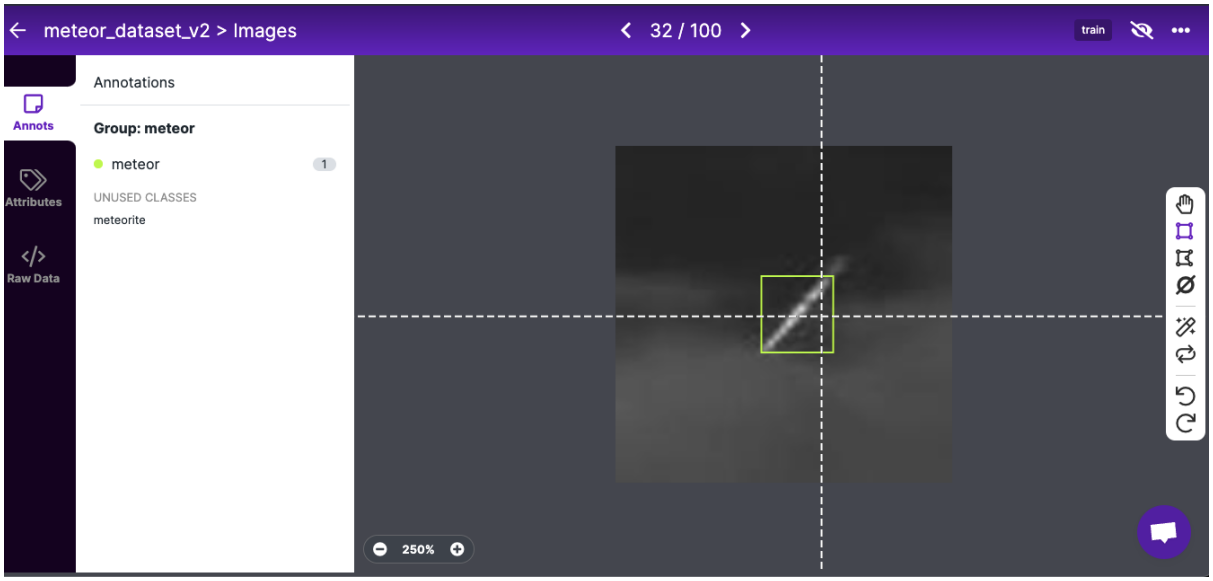


Fig. 22. Etiquetar las imágenes.

Fuente: (Roboflow, 2022)

Al finalizar con todas las imágenes se procede al pre procesamiento.

2.3.3 Pre procesamiento

- Redimensionado: 512x512

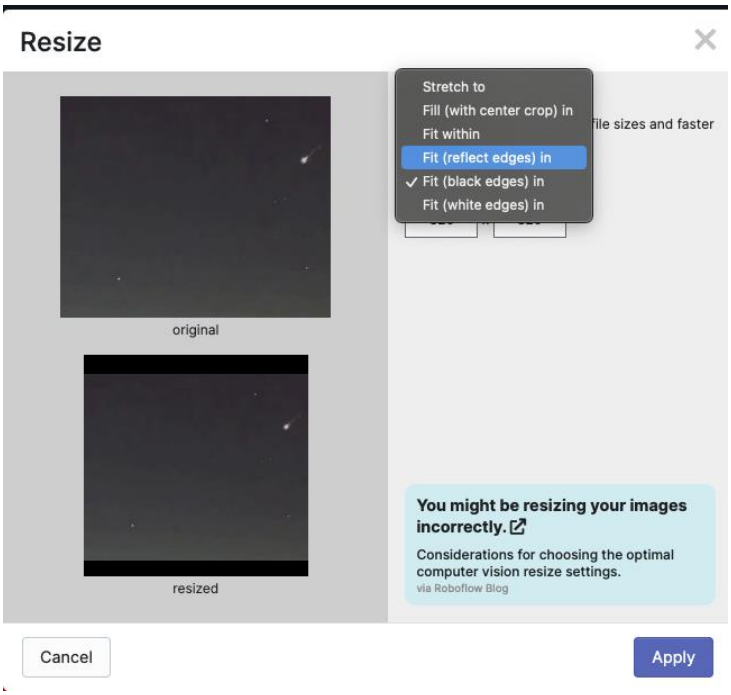


Fig. 23. Propiedad de rellenar imágenes.

Fuente: (Roboflow, 2022)

En este ajuste de dimensión se puede elegir entre algunas opciones como completar, ajustar y el que se ha elegido fue rellenar con bordes reflejo que permiten simular espacio para rellenar la imagen con pixeles existentes.

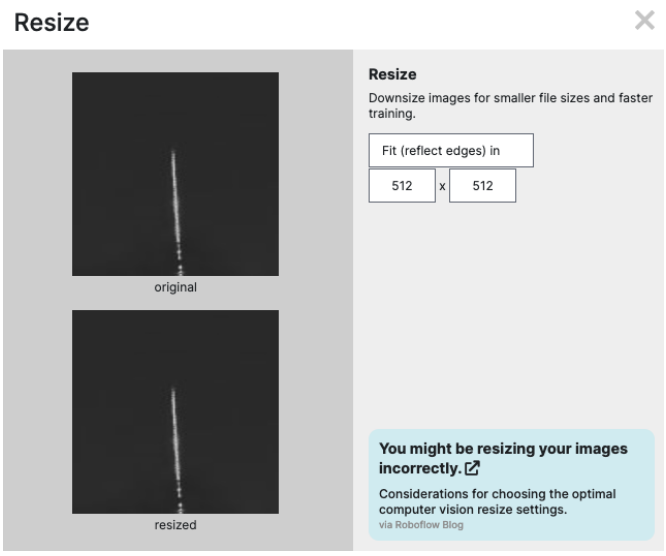


Fig. 24. Redimensionado del dataset.

Fuente: (Roboflow, 2022)

- Aumento:

  Se genera a partir de 1 imagen 2 imágenes más gracias a este proceso.

  - Giro: Giro horizontal y vertical.

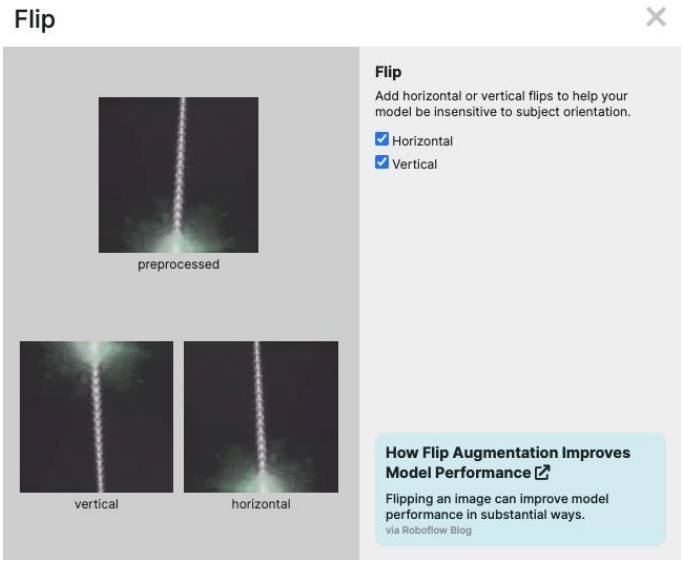


Fig. 25. Giro horizontal y vertical.

Fuente: (Roboflow, 2022)

  - Rotación: de 90° (hacia las manecillas del reloj, contra las manecillas del reloj y en dirección contraria).

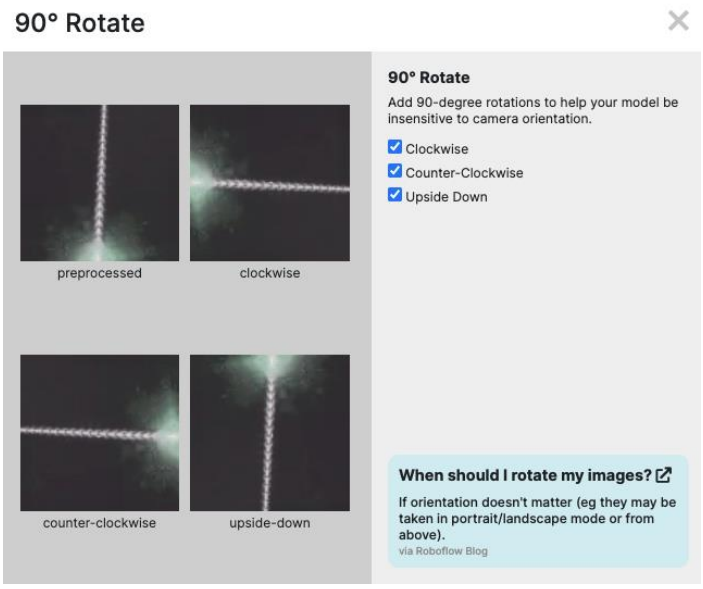


Fig. 26. Rotación de 90 grados.

Fuente: (Roboflow, 2022)

- Rotación aleatoria entre -45° y +45°.

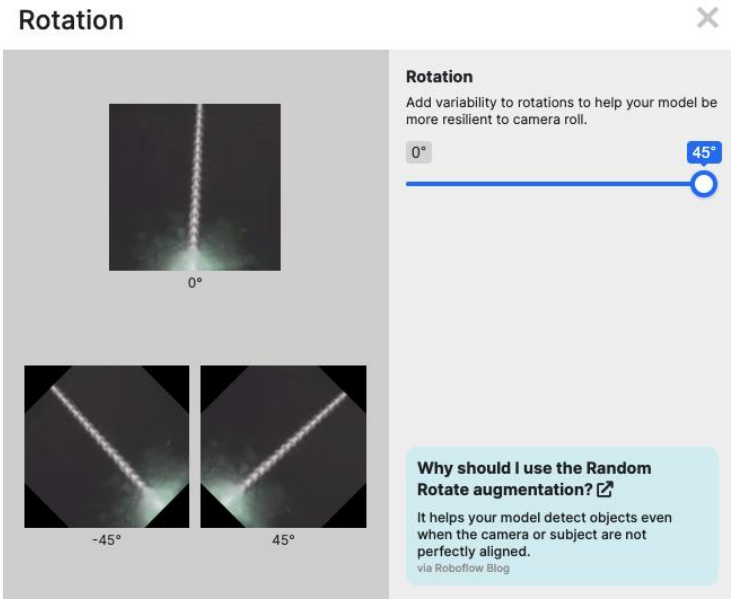


Fig. 27. Rotación aleatoria entre –45 y 45 grados.

Fuente: (Roboflow, 2022)

Gracias a este proceso de aumentación se consiguió un total de 20507 imágenes. 16656 para el entrenamiento y 3853 para la validación posterior del modelo. A estos 3853 luego se les

sumará los verdaderos negativos que pondrán a prueba la calidad del modelo frente a la principal problemática (aviones, pájaros, alumbrado público, entre otros).

2.3.4 Transformación

Para una carga del dataset más adecuada y rápida Tensorflow proporciona su propio formato de archivo multidimensional que contienen tanto las imágenes como las etiquetas y bounding box del ground truth.

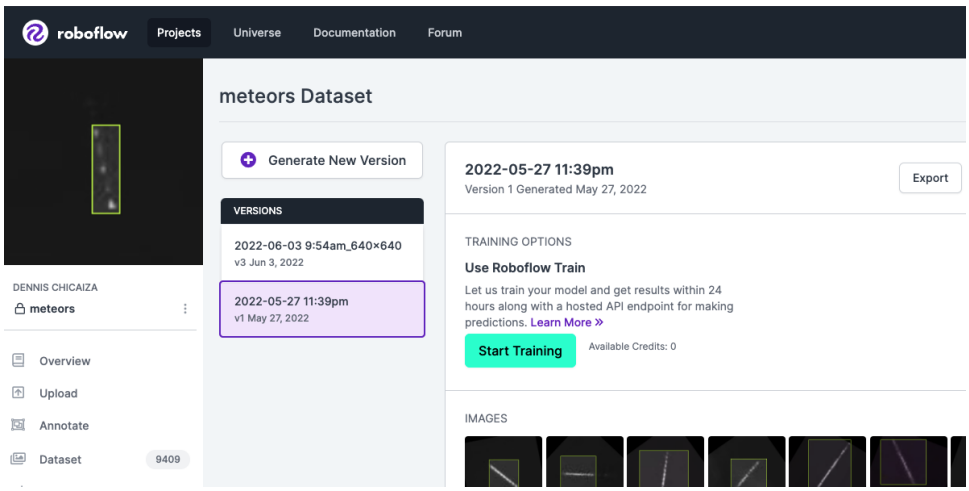


Fig. 28. Versión generada del dataset con la selección y aumentación definida.

Fuente: (Roboflow, 2022)

Al terminar con el preprocesamiento se genera una nueva versión del dataset con Roboflow y este permite exportar en todos los formatos siguientes:

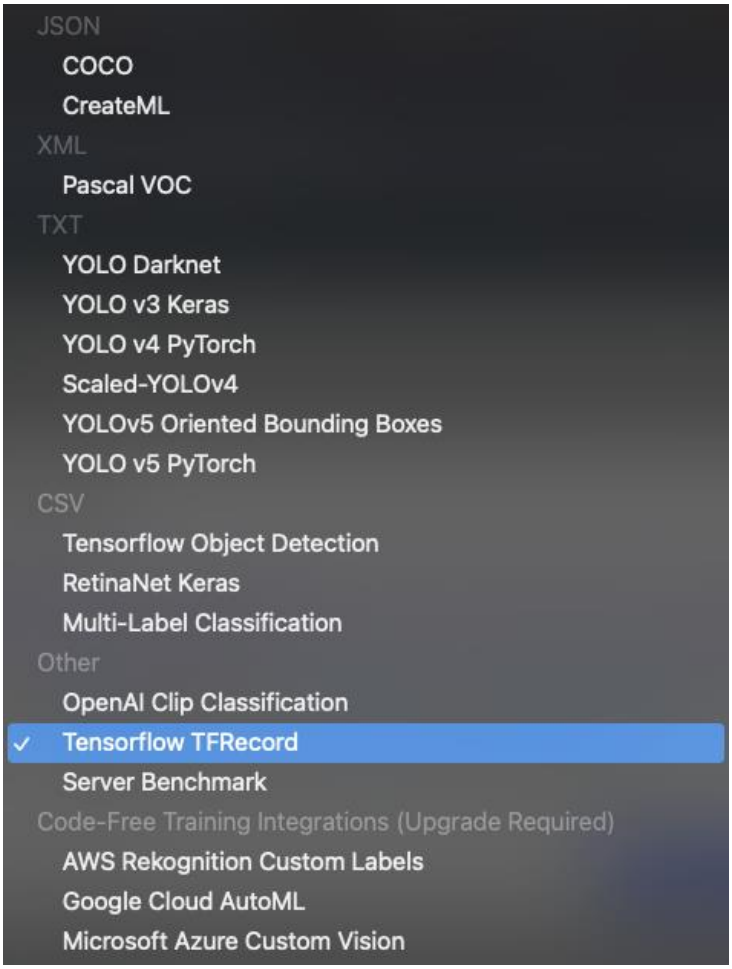


Fig. 29. Formatos en los que Roboflow permite hacer la Transformación del dataset.

Fuente: (Roboflow, 2022)

2.3.5 Configuración de hiperparámetros

Antes de empezar con el entrenamiento la definición de qué hiperparámetros se usarán es esencial para conseguir un modelo óptimo ante la problemática a resolver. A continuación, se describe cada uno de los hiperparámetros que fueron modificados para obtener los resultados esperados.

- **Funciones de pérdida (Loss Functions)**

Las funciones de pérdida juegan un papel central en los modelos basados en el aprendizaje profundo y su elección y definición adecuadas pueden ser clave para un buen desempeño (Luís & Medeiros, 2019).

En el proceso de entrenamiento de redes neuronales**,** la función de pérdida es el estándar de evaluación de todo el modelo de red. No solo representa el estado actual de los parámetros de la red, sino que también proporciona el gradiente de los parámetros en el método de descenso de gradiente, por lo que la función de pérdida es una parte importante del entrenamiento de aprendizaje profundo (Xin & Wang, 2019).

- **Archivo de configuración (Config file)**

Corresponde al archivo de configuración de la arquitectura e hiperparámetros que serán usados a lo largo del entrenamiento del modelo.

- **Tamaño del lote (Batch size)**

El batch size o lote es un valor importante a la hora de entrenar modelos de redes neuronales profundas. Se define la cantidad de imágenes que el modelo verá en cada iteración. Si el lote es demasiado pequeño el modelo será incapaz de generalizar correctamente y por ende no podrá encontrar mínimos globales para los pesos finales del modelo. Corre el riesgo de underfitting que ocurre cuando los valores tanto de entrenamiento como de validación son precarios.

En cuanto que, si el lote es muy grande, tardará demasiado en entrenarse el modelo, además de poder enfrentarse a un overfitting que ocurre cuando el modelo aprende de memoria los valores sin poder generalizar adecuadamente. Ocurre cuando los valores de entrenamiento son buenos pero los de validación son bajos.

El tamaño del lote también depende del tamaño del dataset. Por lo que leer varios casos de estudio en el capítulo anterior fue crucial para identificar un tamaño ideal para el entrenamiento.

- **Extractor de características (Feature extractor)**

El extractor de características es una propiedad dentro de la arquitectura del modelo que identificará las características y patrones existentes dentro del dataset.

- **Iteraciones (Iterations)**

Las iteraciones son la cantidad de batches o steps. En cada iteración el modelo ve cierta cantidad de lotes hasta completar una vista completa de todos los datos denominado época.

- **Tamaño de las imágenes (Image size)**

El tamaño de las imágenes es crucial para obtener buenos resultados. Cuando se entrena un modelo en una arquitectura existente el tamaño de las imágenes debe corresponder al tamaño de entrada que el modelo requiere. Usualmente en Tensorflow no es necesario redimensionarlas antes del entrenamiento, pero si se aconseja que todas tengan un tamaño igual o mayor al tamaño de entrada a la arquitectura.

- **Función de optimización (Optimization function)**

La función de optimización es la función que actualiza los pesos al final de cada iteración antes de hacerse un back propagation. Existen muchas funciones de activación como Adam, SGD, Momentum. Se puede acceder a ejemplos de cada optimizador en el repositorio de modelos de Tensorflow en el directorio siguiente *Tensorflow/models/research/object_detection/builders/optimizer_builder_tf2_test.py*.

Se ha elegido la función de activación Momentum ante Adam ya que es una versión mejorada de Adam y RMSProp. Tiene la capacidad de ambos con un adicional, el valor del optimizador se va ajustando en cada iteración, encontrando mucho más rápido los mínimos globales. Al ajustarse en cada iteración se evita enormemente el estancamiento en mínimos locales que son los valores óptimos en una sola iteración, más no del dataset completo.

- **Función de activación (Activation function)**

La función de activación de la última capa es crucial dependiendo del modelo que se quiera desarrollar. Si se tiene una sola clase a entrenar y detectar, la función de activación *sigmod* es la correcta, que devuelve valores entre 0 y 1. En cuanto que, si se quiere

entrenar y detectar 2 o más clases, la función *softmax* es la idónea devolviendo valores entre –1 y 1.

Gracias a los estudios revisados al final del capítulo anterior, se pudo definir los hiperparámetros que se utilizarán en el entrenamiento del modelo y se enlistan a continuación:

Se exponen los hiperparámetros utilizados para el entrenamiento del modelo.

Fig. 30. Hiperparámetros del entrenamiento.

Fuente: Propia

2.3.6 Entrenamiento

Ahora que los hiperparámetros han sido seleccionados, se procede a entrenar el modelo en cloud. La herramienta de Google Colab es muy versátil a la hora de crear entornos de entrenamiento ya que puedes crear y eliminar estos entornos con mucha facilidad sin poner en riesgo tu sistema operativo o el proyecto en desarrollo.

Se ha contratado la versión de Google Colab Pro desde el 04 de marzo de 2022 hasta el 14 de mayo de 2022. Luego al haber elegido la arquitectura de Efficient Det, se amplió a la versión de Google Colab Pro + con la intención de acelerar el proceso de entrenamiento y utilizar los mejores recursos de este servicio hasta el 29 de mayo de 2022. A esto se le debe añadir que la arquitectura EfficientDet requería ser entrenada en una máquina de computación denominada TPU preferiblemente.

Una TPU es un procesador especializado para hacer cálculos con tensores de Tensorflow por lo que es beneficiosa para ciertas arquitecturas que fueron diseñadas propiamente por Google. Debido a los altos tiempos de respuesta en el entrenamiento de modelos EfficientDet en GPU, se utilizó esta alternativa y con resultados excepcionales como se comenta en (Jouppi et al., 2017).

La TPU, es de 15 a 30 veces más rápida en procesamiento de los cálculos que una GPU K80 o una CPU Haswell (Intel) y como menciona (Jouppi et al., 2017) la carga máxima de estas al procesar redes neuronales convolucionales no pasa del 5% en muchos de los centros de datos que ya apuestan por esta tecnología.

Para el entrenamiento se utilizaron las arquitecturas del repositorio oficial de Tensorflow y el notebook que Roboflow pone a disposición de la comunidad científica:

- https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md
- https://colab.research.google.com/drive/1sLqFKVV94wm-lglFq_0kGo2ciM0kecWD?usp=sharing

Una comparación entre entrenar con los optimizadores Adam y Momentum fue realizada para discernir cual es la más adecuada para obtener los resultados esperados. A continuación, se expone la gráfica de entrenamiento usando el optimizador ADAM y Momentum respectivamente en un entrenamiento demo de 30 iteraciones.

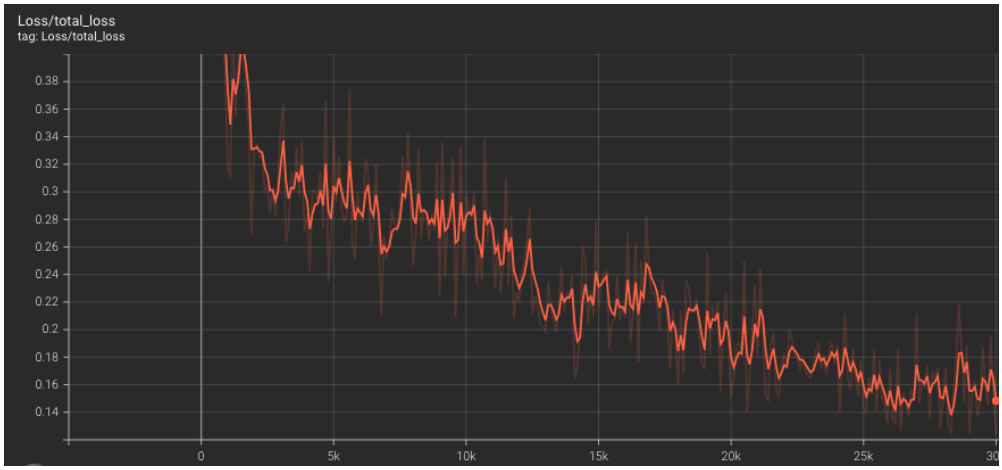


Fig. 43. Entrenamiento con optimizador Adam.

Fuente: Propia

Como se puede observar el optimizador Adam tiene demasiados picos entre iteraciones, lo que se traduce en estancamientos en mínimos locales. Se infiere que el tipo de dataset no se ajusta para ser usado con el optimizador Adam.

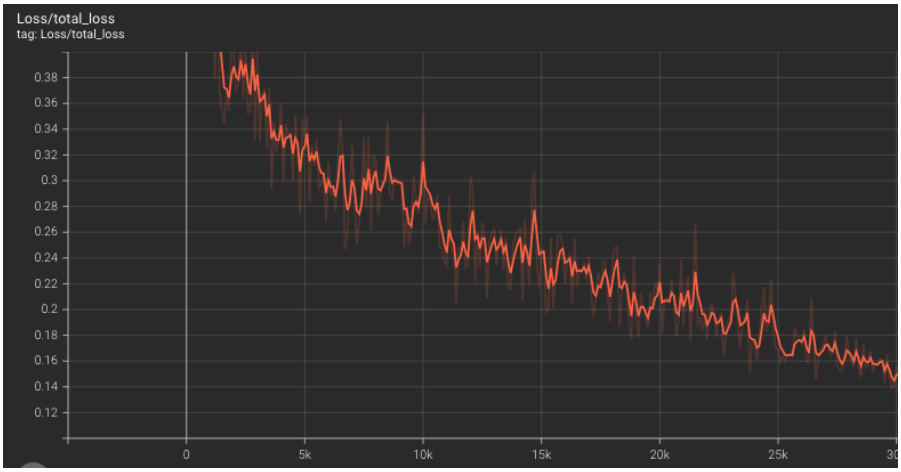


Fig. 44. Entrenamiento con optimizador Momentum.

Fuente: Propia

Se decidió usar Momentum ante Adam ya que es capaz de actualizar el learning rate, los pesos y sesgos en cada iteración suavemente, algo muy beneficioso para encontrar los mínimos óptimos globales.

El tiempo que tomó entrenar el modelo a 5 mil iteraciones fue de 20 horas y 30 minutos. La API de detección de objetos de Tensorflow 2, guarda cada mil iteraciones un checkpoint con el que se puede hacer predicciones. Se usaron cada uno de los checkpoint para evaluar el modelo cada vez que fue guardado y así analizar la mejora del modelo al final.

Cuando se tiene el checkpoint que más se ajusta a los resultados esperados, se procede a exportar ese checkpoint en forma de grafo. El script usado se encuentra en el mismo repositorio de la API de detección de objetos de Tensorflow 2 denominado ***exporter_main_v2.py*** con el que se podrá proceder a la implementación. La exportación consiste en fusionar capas para optimizar el modelo en el despliegue ya que haría varios cálculos de tensores a la vez. En este punto se elimina cierta información que sólo en el entrenamiento era relevante como los gradientes en cada checkpoint, que permiten continuar entrenando el modelo desde un checkpoint en específico. El modelo final se encuentra en formato .pb (protobuf).

2.3.7 Despliegue

El framework Flask es un marco de trabajo para desarrollo web con Python. Se ha elegido esta herramienta debido a que el actual software de la AllSkyCams está desarrollado principalmente en Python con Flask.

A continuación, se expone el despliegue en un apartado del actual sistema dentro de Learning > AllSkyNet, dónde el modelo será accesible para detección de las imágenes que la AllSkyCams vaya captando y almacenando.

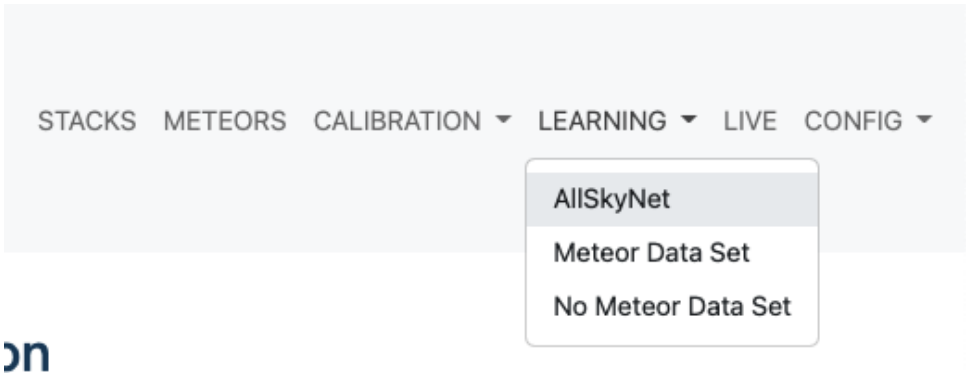


Fig. 31. AllSkyNet implementado en la barra de navegación del AllSkyCams.

Fuente: Propia

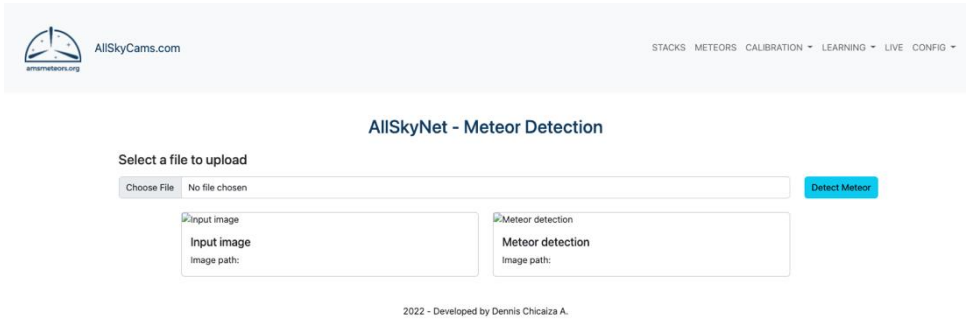


Fig. 32. AllSkyNet implementado en el sistema AllSkyCams (Ambiente de desarrollo).

Fuente: Propia

Se debe elegir una imagen a detectar. Cabe recalcar que las imágenes deben encontrarse en un directorio predefinido para que no haya inconvenientes. Al pasar a producción el modelo, la configuración del directorio donde se irán almacenando temporalmente las imágenes antes de guardarse en el disco será realizada.

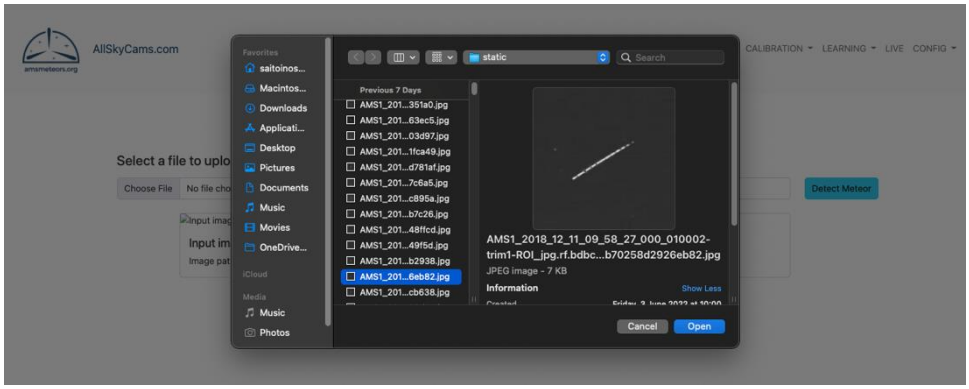


Fig. 33. Selección de una imagen a detectar.

Fuente: Propia

Antes de cargar la imagen de prueba se debe asegurar que estas pertenezcan al conjunto de datos de prueba y no al de entrenamiento si no los resultados serán demasiado optimistas además se debe mencionar que las imágenes de test deben encontrarse redimensionadas a 512x512 píxeles, aunque Roboflow hace esto por nosotros. Teniendo esto en mente, se procede a pulsar el botón Detect Meteor (Detectar Meteoro).

Al hacer esta acción por primera vez, el modelo tarda un poco más en detectar el objeto. Luego, el modelo es tan rápido como se pudo analizar en la tabla de modelos pre entrenados de la documentación de Tensorflow 2 Detection Model Zoo.

A la izquierda aparece la imagen original de entrada y a la izquierda la misma imagen con el bounding box del meteoro detectado, la clase a la que pertenece y el porcentaje de acierto tanto de clasificación como de regresión.

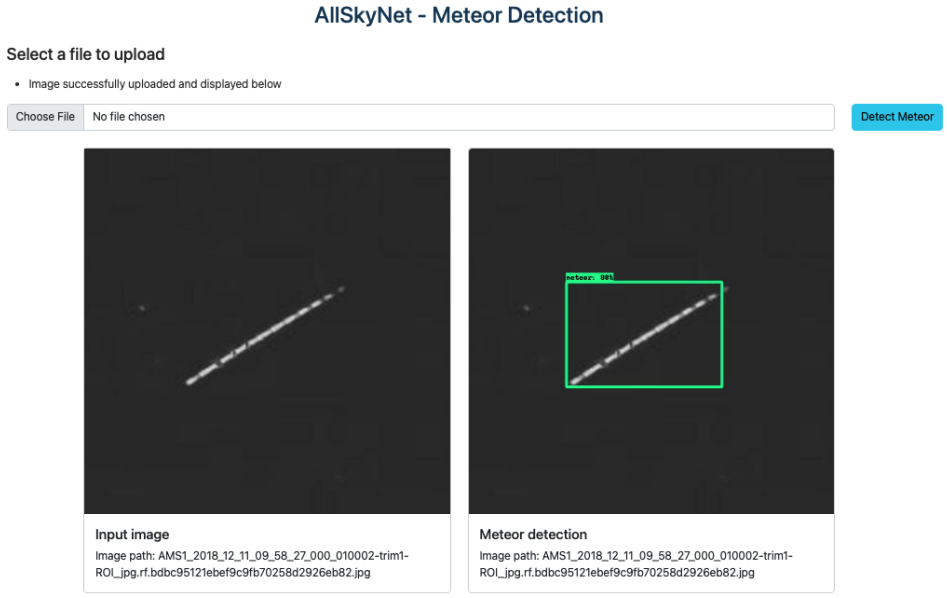


Fig. 34. Meteoro detectado.

Fuente: Propia

El siguiente diagrama expone en qué lugar del procesamiento de imágenes por la AllSkyCams se dispone el modelo AllSkyNet. Antes de guardar permanentemente los datos como meteoros verdaderos. El modelo de detección de meteoros hace un filtro para luego poder seguir con la reducción de meteoros, clasificación y estudio.

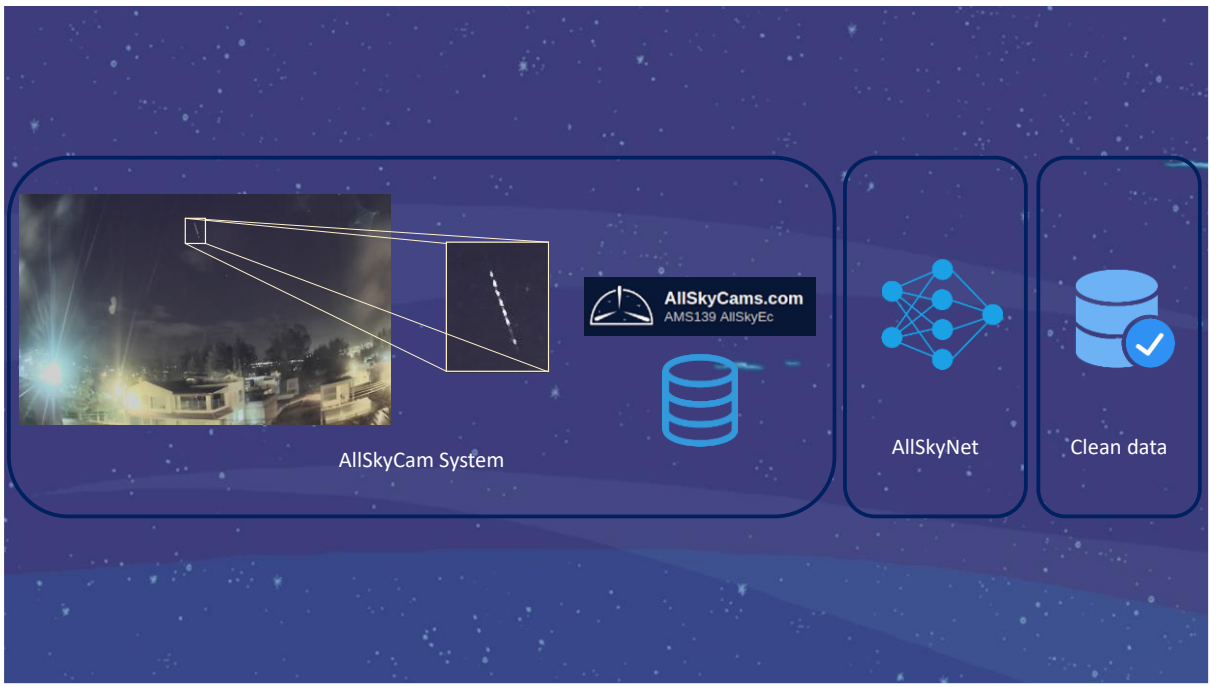


Fig. 35. Nuevo proceso AllSkyCams.

Fuente: Propia

2.3.8 Conocimiento

Al finalizar la implementación el conocimiento se irá generando conforme se utilice el modelo, con ello se podrá refinar y optimizar el modelo cada vez.

CAPÍTULO 3

Resultados

En este último capítulo, se exponen los resultados obtenidos en el entrenamiento y validación del modelo AllSkyNet.

3.2 Métricas de evaluación

Con los artefactos correctos (dataset, arquitectura e hiperparámetros), (Mittal et al., 2020) estima una eficiencia promedio de aproximadamente ~ 93-95% para disciplinas dentro de la visión artificial en el campo de la astronomía. En la presente investigación, gracias al amplio y diverso conjunto de datos se pudo abordar los múltiples casos que puedan presentarse al detectar meteoros y se evalúan a continuación.

Al ser un problema de regresión y clasificación, las métricas de evaluación serán por un lado métricas propias de clasificación como la matriz de confusión, pérdida en Clasificación, Precisión, F1-Score y Recall, y por el otro, métricas propias de Detección de objetos como Pérdida en localización o regresión, Precision/mAP@0.5IoU y Sensibilidad (Recall).

3.2.1 Métricas de Clasificación

- **Matriz de confusión**

Para evaluar el modelo entrenado en términos de clasificación, utilizamos la técnica de Matriz de confusión. La función (Matriz de confusión) calcula la matriz de confusión y devuelve el resultado como un Array que muestra cómo nuestro modelo de clasificación se confunde con qué clase. En el caso del presente proyecto se analizará una matriz de confusión binaria de 2x2. Proporciona información no solo sobre los errores que comete el clasificador, sino también sobre más formas de error diferentes (Karanam et al., 2020).

(Tanoglidis et al., 2021) facilita la comprensión de las 4 posibilidades que existen al evaluar con matriz de confusión:

- Verdadero negativo (TN): cuando no ocurre una anomalía en la escena y el modelo no detecta una.
- Falso positivo (FP): cuando no se produce una anomalía en la escena, pero el modelo la considera como tal.

- Falso negativo (FN): Cuando ocurre una anomalía en la escena, pero el modelo no la detecta.
- Verdadero positivo (TP): cuando ocurre una anomalía en la escena y el modelo realmente la considera como tal.

Se obtuvo la matriz de confusión con un umbral de 0.5 o 50% mediante el script ***confusion_matrix_tf2.py*** obtenido de https://github.com/svpino/tf_object_detection_cm en dónde solamente se tomarán como verdaderos positivos aquellas predicciones que igualen o superen dicho umbral fijado. (Affonso et al., 2017) propone comparar los errores de tendencias de clase basados en la matriz de confusión, como se presenta en la siguiente tabla:

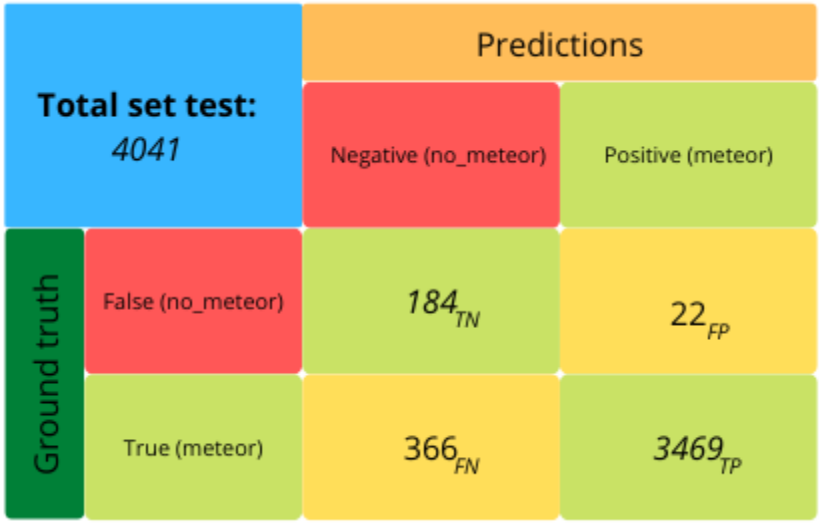


Fig. 36. Matriz de confusión.

Fuente: Adaptado de (Affonso et al., 2017)

Al analizar la matriz de confusión obtenida, podemos observar que el modelo es bastante robusto. Aunque el dataset no tiene un balance apropiado entre la clase entrenada meteoros y no meteoros para validación. Se ha tratado de poner a prueba el modelo con 188 ejemplares de no meteoros de alta complejidad para comprobar que el modelo no tenga un recuerdo (recall) bajo a la hora de encontrarse con imágenes que contengan ya sean aviones, pájaros, la luna, estrellas, alumbrado público entre otros; que eran la principal dificultad que AllSkyCams atraviesa.

De estos 188 ejemplares, el modelo fue capaz de discriminar 184. Solamente fueron 22 los que creía haber encontrado meteoros donde no los había. De un total de 3853 ejemplares de meteoros, fue capaz de detectar con éxito 3469, fallando en 366 de los

cuales predijo que no existía ningún meteoro en imágenes en las que si había. Con esta información, se procede con las demás métricas de evaluación para el problema de clasificación.

- **Precisión**

Con esta métrica se es capaz de medir la calidad del modelo en cuanto al problema de clasificación, aplicando la siguiente formula de precisión obtenida de (Lukic et al., 2020) y los datos de la matriz de confusión. La precisión del modelo es la siguiente:

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

$$Precision = \frac{3469}{3469 + 22}$$

$$Precision = 0{,}993$$

$$Precision = 99{,}36\%$$

- **Recuerdo (Recall)**

El recuerdo es la tasa de meteoros que fue capaz de predecir del total de imágenes. Usando la ecuación que (Lukic et al., 2020) proporciona, el cálculo del recuerdo (Recall) es el siguiente:

$$Recall = \frac{TP}{TP + FN}$$

$$Recall = \frac{3469}{3469 + 366}$$

$$Recall = 0{,}9045$$

$$Recall = 90{,}45\%$$

- **F1 Score**

La puntuación F1 relaciona la tasa de acierto dentro de una imagen (precisión) y la tasa de verdaderos positivos en el conjunto de datos de prueba (recall) (Ordóñez & Roggen, 2016).

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

$$F1\ score = \frac{2 \times 0{,}993 \times 0{,}9045}{0{,}993 + 0{,}9045}$$

$$F1\ score = 0{,}9466$$

$$F1\ score = 94{,}66\%$$

- **Pérdida en Clasificación (Classification Loss)**

Los resultados fueron excelentes para clasificación y regresión. Contra más cerca a cero los resultados son mejores, es por eso por lo que los valores se han considerado suficientes para completar con éxito la implementación del modelo.

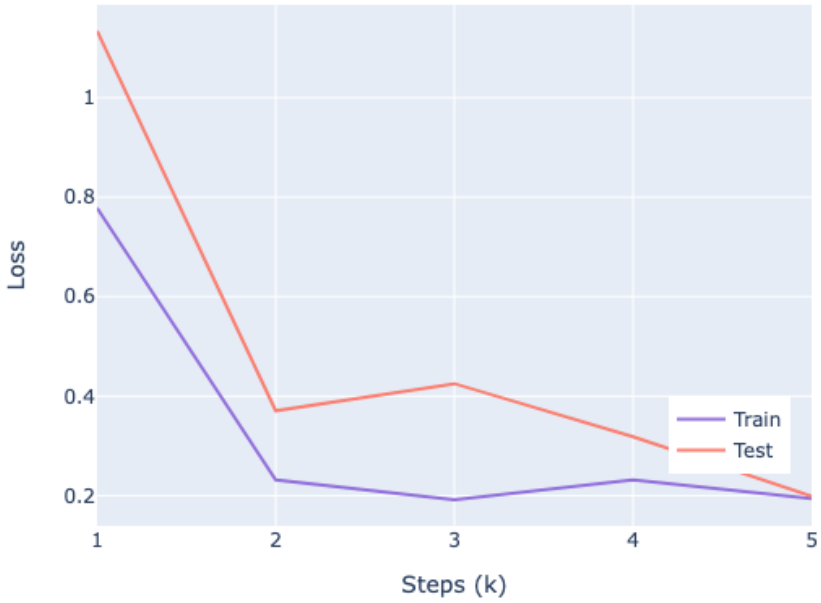


Fig. 37. Pérdida en Clasificación.

Fuente: Propia

A continuación, se describen los valores obtenidos en el entrenamiento y validación de la pérdida del modelo para el problema de Clasificación.

TABLA 7. Pérdida en Clasificación.

Step (k)	Train	Test
1	0,7782	1,1340
2	0,2316	0,3707
3	0,1917	0,4251
4	0,2314	0,3186
5	0,1945	0,1991

- **Curva Precision vs. Recall**

Al analizar qué es lo que AllSkyCams requiere, se pudo concluir que la precisión deberá predominar ante la sensibilidad (recall). Por lo que con un umbral de 0.5 o 50% el modelo obtuvo los resultados que se esperaba.

A continuación, se presenta en el siguiente gráfico la relación que existe entre la precisión y el recuerdo para diferentes umbrales. Cuando se requiere una precisión alta se debe arriesgar sensibilidad, en cuanto se requiere mayor sensibilidad, la precisión se ve reducida. Por lo que estas dos variables son inversamente proporcionales.

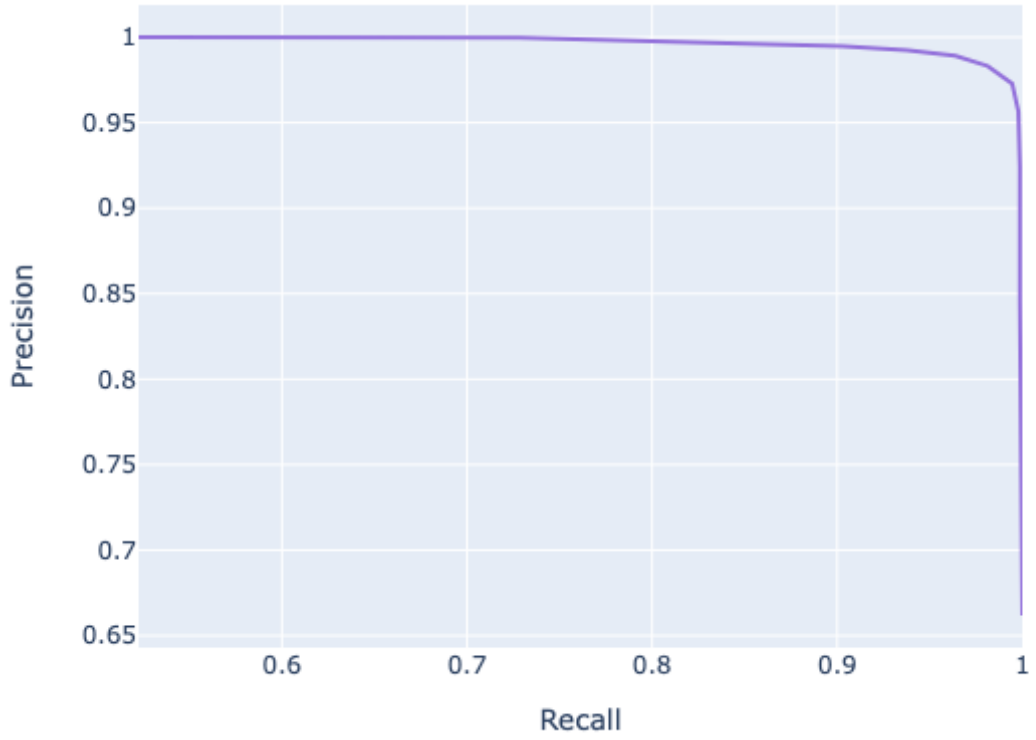


Fig. 38. Curva Precisión vs. Sensibilidad.

Fuente: Propia

TABLA 8. Relación Precisión con Sensibilidad.

Threshold	Precision	Recall
0,75	1,0000	0,5226
0,70	0,9996	0,6466
0,65	0,9996	0,7283
0,60	0,9977	0,7996
0,55	0,9960	0,8570
0,50	**0,9936**	**0,9045**
0,45	0,9923	0,9379
0,40	0,9891	0,9638
0,35	0,9831	0,9813
0,30	0,9729	0,9945
0,25	0,9564	0,9979
0,20	0,9244	0,9987
0,15	0,8468	0,9987
0,10	0,6618	1,0000

3.2.2 Métricas de Regresión

- **Average Precision**

La precisión es una métrica que se utiliza en la evaluación de la Intersección sobre Unión (IoU) para medir la efectividad del modelo. La tasa de precisión (AP) se seleccionó como índice para evaluar la precisión del modelo en términos de detección de meteoros que involucra tanto clasificación como regresión. La Intersection Over Union (IoU) es un indicador importante que refleja la diferencia entre el cuadro de predicción y el cuadro de verdad (ground truth). Cuanto mayor sea el valor de IoU, menor será la diferencia entre ellos (Wang et al., 2021).

En la siguiente figura se exponen los valores obtenidos para la evaluación de la última iteración (step), tomando diferentes umbrales como lo realiza (Shilon et al., 2019) en la evaluación de su modelo. El umbral con el que se evalúa el modelo es de 0.5.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.520
Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.927
Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.546
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.177
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.160
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.549
```

Fig. 38. Resultados mAP métricas para diferentes IoU en el último checkpoint.

Fuente: Propia

En el siguiente gráfico se exponen los valores obtenidos desde la iteración cero a la quinta. Siendo esta última la que se toma como el modelo final optimizado. La validación del modelo se llevó a cabo bajo un dataset de 4041 ejemplares de meteoros y no meteoros. Este dataset es importante mencionar que no está balanceado ya que consta de 3853 imágenes de meteoros y solamente 188 de no meteoros. Los resultados para esta evaluación fueron extraídos de la ejecución de la evaluación que incorpora la API de Detección de Objetos de Tensorflow 2.

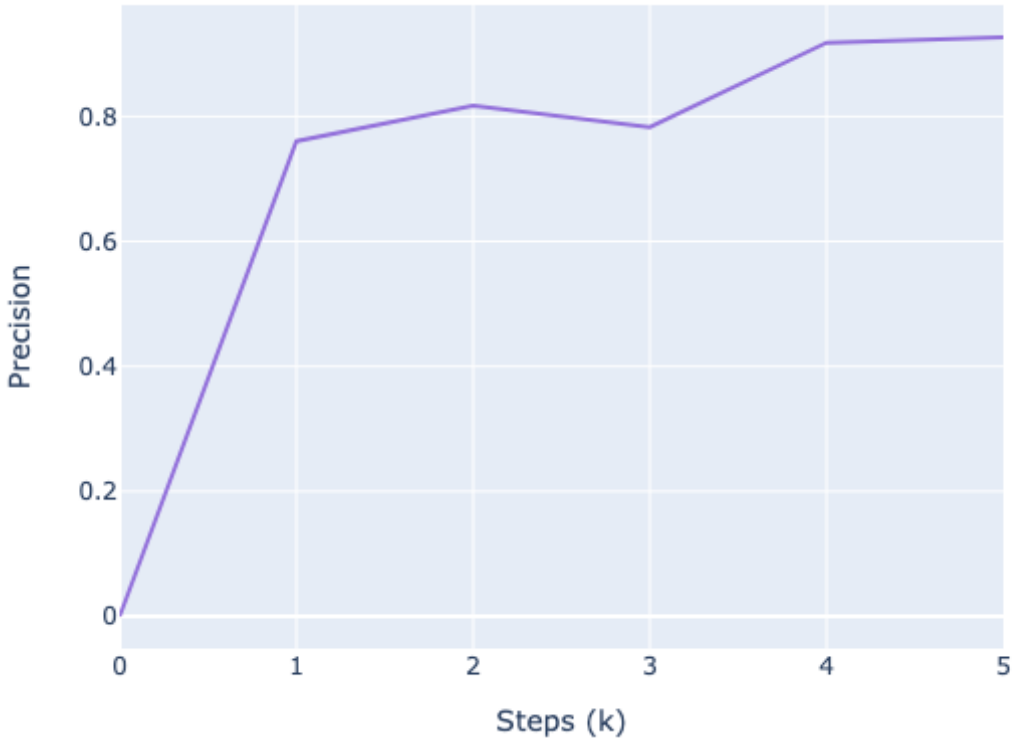


Fig. 40. Crecimiento de la precisión por iteraciones.

Fuente: Propia

TABLA 9. Crecimiento de la precisión por iteraciones.

Steps (k)	Precision
1	0,7603
2	0,8169
3	0,7828
4	0,9181
5	0,9267

- **Pérdida en Localización o Regresión (Localization Loss)**

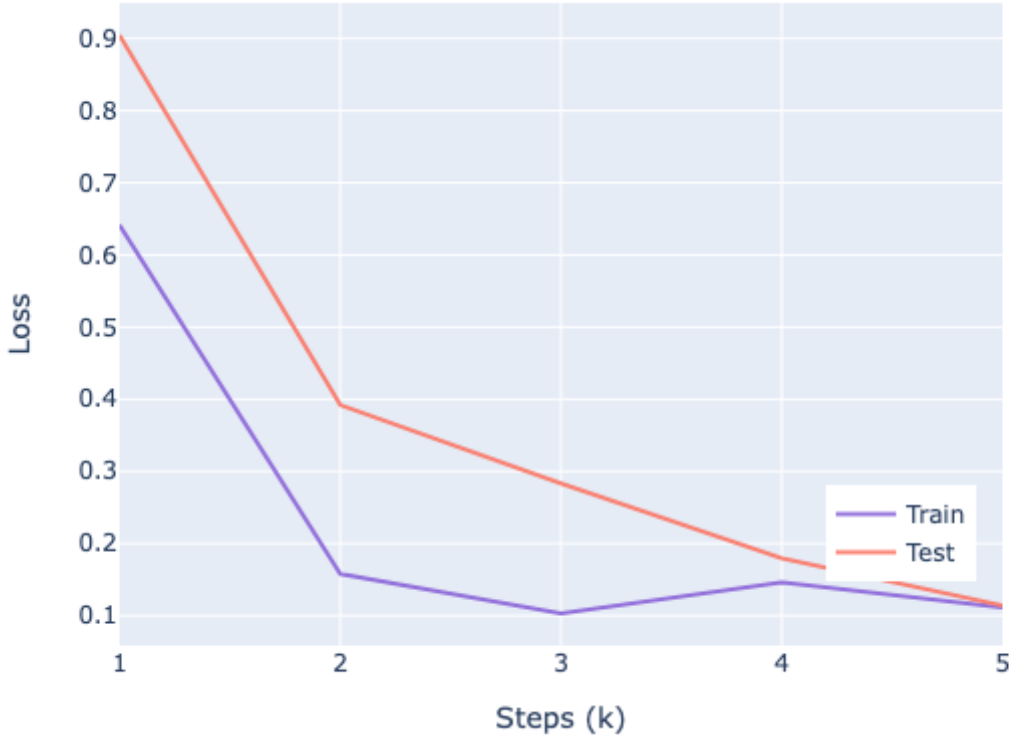


Fig. 41. Pérdida en Localización (Regresión).

Fuente: Propia

TABLA 10. Pérdida en Localización (Regresión).

Step	Train	Test
1k	0,6414	0,9046
2k	0,1580	0,3922
3k	0,1030	0,2834
4k	0,1463	0,1794
5k	0,1113	0,1140

- **Pérdida en Regularización (Regularization Loss)**

La pérdida en Regularización representa la reducción del error de generalización, sin modificar su error de entrenamiento. En la presente investigación se hizo uso de L2 regularizer como parámetro, con un valor de 4e-05. En el proceso de entrenamiento este

valor se ve incrementado hasta encontrar un balance, en cuanto en la validación se ve reducido drásticamente. Previene el overfitting en el entrenamiento.

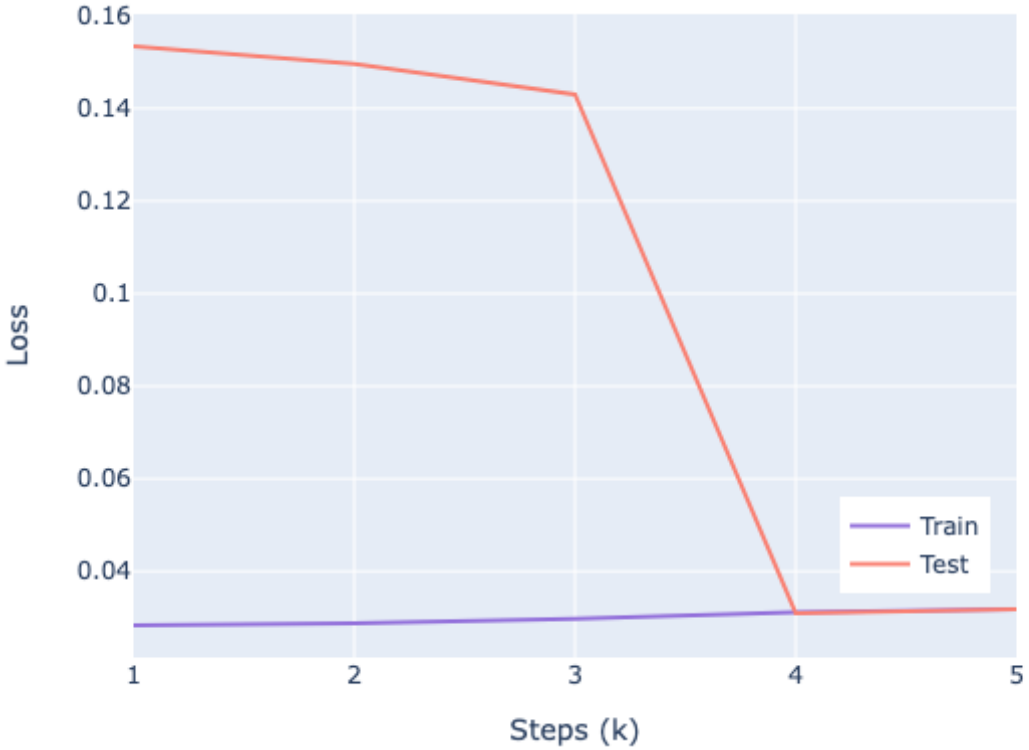


Fig. 42. Pérdida en Regularización.

Fuente: Propia

TABLA 11. Pérdida en Regularización.

Step	Train	Test
1k	0,0283	0,1534
2k	0,0287	0,1496
3k	0,0298	0,1430
4k	0,0311	0,0310
5k	0,0318	0,0318

- **Pérdida Total (Total Loss)**

La pérdida total es la sumatoria de las 3 pérdidas anteriores. De esta manera es como la API de detección de objetos entrega los valores de evaluación del modelo.

$$Total\ loss = Classification\ loss + Localization\ loss + Regularization\ loss$$

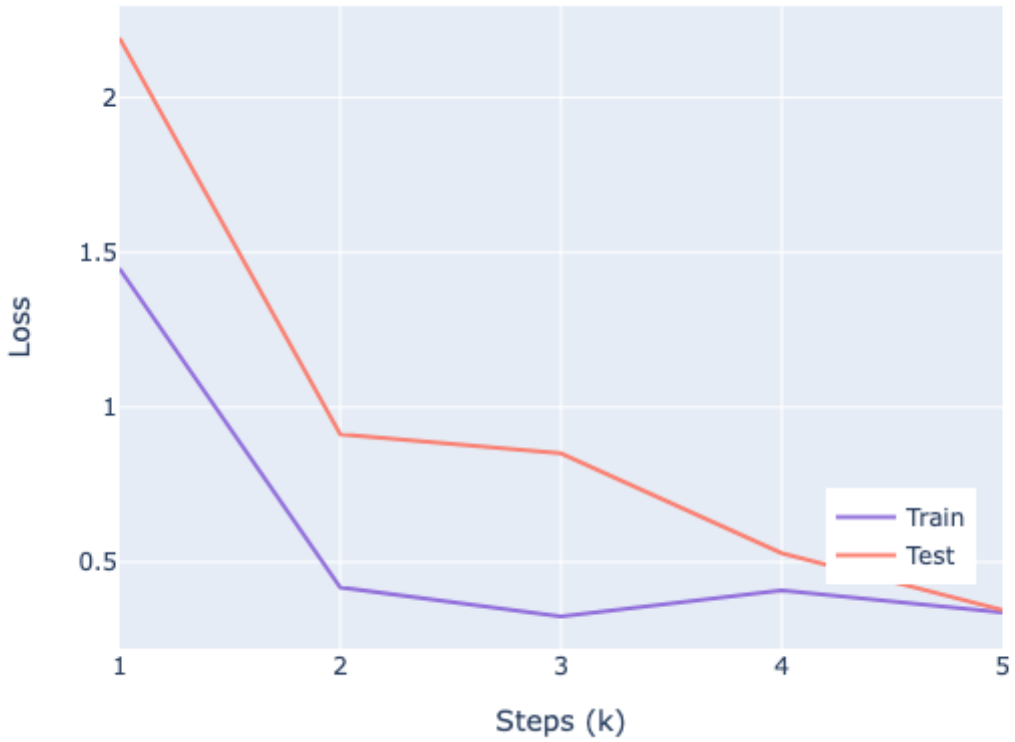


Fig. 43. Pérdida Total.

Fuente: Propia

TABLA 12. Pérdida Total

Step	Train	Test
1k	1,4480	2,1920
2k	0,4184	0,9125
3k	0,3245	0,8515
4k	0,4088	0,5292
5k	0,3376	0,3450

Discusión

Dentro de las arquitecturas de tipo SSD (Single Shot Detection), se escogió la arquitectura más simple de la familia de las EfficientDet, la EfficientDet D0. La arquitectura EfficientDet D0 tiene menos parámetros entrenables que muchas de las arquitecturas más utilizadas actualmente y que la literatura compara por rendimiento como YoloV3 o RetinaNet. Como se pudo analizar en la *figura 7*, la reducción de los parámetros entrenables de la arquitectura seleccionada permite evitar el sobreajuste (overfitting) del modelo. Los hiperparámetros seleccionados fueron tomados de trabajos afines descritos en la sección *1.6*, aunque se tuvo que experimentar distintas combinaciones para dar con la más apropiada para el caso. Mientras que muchos recomiendan *Adam* como optimizador, en el entrenamiento del modelo AllSkyNet, lo más acertado fue trabajar con Momentum, ya que evita el estancamiento en óptimos locales por su cualidad de actualización de pesos y bias en cada iteración (Solawetz, 2020). Se infiere que la causa de este fenómeno se encuentra en la calidad propia del dataset.

Por otra parte, el proceso Knowledge Discovery in Databases (KDD) fue tomado como metodología de desarrollo, de esta manera se pudo tener un marco de referencia estructural de cómo se debe tratar los datos para el desarrollo de un modelo de red neuronal profunda.

Cabe mencionar las herramientas cloud que se consideraron para el desarrollo del modelo. Trabajos afines proponen Roboflow en su versión gratuita y Google Colab en su versión Pro +. Con esto se garantiza un desarrollo eficaz y ágil del modelo ya que permiten realizar ciertas tareas como la selección, pre procesamiento y transformación de una forma muy versátil en una sola herramienta. Algunos de los trabajos afines revisados proporcionaban el tiempo de entrenamiento de sus modelos, rondando las 24 horas siendo similar al tiempo que le tomó al AllSkyNet.

Como aporte adicional, la herramienta de detección de objetos con GUI de BMW (BMW Innovation Lab, 2022) se encontró al final del desarrollo del modelo. Esta herramienta es una interfaz gráfica basada en la API de detección de objetos de Tensorflow 2. Está disponible para Java, Python, JavaScript, TypeScript, C++ y Scala.

Al final de la investigación, hay que mencionar que no se ha encontrado trabajos similares acerca de la detección de meteoros con los que se pueda realizar una comparación de la propuesta.

Limitaciones

En la recolección y selección de las imágenes idóneas para el entrenamiento, se debe tener muy encuentra que las imágenes deben ser de al menos el tamaño que requiere la arquitectura a usar o mayor, siendo en la presente investigación, un tamaño de 512 píxeles por 512 píxeles. Se experimentó al inicio del proyecto con imágenes inferiores a los tamaños aconsejados y la cantidad de ruido que existe en las imágenes es tal, que el modelo era incapaz de aprender características relevantes dentro del dataset.

El etiquetado de las imágenes siempre ha sido un proceso que condiciona mucho el tiempo de desarrollo de un modelo de detección, por lo que se tiene en mente desarrollar en un futuro, una herramienta que ayude a etiquetar de una manera más elegante y versátil.

A la hora de transformar el dataset etiquetado al tipo de dato requerido por la API de detección de objetos de Tensorflow 2, esta debe exportarse estrictamente en formato TFRecord por lo que un conocimiento de este tipo de dato es preciso para entender qué es lo que son, cómo trabajan y la eficiencia que tienen en comparación con tipos como CSV, XML o JSON. La documentación de este conocimiento no es nada fácil de encontrar y se debe inferir al principio para poder luego atar cabos con la poca literatura existente.

Se encontraron algunos inconvenientes a la hora del entrenamiento, retrasando este proceso. Cuando se procede a instalar las dependencias que se requiere con el script *setup.py*, la librería de opencv-python se encuentra en la versión 4.1.2.30 mientras que la librería conjunta opencv-python-headless se encuentra en la versión 4.1.2.64. Esta variación en la versión no permite que el entrenamiento del modelo inicie por lo que se debe configurar la versión de opencv-python-headless para que coincida con la de opencv-python. Ambas deberían tener la versión 4.1.2.30. Se exponen los comandos a ejecutar en la terminal para la solvencia de esta inconsistencia.

!pip uninstall opencv-python-headless==4.5.5.62 -y
!pip install opencv-python-headless==4.1.2.30

Antes de optar por un entrenamiento en la nube de Google Cloud, el entrenamiento se realizaba en una tarjeta gráfica local Intel UHD Graphics 630 de 1536 MB. Puede que la falta de experticia juegue un papel importante pero la observación y comparación de un modelo entrenado en esta GPU con un modelo entrenado en Google Colab con GPU/TPU, dio como resultado que el modelo entrenado con la GPU local es menos preciso.

A la hora de entrenar con GPU en Google Colab, es suficiente con ejecutar las líneas que se encuentran en el notebook utilizado, pero cuando se adquiere el plan de Colab Pro +, existe otra inconsistencia en la versión del CUDA y cuDNN, que debe ser resuelta antes de poder entrenar el modelo. Al ejecutar la siguiente línea en la terminal, actualiza la versión de estas dos herramientas a la versión 8.1.0.77 y 11.2 permitiendo el entrenamiento del modelo.

!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2

Por último, hay que mencionar que ciertas métricas como el Accuracy o la curva ROC que usualmente se usan para evaluar modelos, no aportan información relevante para evaluar modelos de detección de objetos de una sola clase (meteoros). Se investigó acerca de este tipo de casos y se pudo denominar al dataset como “dataset no balanceado” debido a que su clase meteoro es la única existente, por lo que no existen ground truth en una clase opuesta o negativa. Para este tipo de problemas, métricas como la precisión, sensibilidad y curva de Precisión vs. Sensibilidad son mucho más acertadas (Saito & Rehmsmeier, 2015), por lo que fueron las utilizadas en el capítulo 3.

Conclusiones

A la hora de elaborar el marco teórico referencial, se pudo establecer las herramientas propicias para resolver el problema de detección de objetos. La arquitectura de tipo SSD específicamente la EfficientDet D0, favoreció en la detección multi escala, aunque se aconseja que las imágenes de entrada sean lo más grandes posibles en términos de resolución, debido a su arquitectura piramidal descendente.

En el desarrollo del modelo se subió el conjunto de datos (dataset) a Roboflow para procesar las imágenes rápidamente. Esta herramienta permitió un pre procesamiento más organizado, transformando las imágenes a la resolución requerida (512x512 píxeles), dividiendo el dataset en proporción 80/20 y exportando el dataset en el tipo de dato requerido (TFRecord), siendo esta la entrada para el entrenamiento del modelo. El modelo fue entrenado con la API de detección de objetos de Tensorflow 2. Esta herramienta da muy buenos resultados a la hora de resolver problemas de clasificación y regresión simultáneamente, aunque su curva de aprendizaje es elevada, debido a la baja documentación existente en internet.

La plataforma de entrenamiento fue Google Colab Pro + donde, clonando tanto el repositorio de la API de detección de objetos de Tensorflow 2 como la arquitectura EfficientDet D0, se realizó el entrenamiento en un tiempo de 20 horas y 30 minutos. Al haber adquirido el plan de Google Colab más avanzado, se disponía de entrenamiento en segundo plano, por lo que se pudo descuidar el entrenamiento durante ese tiempo. Cuando el modelo estuvo refinado al máximo, se procedió a desarrollar una interfaz gráfica con la que probar el modelo con las imágenes del conjunto de datos de prueba. Esta fue desarrollada con Flask haciendo uso de las mismas tecnologías que los actuales servicios software de la AllSkyCams para que la futura implantación y despliegue sea completamente compatible.

Al haber realizado el entramiento con una sola clase “meteoro” se evita lo que Mike Hankey de AllSkyCams recurrió a realizar, esto fue, un módulo o script por cada objeto que no necesita ser analizado, así como aves, aviones, nubes, alumbrado público. En vez de este laborioso trabajo, el modelo de detección de meteoros *AllSkyNet*, es capaz de detectar únicamente meteoros. Se vio en la validación del modelo con imágenes que no era meteoros, que el modelo es capaz de discriminar bastante bien a los demás objetos que aparecen en las imágenes. Teniendo solamente un 19,36% en promedio de que estos falsos positivos son meteoros.

En el capítulo 3, se despliegan las evaluaciones realizadas al modelo entrenado. Como se esperaba en el alcance del presente proyecto, la precisión de la detección debía ser como

mínimo de 85%. Los resultados se dividieron en métricas de evaluación para el problema de clasificación y métricas de evaluación para el problema de regresión. Con la matriz de confusión realizada se obtuvo un 99,36% de precisión para el problema de clasificación y una sensibilidad de 90,45% para el problema de clasificación apostando por un balance entre precisión y sensibilidad requerida por AllSkyCams. La relación promedio de estas dos métricas denominado F1 Score fue de 94,66%. La pérdida final del modelo para el problema de clasificación en entrenamiento fue de 0,1945 y en validación de 0,1991. Asegurando que no se tiene ni overfitting ni underfitting por la poca distancia que existe entre estos valores. Para finalizar con las métricas de clasificación se expone la *figura 38* que relaciona la precisión con la sensibilidad, siendo inversamente proporcional ya que, si se requiere de más precisión, la sensibilidad será menor y viceversa. El balance entre estas dos fue tomado como el adecuado para el caso.

Con la métrica de evaluación Average Precision (AP) con un umbral de 0,5 se obtuvo una precisión de 92,7% para el problema de regresión referente a la localización de los meteoros (bounding box). La pérdida final del modelo para el problema de regresión en entrenamiento fue de 0,1113 y en validación de 0,1140. Asegurando que tampoco se tiene ni overfitting ni underfitting por la poca distancia que existe entre estos valores.

Con estos valores obtenidos, el modelo ha sido desarrollado con éxito, cumpliendo con creces los objetivos propuestos.

Recomendaciones

Se recomienda usar Roboflow para el procesamiento del dataset ya que permite exportar la entrada al entrenamiento de manera correcta. Existen scripts de terceros que transforman de CSV o JSON a TFRecord, pero suelen tener las coordenadas de los bounding box modificados. Un ejemplo sería que sus coordenadas sigan el orden xmin, ymin, xmax, ymax pero el script de transformación a tfrecord siguen el orden xmin, xmax, ymin, ymax. Se expone el ejemplo por experiencia propia al desarrollar el presente trabajo.

Para poder evaluar el modelo adecuadamente, se recomienda exportarlo “Freezing the Graph”. Guardando sus variables, pesos y checkpoint del modelo. Con esto, es posible generar la matriz de confusión.

Por ahora no existe una forma unificada de desplegar un gráfico que detalle la precisión o perdidas en cada checkpoint, por lo que se recomienda realizar una evaluación con cada checkpoint. Esto generará archivos denominados tfevents por cada evaluación que luego pueden ser visualizados en conjunto con la herramienta de TensorBoard.

Se recomienda revisar el archivo pipeline.config que es el archivo que contiene todas las configuraciones del entrenamiento y validación, para verificar si la arquitectura requiere de GPU o TPU para el entrenamiento. Este dato aparece al inicio del archivo indicando qué se debe usar para disminuir los tiempos de entrenamiento al mínimo y obtener resultados acordes al tipo de GPU requerida.

Se recomienda la herramienta de detección de objetos con GUI de BMW para realizar cualquier proyecto que se tenga en mente ya que permite entrenar y validar el modelo mediante una interfaz gráfica amigable, evitando engorrosas configuraciones de bajo nivel que requieren más conocimiento en Tensorflow. AllSkyNet en su siguiente versión se desarrollará con esta herramienta. Principalmente, para comparar si los resultados se ven afectados y con ello poder diseñar modelos más eficientes. La ventaja de esta herramienta es que, en la evaluación del modelo, proporciona diversos gráficos muy interesantes que aportar información valiosa. La desventaja es que se requiere de una GPU física y de conocimientos básicos en Docker (Docker, 2022).

Referencias

Affonso, C., Rossi, A. L. D., Vieira, F. H. A., & de Carvalho, A. C. P. de L. F. (2017). Deep learning for biological image classification. *Expert Systems with Applications*, *85*, 114–122. https://doi.org/10.1016/j.eswa.2017.05.039

Ali-Dib, M., Menou, K., Jackson, A. P., Zhu, C., & Hammond, N. (2020). Automated crater shape retrieval using weakly-supervised deep learning. *Icarus*, *345*(February), 113749. https://doi.org/10.1016/j.icarus.2020.113749

Armstrong, J. A., & Fletcher, L. (2019). Fast Solar Image Classification Using Deep Learning and Its Importance for Automation in Solar Physics. *Solar Physics*, *294*(6). https://doi.org/10.1007/s11207-019-1473-z

Baek, J. H., Kim, S., Choi, S., Park, J., Kim, J., Jo, W., & Kim, D. (2021). Solar Event Detection Using Deep-Learning-Based Object Detection Methods. *Solar Physics*, *296*(11). https://doi.org/10.1007/s11207-021-01902-5

Bai, X., Pang, Y., & Zhang, G. (2020). Special focus on deep learning for computer vision. *Science China Information Sciences*, *63*(2), 2–3. https://doi.org/10.1007/s11432-020-2766-x

Barchi, P. H., de Carvalho, R. R., Rosa, R. R., Sautter, R. A., Soares-Santos, M., Marques, B. A. D., Clua, E., Gonçalves, T. S., de Sá-Freitas, C., & Moura, T. C. (2020). Machine and Deep Learning applied to galaxy morphology - A comparative study. *Astronomy and Computing*, *30*, 100334. https://doi.org/10.1016/j.ascom.2019.100334

Bird, J., Petzold, L., Lubin, P., & Deacon, J. (2021). Advances in deep space exploration via simulators & deep learning. *New Astronomy*, *84*(August 2020), 101517. https://doi.org/10.1016/j.newast.2020.101517

BMW Innovation Lab. (2022). *BMW TensorFlow Training GUI*. BMW Innovation Lab. https://www.programcreek.com/python/?project_name=BMW-InnovationLab%2FBMW-TensorFlow-Training-GUI#

Carrasco-Davis, R., Cabrera-Vives, G., Förster, F., Estévez, P. A., Huijse, P., Protopapas, P., Reyes, I., Martínez-Palomera, J., & Donoso, C. (2019). Deep learning for image sequence classification of astronomical events. *Publications of the Astronomical Society of the Pacific*, *131*(1004), 108006. https://doi.org/10.1088/1538-3873/aaef12

Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2018). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(4), 834–848. https://doi.org/10.1109/TPAMI.2017.2699184

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-Janua*, 1800–1807. https://doi.org/10.1109/CVPR.2017.195

Čokina, M., Maslej-Krešňáková, V., Butka, P., & Parimucha. (2021). Automatic classification of eclipsing binary stars using deep learning methods. *Astronomy and Computing*, *36*, 100488. https://doi.org/10.1016/j.ascom.2021.100488

Docker. (2022). *Develop faster. Run anywhere.* Docker.Com.

Dong, C., Loy, C. C., He, K., & Tang, X. (2016). Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(2), 295–307. https://doi.org/10.1109/TPAMI.2015.2439281

Farrukh, M. U. (2020). *Modeling on Feature Vectors in Compressed Spaces by the use of Neural network techniques*. *March 2019*, 0–9. https://doi.org/10.13140/RG.2.2.29790.59203

Garbisu Arocha, H. (2016). Utilización de redes neuronales recurrentes para el análisis de firmas. *Universidad de Las Palmas de Gran Canaria*, 41. https://accedacris.ulpgc.es/jspui/handle/10553/18830

Garcia, R., & Alegre, P. (2019). *Visual Analytics as a Tool for Deep Learning Engineering*. *June*.

Gechman, M. (2013). Software Development Methodologies. *Project Management of Large Software-Intensive Systems*, *June*, 49–66. https://doi.org/10.1201/9780429027932-4

González, R. E., Muñoz, R. P., & Hernández, C. A. (2018). Galaxy detection and identification using deep learning and data augmentation. *Astronomy and Computing*, *25*, 103–109. https://doi.org/10.1016/j.ascom.2018.09.004

Hankey, M., Perlerin, V., & Meisel, D. (2020). The all-sky-6 and the Video Meteor Archive system of the AMS Ltd. *Planetary and Space Science*, *190*(November 2019), 105005. https://doi.org/10.1016/j.pss.2020.105005

He, T., & Li, X. (2019). Image quality recognition technology based on deep learning. *Journal of Visual Communication and Image Representation*, *65*, 102654. https://doi.org/10.1016/j.jvcir.2019.102654

Horak, K., & Sablatnig, R. (2019). *Deep learning concepts and datasets for image recognition: overview 2019*. *February 2020*, 100. https://doi.org/10.1117/12.2539806

Hosenie, Z., Bloemen, S., Groot, P., Lyon, R., Scheers, B., Stappers, B., Stoppa, F., Vreeswijk, P., De Wet, S., Wolt, M. K., Körding, E., McBride, V., Le Poole, R., Paterson, K., Pieterse, D. L. A., & Woudt, P. (2021). MeerCRAB: MeerLICHT classification of real and bogus transients using deep learning. *Experimental Astronomy*, 319–344. https://doi.org/10.1007/s10686-021-09757-1

Jaramillo, M., & Bustamante, D. (2020). *Anomaly Detection System in Video Surveillance using Deep Learning Techniques*. https://repositorio.yachaytech.edu.ec/handle/123456789/183

Jia, X. (2017). Image recognition method based on deep learning. *Proceedings of the 29th*

Chinese Control and Decision Conference, CCDC 2017, 4730–4735. https://doi.org/10.1109/CCDC.2017.7979332

Jimenez, M., Torres Torres, M., John, R., & Triguero, I. (2020). Galaxy image classification based on citizen science data: A comparative study. *IEEE Access*, *8*, 47232–47246. https://doi.org/10.1109/ACCESS.2020.2978804

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P. L., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., … Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. *Proceedings - International Symposium on Computer Architecture*, *Part F1286*, 1–12. https://doi.org/10.1145/3079856.3080246

Jyothi, V. K., Guru, D. S., & Sharath Kumar, Y. H. (2018). Deep Learning for Retrieval of Natural Flower Videos. *Procedia Computer Science*, *132*, 1533–1542. https://doi.org/10.1016/j.procs.2018.05.117

Karanam, S. R., Srinivas, Y., & Krishna, M. V. (2020). Study on image processing using deep learning techniques. *Materials Today: Proceedings*, *xxxx*. https://doi.org/10.1016/j.matpr.2020.09.536

Krishna, M. M., Neelima, M., Harshali, M., & Rao, M. V. G. (2018). Image classification using Deep learning. *International Journal of Engineering and Technology(UAE)*, *7*(March), 614–617. https://doi.org/10.14419/ijet.v7i2.7.10892

Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-Janua*, 936–944. https://doi.org/10.1109/CVPR.2017.106

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, *234*(October 2016), 11–26. https://doi.org/10.1016/j.neucom.2016.12.038

Luís, A., & Medeiros, S. De. (2019). *Deep Learning for Single Image Super-Resolution using residual image learning and multiple degradations*.

Lukic, V., de Gasperin, F., & Brüggen, M. (2020). ConvoSource: Radio-Astronomical source-finding with convolutional neural networks. *Galaxies*, *8*(1). https://doi.org/10.3390/GALAXIES8010003

Manassés, R. (2018). *Deep Learning Methods for Detecting Anomalies in Videos : Theoretical and Methodological Contributions*. 120.

Mittal, A., Soorya, A., Nagrath, P., & Hemanth, D. J. (2020). Data augmentation based morphological classification of galaxies using deep convolutional neural network. *Earth Science Informatics*, *13*(3), 601–617. https://doi.org/10.1007/s12145-019-00434-8

Morera, Á., Sánchez, Á., Moreno, A. B., Sappa, Á. D., & Vélez, J. F. (2020). Ssd vs. Yolo for

detection of outdoor urban advertising panels under multiple variabilities. *Sensors (Switzerland)*, *20*(16), 1–23. https://doi.org/10.3390/s20164587

Nwankpa, C. E., Ijomah, W., Gachagan, A., & Marshall, S. (2018). *Activation Functions : Comparison of Trends in Practice and Research for Deep Learning*. 1–20. https://arxiv.org/pdf/1811.03378.pdf

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., & Walsh, J. (2020). Deep Learning vs. Traditional Computer Vision. *Advances in Intelligent Systems and Computing*, *943*(Cv), 128–144. https://doi.org/10.1007/978-3-030-17795-9_10

O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. *November*. http://arxiv.org/abs/1511.08458

Ordóñez, F. J., & Roggen, D. (2016). Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. *Sensors (Switzerland)*, *16*(1). https://doi.org/10.3390/s16010115

Pérez, J. (2019). ANÁLISIS DE TRÁFICO VEHICULAR MEDIANTE VISIÓN ARTIFICIAL. *Αγαη*, *8*(5), 114.

Pypi.org. (2022). *Pypi*.

Raju, H., & Das, S. (2021). CNN-Based Deep Learning Model for Solar Wind Forecasting. *Solar Physics*, *296*(9), 1–25. https://doi.org/10.1007/s11207-021-01874-6

Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, *25*(6), 5193–5254. https://doi.org/10.1007/s10664-020-09881-0

Roboflow. (2022). *Roboflow*.

Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, *10*(3), 1–21. https://doi.org/10.1371/journal.pone.0118432

Saranya, P., Monica, V., Priyadharshini, J., & N, A. P. D. (2017). *Comparative Study of Software Development Methodologies*. 172–179.

Shilon, I., Kraus, M., Büchele, M., Egberts, K., Fischer, T., Holch, T. L., Lohse, T., Schwanke, U., Steppa, C., & Funk, S. (2019). Application of deep learning methods to analysis of imaging atmospheric Cherenkov telescopes data. *Astroparticle Physics*, *105*, 44–53. https://doi.org/10.1016/j.astropartphys.2018.10.003

Shin, H. C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., & Summers, R. M. (2016). Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging*, *35*(5), 1285–1298. https://doi.org/10.1109/TMI.2016.2528162

Silburt, A., Ali-Dib, M., Zhu, C., Jackson, A., Valencia, D., Kissin, Y., Tamayo, D., & Menou,

K. (2019). Lunar crater identification via deep learning. *Icarus*, *317*, 27–38. https://doi.org/10.1016/j.icarus.2018.06.022

Smithson, S. C., Yang, G., Gross, W. J., & Meyer, B. H. (2016). *Neural networks designing neural networks*. 1–8. https://doi.org/10.1145/2966986.2967058

Solawetz, J. (2020). *How to Train a TensorFlow 2 Object Detection Model*. https://blog.roboflow.com/train-a-tensorflow2-object-detection-model/

Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 10778–10787. https://doi.org/10.1109/CVPR42600.2020.01079

Tanoglidis, D., Ćiprijanović, A., & Drlica-Wagner, A. (2021). DeepShadows: Separating low surface brightness galaxies from artifacts using deep learning. *Astronomy and Computing*, *35*, 100469. https://doi.org/10.1016/j.ascom.2021.100469

Tensorflow.org. (2022). *Tensorflow*.

Thakur, N., Raju, S., & Gupta, A. (2017). Feature extraction: an application to object and image recognition. *International Journal of Latest Trends in Engineering and Technology*, *1*, 178–184. https://doi.org/10.21172/1.91.27

Tiwari, A., Goswami, A. K., & Saraswat, M. (2013). Feature Extraction for Object Recognition and Image Classification. *International Journal of Engineering Research & Technology (IJERT)*, *2*(10), 1238–1246. http://www.ijert.org/view-pdf/5674/feature-extraction-for-object-recognition-and-image-classification

Vanderbilt Museum. (2020). *Asteroids, Meteors and Meteroids:*

Walmsley, M., Lintott, C., Géron, T., Kruk, S., Krawczyk, C., Willett, K. W., Bamford, S., Kelvin, L. S., Fortson, L., Gal, Y., Keel, W., Masters, K. L., Mehta, V., Simmons, B. D., Smethurst, R., Smith, L., Baeten, E. M., & MacMillan, C. (2022). Galaxy Zoo DECaLS: Detailed visual morphology measurements from volunteers and deep learning for 314 000 galaxies. *Monthly Notices of the Royal Astronomical Society*, *509*(3), 3966–3988. https://doi.org/10.1093/mnras/stab2093

Wang, X., Niu, S., & Wang, H. (2021). Image Inpainting Detection Based on Multi-task Deep Learning Network. *IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India)*, *38*(1), 149–157. https://doi.org/10.1080/02564602.2020.1782274

Xin, M., & Wang, Y. (2019). Research on image classification model based on deep convolution neural network. *Eurasip Journal on Image and Video Processing*, *2019*(1). https://doi.org/10.1186/s13640-019-0417-8

Zhang, X., Wang, L., & Su, Y. (2021). Visual place recognition: A survey from deep learning perspective. *Pattern Recognition*, *113*, 107760. https://doi.org/10.1016/j.patcog.2020.107760

Zhao, R., Geng, J., & Shen, Y. (2020). Digital media vision innovation based on FPGA and Convolutional Neural Network. *Microprocessors and Microsystems*, *October*, 103465. https://doi.org/10.1016/j.micpro.2020.103465

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object Detection with Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Zoph, B., Le, Q., & Ramachandran, P. (2018). Swish: A Self-Gated Activation Function. *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, *1*, 1–12.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning Transferable Architectures for Scalable Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 8697–8710. https://doi.org/10.1109/CVPR.2018.00907

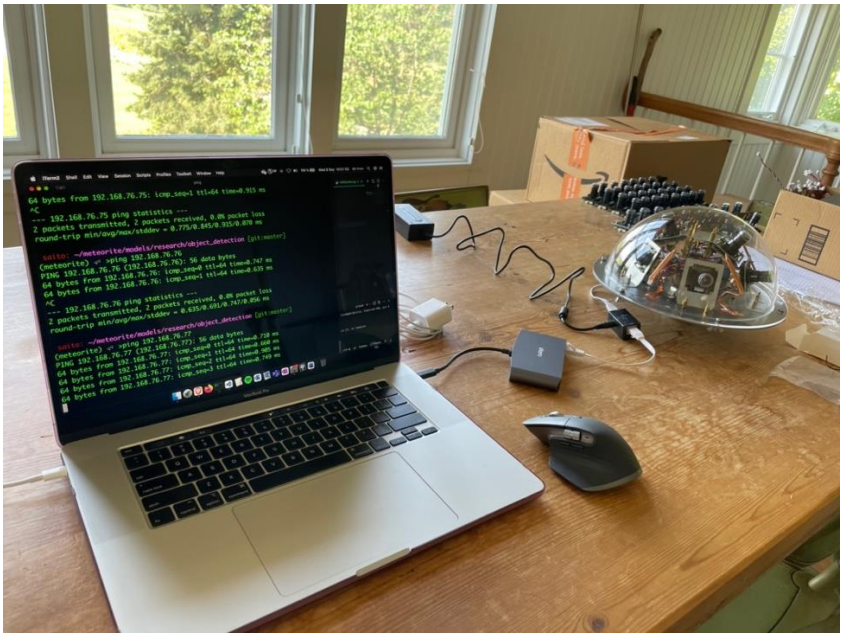
Apéndice

Apéndice A: Dennis Chicaiza en el proceso de etiquetado del dataset. Enero de 2021.

Apéndice B: Predicción del primer modelo de detección. Julio de 2021.

Apéndice C: Dennis Chicaiza montando de sus primeras AllSkyCam7. Septiembre de 2020.

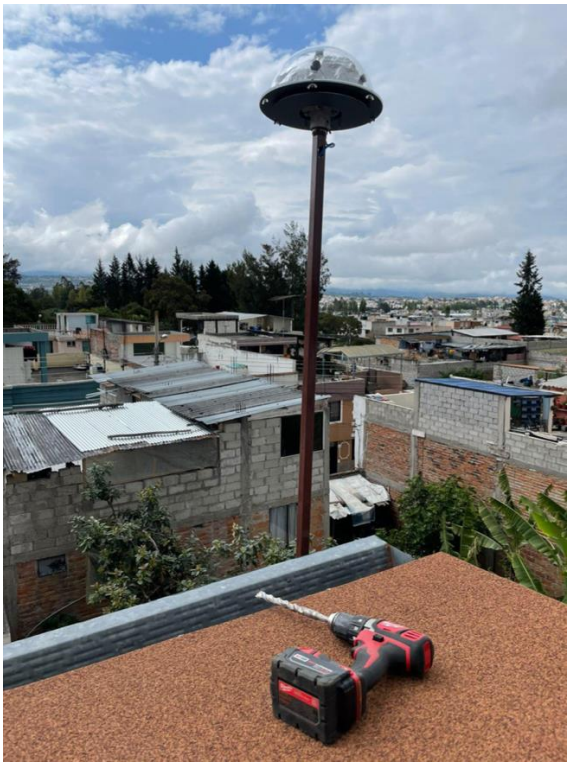
Apéndice D: Test de conexión de la AllSkyCam7. Septiembre de 2020.



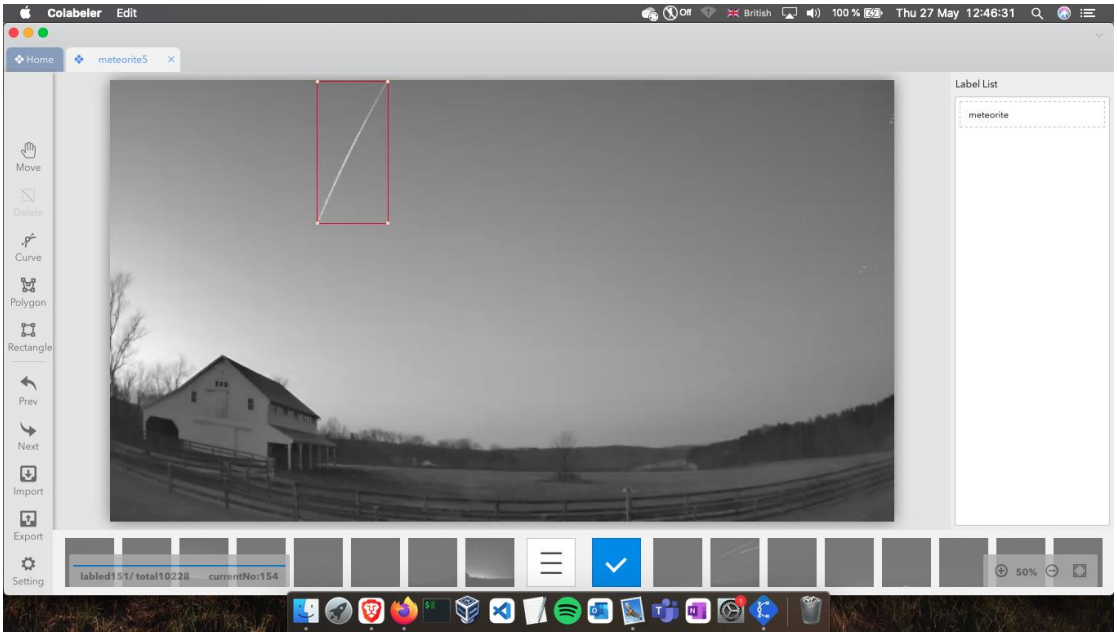
Apéndice E: Montaje de la AllSkyCam7 en Ibarra, Ecuador. Estación AMS139. Diciembre de 2021.

Apéndice F: AllSkyCam7. Estación AMS139 montada en Ibarra, Ecuador. Diciembre de 2021.

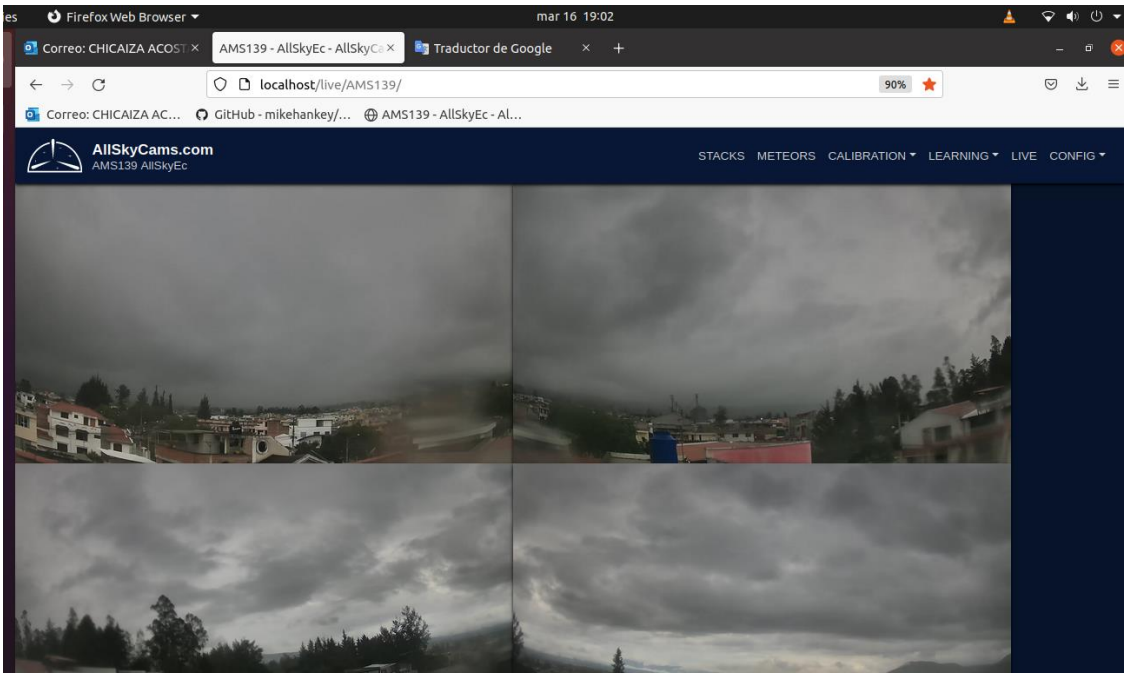
Apéndice G: AllSkyCam7. Estación AMS139 montada en Ibarra, Ecuador. Diciembre de 2021.



Apéndice H: Etiquetado de los meteoros en Colabeler. Mayo de 2021.



Apéndice I: Software actual del AllSkyCams monitoreando Ibarra. Marzo de 2022.



Apéndice J: Evidencia de todos los cien entrenamientos realizados con la API de detección de objetos de Tensorflow 2. Desde febrero 2022 hasta junio de 2022.