

UNIVERSIDAD NACIONAL DE CÓRDOBA



PROYECTO INTEGRADOR INGENIERÍA EN COMPUTACIÓN

Detector de Pivotes de Riego y Silobolsas en Imágenes Satelitales para aplicaciones agrícolas

Autor: Gianfranco BARBIANI

Matricula: 36372639

Tel.: 3512093135

Email: gianfran-
co.barbiani@mi.unc.edu.ar

Director: Mg. Ing. Miguel
SOLINAS

Autor: Loreley BUSTAMANTE

Matricula: 32298299

Tel.: 3516770907

Email: lore-
ley.bustamante@mi.unc.edu.ar

Codirector: Dr. Ing. Miguel
SOLINAS Jr.

15 de septiembre de 2022

Resumen En este Proyecto Integrador se diseñó e implementó una aplicación que hace uso de Machine Learning (ML) aplicado a la detección de objetos en imágenes satelitales para su aplicación en agricultura. Se describen las motivaciones, problemas encontrados y posibles soluciones. Se presentan principios de ML, deep learning y visión por computadora. Por último se presentan algunas conclusiones y sugerencias para futuros trabajos.

Keywords— Computer vision, Object detection, Artificial satellites, Rural areas, Public domain software

Índice general

1. Introducción	1
1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	2
1.2.1. Objetivos Principales	2
1.2.2. Objetivos Particulares	2
1.3. Marco de investigación	4
2. Marco Teórico	6
2. Principios de Machine Learning	6
2.1. Introducción: Visión por Computadora	6
2.2. Fundamentos de Machine Learning	6
2.2.0.1. Ajuste del modelo: ¿Por qué se entrena un modelo de Machine Learning?	8
2.3. Redes Neuronales	8
2.4. Definición formal de Redes Neuronales	9
2.5. Principios de la Detección de Objetos	11
2.5.1. Conceptos	12
2.5.1.1. Bounding box	12
2.5.1.2. Non-max suppression	13
2.5.1.3. Anchor Boxes	14
2.5.1.4. Intersection Over Union	16
2.6. Dataset	18
2.6.1. Pre-entrenamiento	18
2.6.2. Formato de las Etiquetas	19
2.6.3. Dataset Personalizado	21
2.6.3.1. Distribución de Etiquetas	22
2.6.3.2. Correlograma de Etiquetas	23
3. Estado del Arte	25
3. Introducción	25
3.1. Detección de Objetos	25
3.2. YOLO: You Only Look Once	25
3.2.1. Funcionamiento de YOLO	26
3.2.1.1. Workflow	27
3.2.2. YOLOv3	28
3.2.2.1. Mejoras	28
3.2.2.2. Comparación con otras estructuras	32
3.2.3. YOLOv4	33
3.2.3.1. Mejoras	33
3.2.4. YOLOv5	36
3.2.4.1. Mejoras	36
3.2.4.2. Memoria y FLOPS	37
3.2.4.3. Comparación de modelos de YOLOv5	38

4. Análisis del Diseño del Sistema	40
4. Introducción	40
4.1. Entrevistas	40
4.2. Motivación e Importancia del proyecto	41
4.3. Casos de uso de Machine Learning - Detección de objetos	41
4.4. Funcionalidades	42
4.5. Definición de los Componentes	42
4.5.1. Selección de los Componentes	43
4.6. Definición de los Requerimientos	45
4.6.1. Diagrama de casos de uso	45
4.7. Definición del Comportamiento	48
4.8. Riesgos	49
4.8.1. Criterios	49
4.8.2. Identificación de Riesgos	49
4.8.3. Análisis de Riesgos	51
4.8.4. Planificación de Riesgos	52
4.9. Control de Versiones	53
5. Implementación	54
5. Introducción	54
5.1. Entornos de trabajo	54
5.1.1. Configuración Nabucodonosor	55
5.1.1.1. Machine Learning & Backend Docker	56
5.1.1.2. Frontend Docker	56
5.2. Dataset	57
5.2.1. Data Augmentation	57
5.2.2. Formatos	58
5.3. Entrenamiento	58
5.4. Pruebas	58
5.4.1. Precisión	59
5.4.2. Recall o Sensibilidad	60
5.4.3. Precisión/Recall	61
5.4.4. F1 Score	62
5.4.5. Loss Function o Función de Pérdida y Precisión Media Promedio (mAP, Mean Average Precision)	63
5.4.6. Matriz de Confusión	66
5.5. Despliegue	68
6. Validación	69
6. Introducción	69
6.1. Validación de los requerimientos de usuario	69
6.1.1. Matriz de trazabilidad	74
7. Conclusión	77
7. Introducción	77
7.1. Problemas y Limitaciones	77
7.2. Trabajos futuros	79
7.3. Aporte personal	79
7.4. Agradecimientos	80

8. Apéndice	81
8. Imágenes Dataset	81
9. Manual de Usuario	90
9.1. Introducción	90
9.2. Pre-requisitos	90
9.3. Instalación	90
9.4. Ingresar a la página	90
9.5. Cargar una imagen	90
9.5.1. Error en el formato de la imagen	91
9.6. Cargar varias imágenes	91
9.7. Inicio de la detección	91
9.8. Descarga de imágenes	92

Índice de figuras

1. Riegos por pivote (circular): sistema de riego móviles mediante pivotes que permiten regar grandes superficies. Fuente: [1]	3
2. Silobolsas: es un sistema que permite almacenar granos secos de maíz, soja, trigo, girasol y arroz en el propio establecimiento productor, a bajo costo y con óptimas condiciones de calidad. Fuente: [2]	3
3. Workflow	4
4. Clasificación. Fuente: [3]	7
5. Regresión. Fuente: [3]	7
6. Ejemplo Linealmente Separable. Fuente: [3]	8
7. Abstracción de una Red Neuronal. Fuente: [4]	8
8. Ejemplo Machine Learning. Fuente: [5]	9
9. Tareas de Visión por computadora. Fuente: [6]	11
10. Bounding Box in YOLO. Fuente:[7]	12
11. Non Max Supression in YOLO. Fuente:[7]	13
12. Anchor Boxes en YOLO. Fuente:[7]	15
13. Ground Thruth and Predicted Bounding Box. Fuente:[8]	16
14. Intersección over Union formula	17
15. Intersección over Union examples	18
16. Las 20 clases de VOC2012. Fuente: [9]	19
17. Distribución de Etiquetas	22
18. Correlograma de Etiquetas: nos ayudan a visualizar la correlación entre las etiquetas de los objetos.	23
19. El sistema de detección de YOLO: Procesar imágenes con YOLO es simple y directo. El sistema (1) cambia el tamaño de la imagen de entrada a 448 × 448, (2) ejecuta una sola red convolucional en la imagen y (3) establece un umbral para las detecciones resultantes según la confianza del modelo. Fuente: [10]	26
20. Funcionamiento de YOLO: primero se divide la imagen en cuadrículas más pequeñas, sobre éstas se calculan diferentes cajas delimitadoras o bounding boxes y las probabilidades de las distintas clases que se encuentre en ellas, y finalmente se obtienen las detecciones. Fuente: [10].	27
21. Predicciones de las cajas delimitadoras. Fuente: [11]	29
22. Cajas delimitadoras con dimensiones previas y predicciones. Se predice el ancho y la altura de la caja como offset desde el centro de la caja. Fuente: [11]	29
23. Redes Piramidales de Características. Fuente: [12]	30
24. Darknet-53. Fuente: [11]	31
25. Comparación de distintas estructuras de redes neuronales convolucionales: se observa el balance entre velocidad y precisión para 0.5 IOU, destacando YOLOv3 por su gran velocidad y manteniendo una buena precisión. Fuente[11].....	32
26. Principales componentes de un Detector de Objetos moderno de una etapa. Fuente: [13]	33
27. Ejemplo de Aumento de datos [14]	34
28. CutMix	35
29. Mosaic representa un nuevo método de aumento de datos. [13]	35

30. Suavizado de Etiquetas	36
31. Diferentes métodos presentes en la Bag of Specials. Fuente:[15]	37
32. Módulo de Atención Espacial. Fuente: [16]	37
33. Módulo de Atención Espacial Modificado. Fuente: [13]	38
34. Arquitecturas: Memoria y FLOPS. Fuente: [17]	39
35. YOLOv5 - Rendimiento. Fuente: [17]	39
36. Workflow: Alcance	40
37. Arquitectura del Software Implementado	43
38. Diagrama de Caso de usos	45
39. Diagrama de Componentes	47
40. Diagrama de Secuencia	48
41. Workflow: Implementación	54
42. Workflow: Etapa de Datos	57
43. Workflow: Etapa de Modelado	58
44. Precisión	60
45. Recall	61
46. Precisión/Recall	62
47. Curva F1 Score	62
48. Loss Function y mAP. Primeras 100 epochs	64
49. Loss Function y mAP. Epochs 101 a 700.	65
50. Matriz de Confusión. Primeras 100 epochs	66
51. Matriz de Confusión. Epochs 101 a 700.	67
52. Workflow: Etapa de Despliegue	68
53. Arquitectura del Software Implementado	68
54. Página principal	70
55. Subir una imagen para analizar	70
56. Seleccionar una imagen desde la PC-Usuario	71
57. Inicio de la detección - Loading	71
58. Error al subir una archivo de formato invalido	72
59. Fin de la detección - Resultado	73
60. Tabla de Información de detecciones - Resultado	74
61. Detección de un archivo con imágenes comprimidas	75
62. Descarga de archivo con imágenes detectadas - Resultados	76
63. Pívot etiquetado manualmente	81
64. Pívot etiquetado manualmente	82
65. Silobolsa etiquetado manualmente	82
66. Silobolsa etiquetado manualmente	83
67. Silobolsa etiquetado manualmente	83
68. Silobolsa etiquetado manualmente	84
69. Pívot generado con data augmentation	85
70. Pívot generado con data augmentation	86
71. Pívot generado con data augmentation	87
72. Silobolsa generado con data augmentation	88
73. Silobolsa generado con data augmentation	89
74. Subir una imagen para analizar	91
75. Error al subir una archivo de formato invalido	92

76. Inicio de la detección - Loading	92
77. Detección de un archivo con imágenes comprimidas	93
78. Descarga de archivo con imágenes detectadas - Resultados	94

Índice de cuadros

1. Requerimientos de Usuario	46
2. Requerimientos Funcionales de la API	46
3. Requerimientos Funcionales del Frontend	46
4. Requerimientos No Funcionales del Sistema	47
5. Probabilidad de Riesgos Vs Efecto	49
6. Riesgos identificados para el proyecto	50
7. Riesgos según su probabilidad y efecto	51
8. Estrategias de manejo de riesgos - Parte 1	52
9. Estrategias de manejo de riesgos - Parte 2	53
10. Librerías Backend	56
11. Caso de Prueba del Requerimiento de Usuario R-01	69
12. Caso de Prueba del Requerimiento de Usuario R-02	71
13. Caso de Prueba del Requerimiento de Usuario R-03	72
14. Caso de Prueba del Requerimiento de Usuario R-04	73
15. Caso de Prueba del Requerimiento de Usuario R-05	74
16. Caso de Prueba del Requerimiento de Usuario R-06	75
17. Matriz de Trazabilidad	76

Chapter 1

Introducción

1. Introducción

Los avances tecnológicos y la reducción en los costos en la industria han hecho proliferar la industria aeroespacial, incluyendo la disponibilidad de imágenes satelitales. La lista de proveedores de éstas es extensa e incluye no solo al sector público, cuyas imágenes están disponibles para todos los ciudadanos, sino también, más recientemente, al privado, donde se destacan por ejemplo los sitios Satellogic [18] y Kaggle [19].

Las imágenes satelitales poseen una serie de particularidades con respecto a otro tipo de imágenes ordinarias (fotografías por ejemplo). Las imágenes satelitales son matrices de varios millones de píxeles, con una basta variedad de resoluciones de terreno, dependiendo del proveedor y de la banda espectral en la que fue obtenida. La resolución de terreno determina el tamaño de los objetos que pueden detectarse en el. Esas imágenes son multiespectrales y existe un arreglo de píxeles por cada banda espectral utilizada. Estos suelen incluir los usuales colores *rojo, verde y azul* (RGB), pero también *infrarrojos de onda cercana, infrarrojos de onda corta, termal o las bandas pancromáticas* por nombrar algunas. El análisis combinado de estas bandas habilita la construcción y estudio de terreno, tales como vegetación, agua, suelo o termales, que luego facilitan posibles soluciones alrededor de, por ejemplo, el uso del suelo y la detección de suelo cubierto. Las imágenes satelitales son una increíble fuente de información contextual para muchas industrias, sin embargo no es trivial su procesamiento y utilización debido a su alto contenido en información.

Se introduce el concepto de Machine Learning como una disciplina del campo de la Inteligencia Artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones. Deep Learning es una sub-rama de Machine Learning, la cual se basa en usar Redes Neuronales Profundas para encontrar patrones en espacios de grandes dimensiones como son las imágenes satélites. Un modelo de Deep Learning es, a grandes rasgos, un agente flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea. Dentro de lo que a aplicaciones de Machine Learning se refiere, se destaca la popularidad del uso de Computer Vision o Visión por computadoras, como uno de los campos de investigación más populares en este momento y se encuentra en la intersección de muchas materias académicas tales como Ciencias de la Computación (Gráficos, Algoritmos, Teoría, Sistemas, Arquitectura), Matemáticas (Recuperación de Información, Aprendizaje Automático, Probabilidad, Estadística), Ingeniería (Robótica, Procesamiento de Imágenes), Física (Óptica), Biología (neurociencia) y Psicología (ciencia cognitiva).

1.1. Motivación

Las tareas de reconocimiento visual como la clasificación, localización y detección de imágenes son utilidades populares que se logran gracias a la aplicación de modelos de Deep Learning. Y es aquí donde el presente proyecto cobra vida: como punto de partida, se conoce que en el territorio argentino, y en particular en las zonas agrícolas se registran diversas infracciones en cuanto a:

- Abuso de recursos hídricos.
- Evasión impositiva en el rubro agropecuario.
- Deforestaciones ilegales.
- Asentamientos y expropiaciones ilegales.

A su vez, se dispone de un conjunto de imágenes satelitales de público acceso, tales como las obtenidas desde Google Earth [20], que describen diferentes áreas del territorio argentino. Con ellas se plantea hacer uso de la Ciencia de Datos, el análisis estadístico y Machine Learning para implementar un software que permita analizar el contenido de una imagen satelital y devuelva resultados útiles para la búsqueda de infractores, incorporando de esta forma tecnologías novedosas y altamente populares en la industria al ámbito público. En la actualidad ya se pueden ver usos de tecnologías como la desarrollada en este proyecto, para el uso en el ámbito público, tal y como se detalla en esta noticia: detección de piscinas no declaradas [21] y también explotadas en el ámbito privado como lo hace la siguiente compañía que ofrece diferentes servicios de: Agricultura, Urbanismo y Medio Ambiente [22].

1.2. Objetivos

En la siguiente sección se detallan los objetivos a cumplir en el presente proyecto.

1.2.1. Objetivos Principales

- El objetivo principal del presente trabajo es desarrollar un prototipo para la detección de imágenes satelitales en un periodo de tiempo corto. En este proceso de desarrollo, se busca adquirir y aplicar conocimientos sobre Deep Learning e Inteligencia Artificial.
- Ayudar a combatir irregularidades detectándolas fácilmente. En particular detectando pivotes de riego (Figura 1) y silobolsas (Figura 2). Ambos utilizados ampliamente en el rubro agrícola.

1.2.2. Objetivos Particulares

- Investigar sobre Machine Learning y Visión por Computadora, su funcionamiento y sus casos de uso.
- Investigar y estudiar diferentes implementaciones de los modelos de Machine Learning más usados en los lenguajes de programación existentes y mejor se apliquen a nuestras necesidades.
- Evaluar en base a los requerimientos, cuál es la implementación o framework más adecuado para realizar este prototipo.



Figura 1: Riegos por pivote (circular): sistema de riego móviles mediante pivotes que permiten regar grandes superficies. Fuente: [1]



Figura 2: Silobolsas: es un sistema que permite almacenar granos secos de maíz, soja, trigo, girasol y arroz en el propio establecimiento productor, a bajo costo y con óptimas condiciones de calidad. Fuente: [2]

- Configurar un modelo de Machine Learning en una supercomputadora con los recursos necesarios, en un ambiente de desarrollo e implementarlo en un contenedor Docker [23] para que sea portable, lo que permite ser instalado en cualquier computadora.
- Complementar el prototipo con una API [24] y una interfaz gráfica para mejorar su usabilidad.

1.3. Marco de investigación

El presente trabajo esta estructurado según el siguiente Workflow:

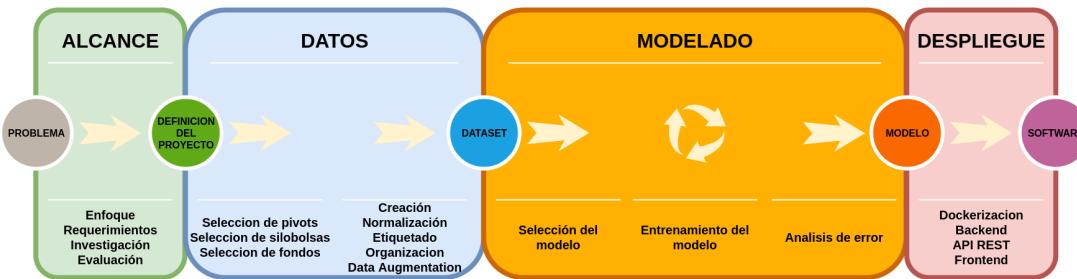


Figura 3: Workflow

- Alcance:** Primero se definió el alcance de nuestro proyecto, para quién estaba enfocado y todo lo necesario para llevarlo a cabo. Se decidió comenzar por una investigación y posterior evaluación de los algoritmos más utilizados para la detección de objetos en tiempo real, con el afán de lograr tiempos de detección reducidos.
- Datos:** Se continuó con la elección de los datos a utilizar, en este caso imágenes de riegos por pivote y silo bolsas con vista satelital. Se siguió con la creación, normalización y etiquetado de un conjunto de datos, para formar el dataset adecuado.
- Modelo:** Luego del análisis entre los diferentes modelos más usados, se optó por una primera elección de YOLOv3 (3), y se procedió con su entrenamiento con los datos previamente seleccionados. Debido a que no se obtuvieron los resultados esperados, se decidió emplear el algoritmo YOLOv5 (3), que además de tener varias mejoras respecto a la versión 3 resultó ser más efectivo para nuestro problema y se obtuvo un mejor rendimiento.
- Despliegue:** Despues se procedió a desarrollar un prototipo de la aplicación con el fin de evaluar la implementación y utilidad de la propuesta. Para eso, el desarrollo se dividió en tres partes:
 - La primera parte consistió en la configuración del Algoritmo YOLOv5 en un entorno virtual contenido en un Docker con los requerimientos y librerías necesarias. Se configuraron 2 entornos virtuales uno para realizar los entrenamientos del algoritmo y dejar corriendo el contenedor Docker con el backend (modelo YOLOv5 + API) y otro entorno con el frontend (página web).

- b) La segunda parte consistió en la programación de la API con la ayuda del framework Flask [25] para el backend. Se implementaron las siguientes funcionalidades:
- 1) Subir una imagen nueva o un archivo comprimido con un conjunto de imágenes para ser analizadas.
 - 2) Consultar la información resultante de cada imagen subida.
 - 3) Consultar la información y los parámetros del modelo utilizado.
 - 4) Otorgar la opción de descargar la/s imagen/es analizada/s junto con los resultados de la predicción.
- c) En la última parte, se desarrolló una aplicación web (Frontend) para que el usuario pueda interactuar con el sistema a través de una interfaz gráfica. Con los siguientes lenguajes de programación: JavaScript, HTML y CSS y el framework Bootstrap [26], se le dio estructura y estilo a la página, dando una simple retroalimentación visual al usuario para indicar los resultados de la detección y del análisis.

Chapter 2

Marco Teórico

2. Principios de Machine Learning

Este trabajo integrador se incluyen campos de la inteligencia artificial, los cuales son Visión por Computadora y Deep Learning o Aprendizaje profundo. Esta sección tiene el propósito de hacer una introducción teórica a ambas ramas de la inteligencia artificial.

2.1. Introducción: Visión por Computadora

Una de las ramas de investigación más populares y utilizada de la Inteligencia Artificial es la Visión por Computadora o también llamada Computer Vision [27] [28] [29]. Este campo se centra en intentar replicar funciones del sistema de visión humano, de forma que permita a las computadoras identificar y procesar objetos en imágenes o videos de forma similar a como los humanos lo hacen. Tiempo atrás, la capacidad de la visión por computadora era muy limitada, pero hoy en día hay muchas tecnologías que utilizan la visión por computadora, entre las cuales se encuentran el reconocimiento de objetos, la detección de sucesos, la reconstrucción de una escena y la restauración de imágenes.

En la actualidad, muchos son los campos que se han visto beneficiados por este conjunto de técnicas. Uno de los más conocidos es el de la robótica por medio de sensores o de cámaras, siendo estos últimos dispositivos idóneos para la aplicación de las estrategias de visión por computadora.

Sin embargo, podemos destacar otros ámbitos capaces de reconocer objetos, por ejemplo, en sistemas de seguridad, seguimiento de objetos o detección de anomalías en piezas fabricadas en una cadena de producción, medicina, etc.

2.2. Fundamentos de Machine Learning

Machine Learning [30], es esencialmente un conjunto de algoritmos diseñado por el ser humano, pero que aprenden de los datos a los cuales son expuestos, dicho de otra manera, un conjunto de datos se usa para entrenar un modelo matemático de modo que cuando vea datos similares en el futuro, sepa reconocerlos. Los modelos, normalmente, toman datos como entrada y luego emiten una predicción de algo de interés. Estos algoritmos son ejecutados típicamente en algún tipo de computadora especial y sirven para resolver problemas de:

- **Clasificación:** Retorna probabilidades de clase o distingue clases. Figura 4.
- **Regresión:** Hace predicciones o estimaciones (en general numéricas). Figura 5.

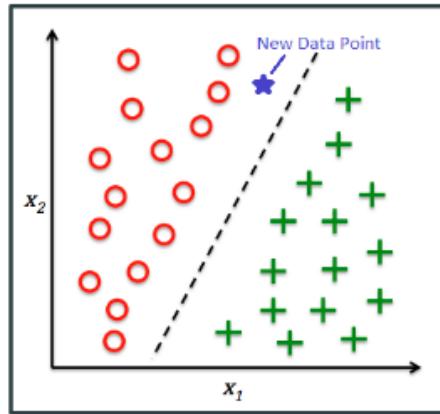


Figura 4: Clasificación. Fuente: [3]

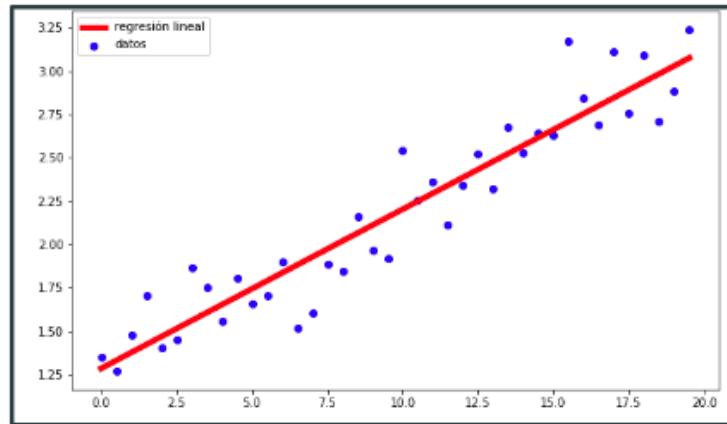


Figura 5: Regresión. Fuente: [3]

El problema surge cuando se intenta resolver un problema de clasificación que no es linealmente separable. Aquí es donde aparecen la Redes Neuronales, con su capacidad de transformar estos problemas en linealmente separables. Figura 6.

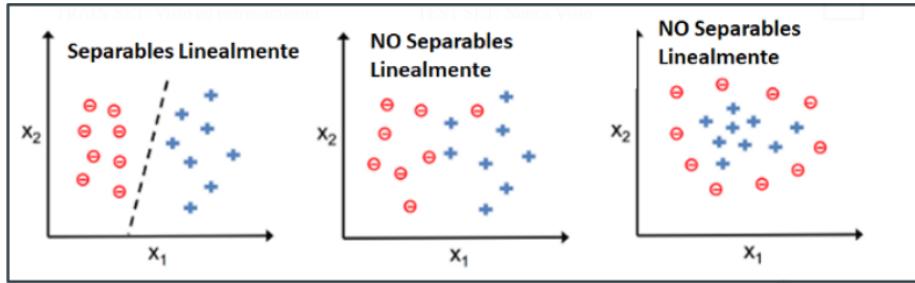


Figura 6: Ejemplo Linealmente Separable. Fuente: [3]

2.2.0.1 Ajuste del modelo: ¿Por qué se entrena un modelo de Machine Learning?

Así como en la vida real a un ser humano se le enseña desde temprana edad a identificar distintos tipos de objetos dentro de su campo de visión, un modelo (o algoritmo) de Machine Learning tiene que aprender a detectar reconocer correctamente.

Para ello se lo entrena a partir de imágenes ya correctamente etiquetadas (ground truth) y, a grandes rasgos, si la predicción es correcta, se le alienta a seguir por ese camino y por el contrario si falla se lo alienta a ir en la dirección opuesta. Esto sería el ajuste con las perillas de una caja negra que viene a representar una abstracción de un modelo de Redes Neuronales, tal y como se representa en la siguiente Figura 7.

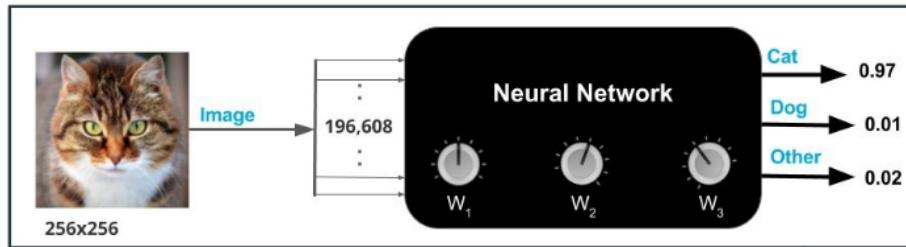


Figura 7: Abstracción de una Red Neuronal. Fuente: [4]

Se entrena el modelo a partir de un conjunto de datos lo suficientemente diverso, de tal manera que el modelo generalice y pueda realizar predicciones a partir de imágenes que nunca “vio” (analizó).

2.3. Redes Neuronales

Las Redes Neuronales Artificiales (ANNs por sus siglas en inglés) son modelos matemáticos concebidos en un principio para estudiar (de cierta manera), el comportamiento del cerebro humano. Estos modelos matemáticos son llamados Redes Neuronales porque fueron inspirados por neuronas biológicas, tal y como formalizaron: McCulloch

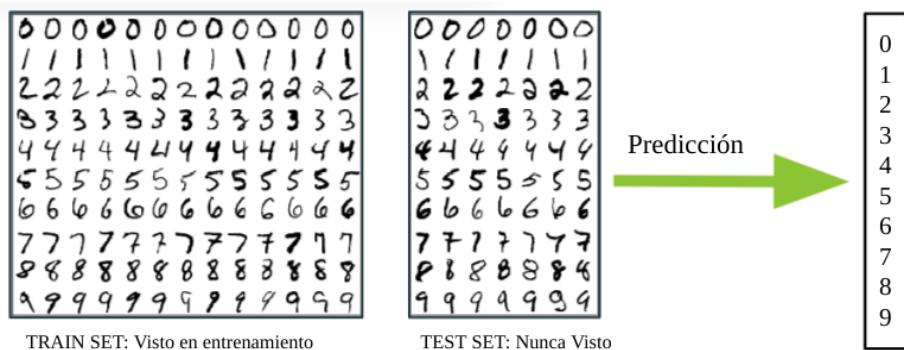


Figura 8: Ejemplo Machine Learning. Fuente: [5]

y Pitts (1943) [31]; y Hebb (1949) [32]. A grandes rasgos, se resume, que las Redes Neuronales Artificiales son formadas por circuitos neuronales y las interconexiones entre neuronas, llamadas sinapsis. Son llamadas sistemas conexionistas porque la información es codificada en los parámetros de las conexiones entre unidades.

En un principio, no hay una división clara entre conexionismo y neurociencia computacional, la principal diferencia es que los sistemas conexionistas se centran en procesos cognitivos de alto nivel, tales como: reconocimiento, memoria, comprensión y razonamiento; en vez de detalles específicos del funcionamiento neuronal. En la década de 1980, el conexionismo experimentó una fuerte revitalización al formalizar una regla general de aprendizaje conocida como Backpropagation LeCun et al.(1988) [33]. Backpropagation, que continua como el “Hoyo en uno” de la investigación conexionista contemporánea, permite a una amplia gama de modelos ANNs aprender una determinada función de mapeo de una entrada a una salida.

Hoy, el conexionismo está mayormente englobado en el aprendizaje profundo (Deep Learning), ya que ha logrado resultados notables en cuanto a avances en diversas aplicaciones. Aún no está claro si las ANNs están limitadas en algún aspecto importante o no (por ejemplo, problemas de olvido catastrófico y ataques adversarios); sin embargo, está claro que los modelos de Deep Learning son lo suficientemente poderosos y flexibles para lidiar con muchos problemas.

En este capítulo, se introducen brevemente conceptos básicos relacionados al presente trabajo. Primero, se presentan las Redes Neuronales y su proceso de dos etapas (inferencia y entrenamiento). Luego, se introducen Clasificadores y otros modelos y técnicas esenciales para entender este trabajo. Finalmente, se presenta la literatura sobre Visión por Computadora y las métricas de evaluación centrales para medir la performance de los modelos de Machine Learning.

2.4. Definición formal de Redes Neuronales

En el aprendizaje supervisado, un dataset (conjunto de datos) de entrenamiento $D = (x_i, y_i)$, representa a una función de mapeo ($F : x_i \rightarrow y_i$) definida por sus observaciones x_i y su correspondiente etiqueta y_i , que son variables independiente e idénticamente distribuidas (i.i.d) de una distribución $P_{x,y}$. Luego, una red Neuronal es entrenada para aprender una función de mapeo f que aproxima a F . El objetivo de una

Red Neuronal es relacionar correctamente las entradas x_i con las salidas y_i , mediante la adaptación de sus parámetros θ . Por ejemplo, en un problema de clasificación de dígitos, x_i consiste en imágenes de dígitos mientras que y_i corresponde a la categoría del dígito.

En dos pasos, un clasificador de redes Neuronales aprende una función de mapeo $y = f(x; \theta)$ y adapta los parámetros θ que minimiza las predicciones del modelo y la verdad absoluta.

El primer paso se llama propagación Feedforward o inferencia. La información de entrada x fluye a través de la red Neuronal, siendo multiplicada por sus computaciones intermedias hacia la salida. En la mayoría de los casos, el mapeo $f(x) : x \rightarrow y$, es una función de activación descripta por $f(x) : g_3 \circ g_2 \circ g_1(x)$ donde g_l son las capas intermedias conectadas en cadena $y = f(x) = g_3(g_2(g_1(x)))$. Las capas intermedias son parametrizadas por un vector de pesos θ_l , que es utilizado para ponderar la entrada antes de ser transformadas por la función de activación. En cada capa oculta, la función de activación transforma la suma ponderada de las entradas en una salida. Las capas ocultas pueden también contener parámetros *bias* que son utilizados usualmente para desplazar la suma ponderada de la entrada. Una capa es comúnmente representada como: $g_l(x) = (\theta_l, x)$ donde θ_l es el parámetro de la capa y g_l es la función de activación de la capa. Una red neuronal comprende de una capa de entrada g_1 , que es la primera capa de la red, una capa de salida g_3 y varias capas intermedias. La última capa, la capa de salida, es donde se espera que esté la predicción. El largo total de esta cadena representa la profundidad del modelo y es de donde proviene el término aprendizaje profundo (Deep Learning).

Los datos de entrenamiento provistos, comprenden de observaciones de F evaluados a diferentes puntos. Cada muestra x_i es acompañada de una etiqueta $y_i = F(x_i)$, que representa la salida deseada para la capa de salida. En conjunto, la etiqueta especifica qué debe ser la capa de salida. Sin embargo, el comportamiento de salida de las capas intermedias no está directamente especificado por los datos de entrenamiento, por eso se llaman capaz ocultas.

El segundo paso, es llamado entrenamiento y consiste en “enviar hacia atrás” (back-propagation), una señal de error sobre los parámetros del modelo (Riedmiller and Braun (1993) [34]). Una función de pérdida (Loss function), es utilizada para computar el error cometido por el modelo en sus predicciones; esto es, la pérdida es alta cuando el modelo está haciendo un trabajo pobre y por el contrario es baja cuando está funcionando bien. La señal de error, es el valor de la diferencia entre las etiquetas predichas y las verdaderas etiquetas deseadas. Luego, este valor de error, es utilizado para ajustar los parámetros del modelo para reducir la discrepancia en la predicción. El gradiente de la función de pérdida con respecto a capas previas, es utilizado como la regla de actualización para ajustar los parámetros de cada capa. El proceso de aprendizaje es repetido hasta converger, por lo que un optimizador iterativamente computa el gradiente sobre lotes aleatoriamente muestreados del set de entrenamiento. Cuando una red neuronal encuentra un set de parámetros “óptimo”, está lista para ser desplegada y comenzar así, a hacer predicciones sobre ejemplos nunca antes observados.

Un amplio rango de características (features), permiten a las ANNs, enseñarle a f a encontrar un mapeo válido entre las entradas y las salidas. Por ejemplo, para evadir limitaciones lineales, funciones no lineales Ψ son utilizadas en capas ocultas en vez de transformaciones lineales, ya que proveen mayores grados de libertad, que habilitan al modelo a “entender” las relaciones no lineales entre los ejemplos x y sus correspondientes etiquetas y . Las transformaciones no lineales, permiten transformar problemas no linealmente separables en linealmente separables. Más específicamente,

las funciones no lineales, permiten al espacio de entrada ser doblado, por lo que éste espacio puede ser dividido en regiones lineales más pequeñas (Pascanu et al.(2013) [35]). La transformación no lineal, la correcta función de perdida y un apropiado número de parámetros en las capas ocultas, permiten a la red neuronal, aproximar cualquier función de mapeo. Éste es el origen del término aproximado universal (Hornik et al. (1989)). Para una introducción detallada de Redes Neuronales, por favor referirse a Hornik et al. (1989) [36].

La función de mapeo aprendida $f(\theta, x)$, es actualizada y evaluada todo el tiempo en aprendizaje continuo. Cuando se evalúa la performance de un clasificador, lo que es juzgado detrás de escena, es el grado de degradación de la función de mapeo relativa a la función original de mapeo F.

2.5. Principios de la Detección de Objetos

Dentro de lo que es Computer Vision, existen diferentes aplicaciones para resolver una amplia gama de problemas:

- **Object Classification - Clasificación de Objetos:** Se determina si una imagen en su totalidad pertenece a una determinada clase.
- **Object Detection - Detección de objetos:** Se localizan y clasifican objetos de diferentes clases en una imagen.
- **Image Segmentation - Segmentación:** consiste en dividir una imagen digital en varias regiones denominadas segmentos. Es un proceso de clasificación por píxel que asigna una categoría a cada pixel de la imagen analizada.

Con la ayuda de la siguiente imagen 9 podemos entender mas fácilmente en qué consisten:

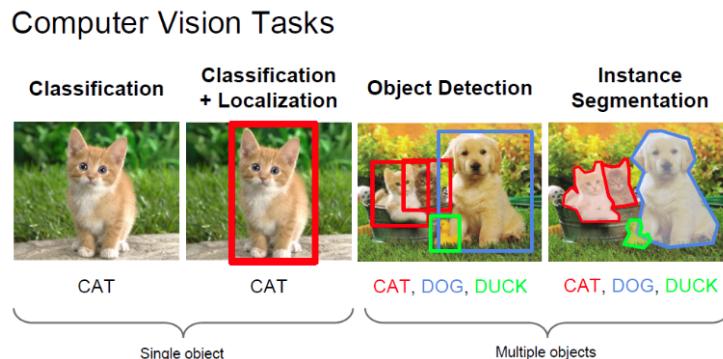


Figura 9: Tareas de Visión por computadora. Fuente: [6]

Este proyecto, optó por la *Detección de objetos (Object Detection)* como herramienta principal para resolver la problemática de la identificación y localización de pivotes de riego y silobolsas en imágenes satélites, dada la naturaleza del problema: Hacer un conteo de la cantidad de estos objetos que se pueden encontrar en una determinada área de suelo.

2.5.1. Conceptos

Sin importar el modelo de Machine Learning subyacente, existen algunos conceptos dentro de lo que es Object Detection que son comunes a ésta aplicación y que ameritan ser explicados. Sin embargo, dado que el modelo seleccionado en este trabajo está basado en YOLO (You Only Look Once) [10], los conceptos a continuación se explican desde el punto de vista en el que son utilizados por YOLO.

2.5.1.1 Bounding box

Cuando se realiza una predicción, el modelo ubica etiquetas (bounding box) en forma de recuadros sobre el objeto [7].

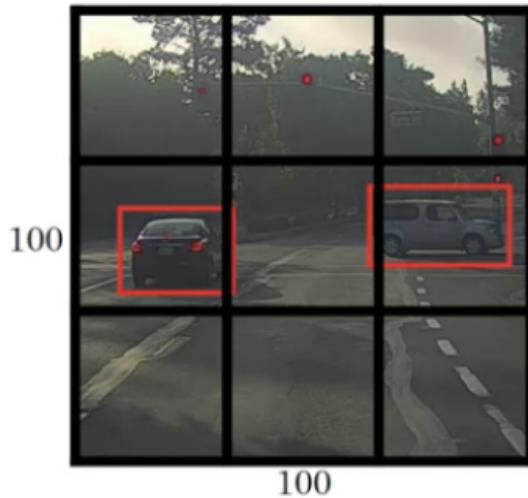


Figura 10: Bounding Box in YOLO. Fuente:[7]

YOLO divide la imagen en celdas, donde cada una de ellas se “encarga” de detectar los objetos cuyo centro de Bounding Box (BB) caiga dentro de ella. Figura 10 .

Cada celda arroja una salida con su predicción. Dicha salida es un vector formado por:

$$Y = \{Pc, Bx, By, Bh.Bw, C1, C2, C3\}$$

con:

- $Pc = 0$ o 1 : probabilidad de que haya un objeto en esa celda.

- Bx, By, Bh, Bw : coordenadas (x, y, altura, ancho) para especificar el cuadro rojo delimitador del objeto asociado a esa celda.
- $C1, C2, C3, \dots, Cn$: Probabilidades de las clases de objetos, por ejemplo puede ser las clases de peatón, coches y motocicletas.

2.5.1.2 Non-max suppression

Es un algoritmo utilizado para que el modelo no detecte 2 veces el mismo objeto. Esto se debe a que YOLO realiza muchas predicciones sobre el mismo objeto, y de esta manera se realiza un filtrado seleccionando solo la mejor (Figura 11).

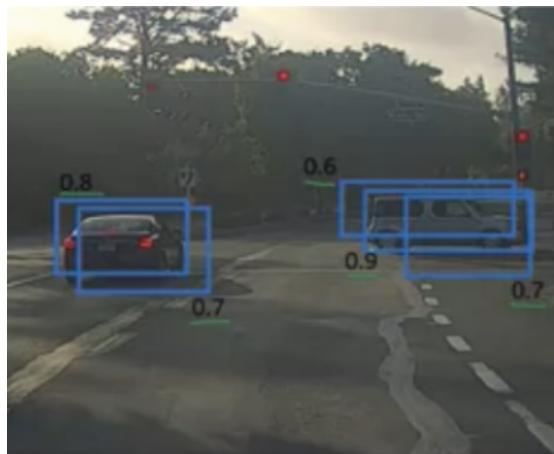


Figura 11: Non Max Supression in YOLO. Fuente:[7]

Procedimiento Non-max supression [7] :

1. Examina las probabilidades asociadas con cada una de las detecciones.
2. Selecciona el BB con el P_c más alto y lo elige como predicción parcial. Se descartan todos los BB con una $P_c \leq 0,6$.
Si hay BB restantes:
3. Descarta los BB remanentes con 0.5 o más de IoU contra el BB seleccionado en el punto anterior.

Para el caso que haya varias clases, hay que correr Non-max una vez por cada clase, chequeando solamente los BB cuya predicción de clase sea la que se está chequeando en ese momento.

2.5.1.3 Anchor Boxes

Si dos o más objetos tienen el centro de su Bounding Box en la misma celda, el detector de objetos puede llegar a confundirse. Solución: Se pre-definen dos o más formas diferentes de cajas de anclaje o anchor boxes y se asocian tantas predicciones como anchor boxes haya. El anchor box que sea más similar a la forma del Bounding Box del objeto, será el asignado en el vector de salida [7] (Figura 12).

La elección de los tamaños y formas de los anchor boxes se puede elegir a mano, o tal vez 5 o 10 formas de anchor box, con una variedad suficiente como para cubrir todos los tipos de objetos que se quiere detectar. También existe la posibilidad de realizar una selección más avanzada, es decir, automatizando la decisión, utilizando un algoritmo de machine learning llamado K-Means.



$$\mathbf{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

(a) Output vector en YOLO

Anchor box 1:



Anchor box 2:



(b) Ejemplos de Anchor Boxes en YOLO

$$\mathbf{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

(c) Output vector con 2 anchor boxes en YOLO

Figura 12: Anchor Boxes en YOLO. Fuente:[7]

2.5.1.4 Intersection Over Union

La intersección sobre la Unión o Intersection over Union (IoU) [8], es una métrica de evaluación utilizada para medir la precisión de un detector de objetos en un conjunto de datos en particular. Sin embargo, hay que tener en cuenta que el algoritmo real utilizado para generar las predicciones no importa. Cualquier algoritmo que proporcione cuadros delimitadores predichos como salida, puede evaluar su rendimiento usando IoU; como por ejemplo, YOLO. Para aplicar IoU (Figura 13) y evaluar un detector de objetos (arbitrario), necesitamos:

- Ground-truth bounding box (es decir, los cuadros delimitadores etiquetados “a mano” del conjunto de prueba que especifican en qué parte de la imagen está nuestro objeto).
- Predicted bounding box, cuadros delimitadores predichos de el modelo.

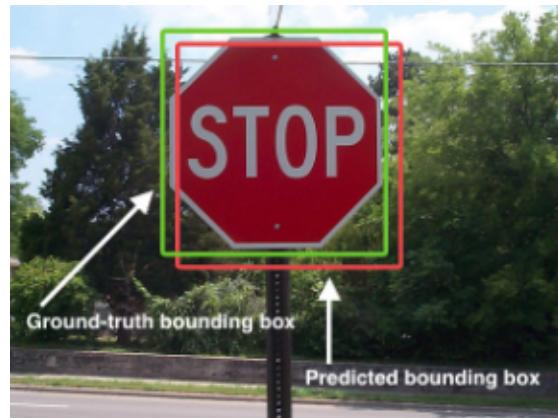


Figura 13: Ground Thruth and Predicted Bounding Box. Fuente:[8]

El objetivo, es calcular la intersección sobre la unión entre estos cuadros delimitadores. Ésta intersección se puede determinar a través de la siguiente fórmula: Figura 14:

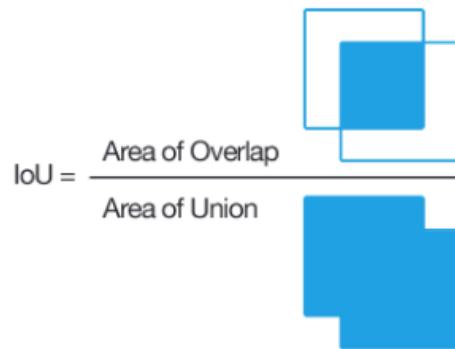


Figura 14: Intersetion over Union formula

- En el numerador se calcula el área de superposición entre el cuadro delimitador predicho y el cuadro delimitador de Ground-truth.
- El denominador es el área de unión, o el área abarcada tanto por el cuadro delimitador predicho, como por el cuadro delimitador de Ground-truth.

Luego, dividir el área de superposición por el área de unión produce el puntaje final: la Intersección Sobre la Unión - IoU. Una IoU mayor a 0.5 normalmente se considera una predicción “buena”.

Cuando se entrena un detector de objetos, se necesita un conjunto de datos, que puede dividirse en dos grupos:

- Un conjunto de entrenamiento utilizado para entrenar el detector de objetos.
- Un conjunto de pruebas para evaluar el detector de objetos.

Ambos conjuntos consistirán en:

- Las imágenes en sí mismas.
- Los cuadros delimitadores asociados con los objetos en la imagen, es decir, las coordenadas (x, y) del objeto en la imagen.

Estos cuadros delimitadores, para ambos conjuntos, están “etiquetados a mano” y, por lo tanto, son llamados “ground-truth”. Su objetivo es tomar las imágenes de entrenamiento + cuadros delimitadores, construir un detector de objetos y luego evaluar su rendimiento en el conjunto de pruebas.

En la realidad, es extremadamente improbable que las coordenadas (x, y) del Predicted bounding box coincidan exactamente con las coordenadas (x, y) del Ground-truth bounding box debido a los parámetros variables del modelo (como tamaño de ventana deslizante, método de extracción de características, etc.). Por ello, se necesita definir una métrica de evaluación que recompense los Predicted bounding box (rojo) por superponerse en gran medida con el Ground-truth (verde).

Como se puede ver en la imagen 15, los Predicted bounding box que se superponen en gran medida con los Ground-truth bounding box tienen puntajes más altos que aquellos con menos superposición. Esto hace que Intersection over Union sea una métrica excelente para evaluar detectores de objetos personalizados.

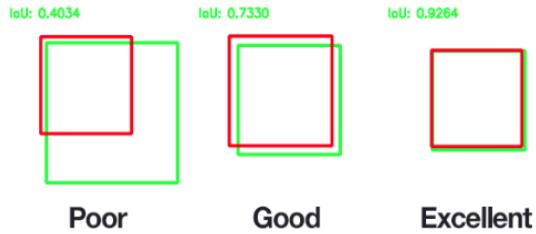


Figura 15: Intersetion over Union examples

2.6. Dataset

En esta sección se detallan las especificaciones de los conjuntos de datos o datasets utilizados para entrenar el modelo YOLO.

2.6.1. Pre-entrenamiento

El modelo se entrenó desde cero (sus pesos establecidos aleatoriamente) usando el famoso dataset: VOC2012 Dataset [9]. Sus características principales son

- Posee 20 clases. Figura 16
- Los datos de entrenamiento/validación contienen 11.530 imágenes comprendiendo 27.450 objetos anotados ROI (región de interés) y 6.929 objetos segmentados (no corresponde para object detection).

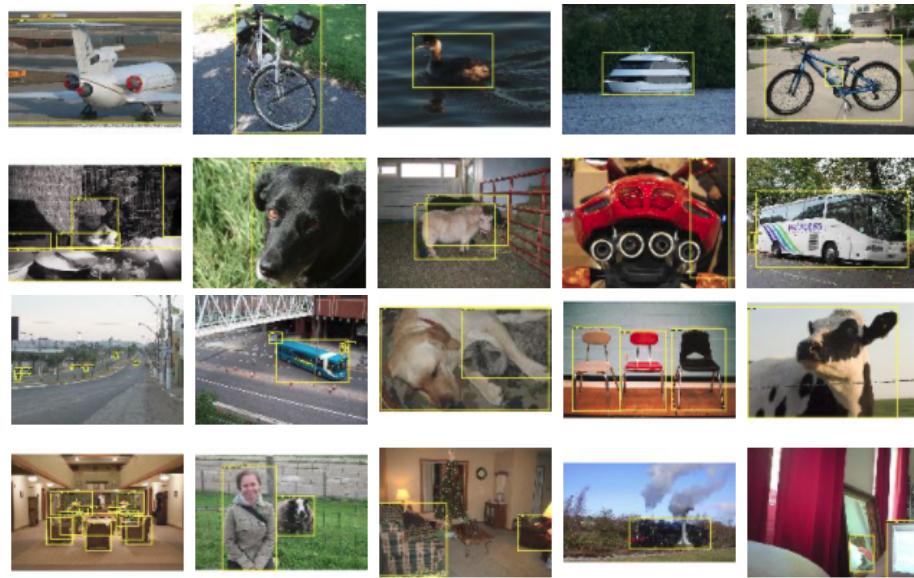


Figura 16: Las 20 clases de VOC2012. Fuente: [9]

2.6.2. Formato de las Etiquetas

Un dataset se compone típicamente de un conjunto de imágenes y un conjunto de etiquetas asociadas a cada una de ellas. Dichas etiquetas respetan un formato estricto, que para el caso del presente proyecto es: “Pascal Visual Object Classes (VOC)”, el cual está caracterizado por:

- Es un archivo XML. Ejemplo de formato VOC2012 [37] :

```

<annotation>
  <folder>Kangaroo</folder>
  <filename>00001.jpg</filename>
  <path>./ Kangaroo/stock -12.jpg</path>
  <source>
    <database>Kangaroo</database>
  </source>
  <size>
    <width>450</width>
    <height>319</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>kangaroo</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
  </object>
</annotation>
```

```
<bndbox>
  <xmin>233</xmin>
  <ymin>89</ymin>
  <xmax>386</xmax>
  <ymax>262</ymax>
</bndbox>
</object>
</annotation>
```

- Se crea un archivo para cada imagen del dataset.
- Las etiquetas están definidas como:

$(xmin-top\ left, ymin-top\ left, xmax-bottom\ right, ymax-bottom\ right)$

2.6.3. Dataset Personalizado

El dataset con el cual se realizó el re-entrenamiento del modelo, está basado en imágenes con vista satelital de pivotes y silobolsas extraídas de diferentes fuentes de acceso libre:

- Usando la herramienta QGIS [38] se obtuvieron imágenes satelitales de las fuentes: Sentinel, Google y Bing Sattelite.
- Google Maps
- Las etiquetas están definidas como: ($xmin-top\ left, ymin-top\ left, xmax-bottom\ right, ymax-bottom\ right$)

Las mismas poseen 3 bandas del espectro visible (RGB) y varían en tamaño, color, relación de aspecto de acuerdo a las diferentes estaciones del año y resolución, es decir, que tan cerca o lejos del nivel del mar se obtuvieron las imágenes. Se seleccionaron de tal manera que su distribución fuera uniforme: tantos objetos en solitario, como en conjunto, así como también balanceado, es decir, cada clase de objetos contenga la relativamente la misma cantidad.

La literatura y la práctica recomiendan un dataset “grande” para obtener resultados aceptables, lo que implica, como regla general:

- Al menos 1500 imágenes por clase.
- Al menos 10000 instancias (objetos etiquetados) por clase.

Sin embargo ante la dificultad de seleccionar a mano un número tan grande de imágenes se optó por aplicar técnicas de Data Augmentation o Aumento de Datos, y de esta manera nos permite aumentar nuestro set de datos de entrenamiento para mejorar la precisión, la generalización, y controlar el overfitting:

- Generación de Imágenes Sintéticas: Utilizando imágenes con diferentes colores y texturas como “fondos” e imágenes de los objetos de interés, se generaron nuevas imágenes sintéticas.
- Rotaciones: Se aplicaron rotaciones sobre las imágenes originales y las sintéticas

Finalmente el dataset quedó compuesto por decenas de miles de imágenes, algunas etiquetadas manualmente y en su gran mayoría imágenes similares a las que se muestran en el Anexo 8:

2.6.3.1 Distribución de Etiquetas

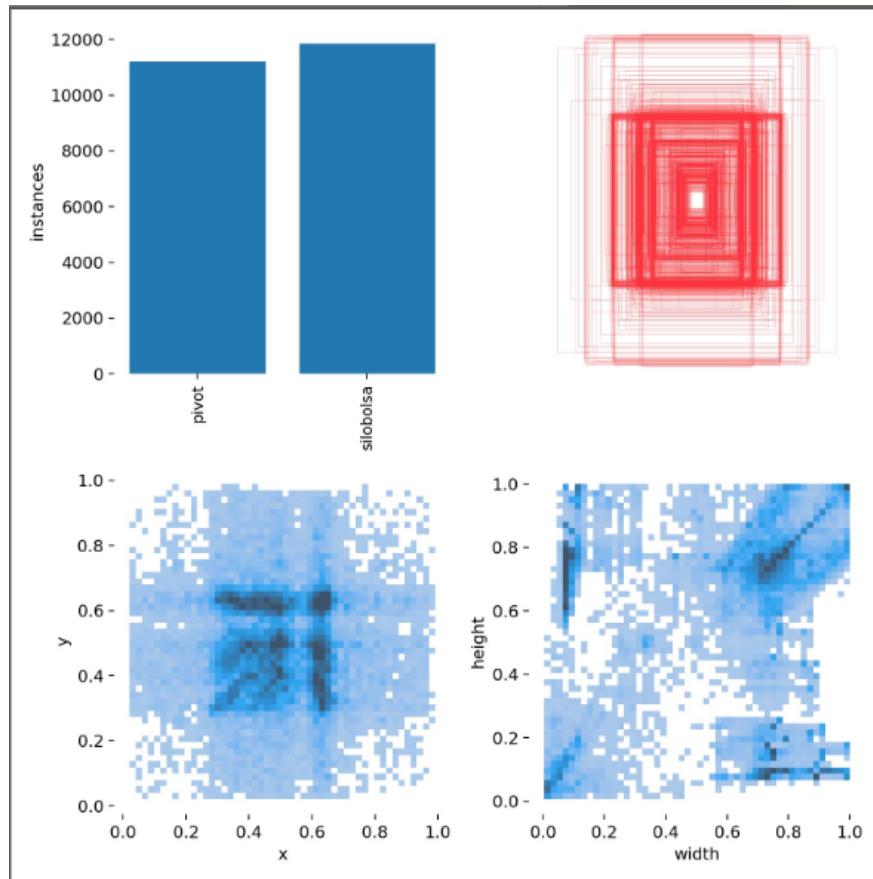


Figura 17: Distribución de Etiquetas

De la anterior Figura 17 se observan las siguientes conclusiones:

- En el gráfico de barras ubicado en la esquina superior izquierda de la figura, se puede ver la distribución de etiquetas por clase. Notar que están relativamente balanceadas: con casi la misma cantidad de pivotes que de silobolsas.
- En la imagen superior derecha , se ven en rojo los diferentes bounding box (BB) que suelen ser en su mayoría cuadrados perfectos (pivotes y silobolsas en diagonal) de gran tamaño o bien muy pequeños, por lo que se concentran en la esquina superior derecha (1,1) e inferior izquierda (0,0).
- También se denota una gran concentración de BB altos y angostos en la esquina superior izquierda (0,1) y anchos y bajos en la esquina inferior derecha (1,0), los cuales se corresponden a etiquetas de silobolsas de grandes dimensiones ubicadas vertical y horizontalmente respectivamente.

- En el gráfico ubicado en la esquina inferior izquierda se observa que el centro de los BB suele concentrarse en el centro de las imágenes y sobre todo evitan las esquinas.

2.6.3.2 Correlograma de Etiquetas

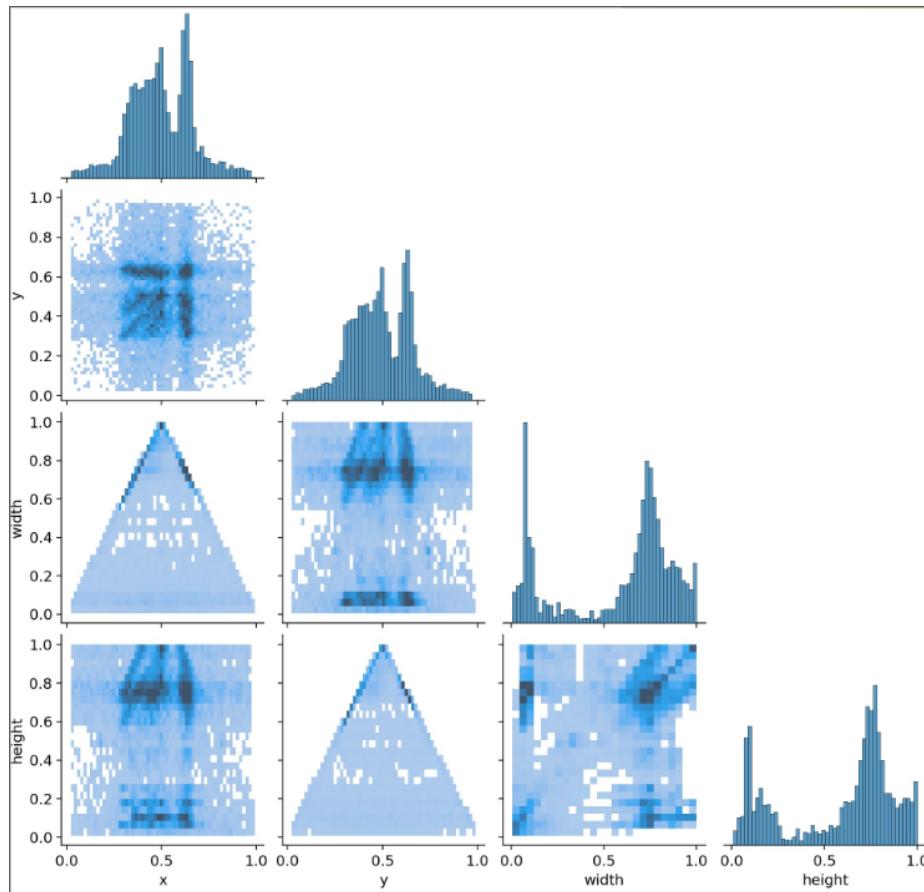


Figura 18: Correlograma de Etiquetas: nos ayudan a visualizar la correlación entre las etiquetas de los objetos.

De la anterior Figura 18 se observan las siguientes conclusiones:

- La distribución de los BB se concentran en el centro de las imágenes, y sobre todo evitan las esquinas (x,y).
- La distribución del ancho y alto de los BB denota que son cuadrados, o bien, rectángulos alargados tanto horizontales como verticales (width,height).
- Los BB ubicados en el centro del eje X de las imágenes tienden a ser extremadamente altas o bajas (x,height).

- Los BB ubicados en el centro del eje Y de las imágenes tienden a ser extremadamente anchas o angostas (y,width).

Chapter 3

Estado del Arte

3. Introducción

En esta sección se tiene el propósito de hacer una distinción entre los principales tipos Detectores de Objetos. Hacer énfasis en el algoritmo de YOLO y sus variantes. Luego se exponen los Benchmarks (puntos de diferencia) entre los diferentes modelos más populares de detección de objetos y análisis del algoritmo de YOLO. La conclusión de la sección se lleva a cabo con la elección del modelo más conveniente para este proyecto.

3.1. Detección de Objetos

Dentro de las arquitecturas Deep Learning se distinguen dos ramas. La primera se conoce como arquitectura de dos fases, generando primero regiones candidatas de la localización del objeto en la imagen y posteriormente clasificando los objetos asignándole una categoría. Se los conoce por detectores de objetos basados en regiones, como los son: Fast R-CNN, Faster R-CNN y R-FCN. Mientras que este tipo de redes son el estado del arte en cuanto a precisión en la localización y clasificación de objetos, tienen la desventaja de ser demasiado lentas para la detección en tiempo real de objetos.

Por otro lado, el segundo tipo de detectores se conocen por tener una arquitectura de una sola fase. Estos realizan una abstracción mayor, donde ven el problema de detección de objetos como un problema de regresión (estimación de las cajas delimitadoras), como lo son SSD y YOLO. Debido a la necesidad de la capacidad de detección de objetos en tiempo real, surgieron las arquitecturas de un paso. Los expertos decidieron reducir el número de fases, logrando en una fase, arquitecturas capaces de inferir directamente a partir de una imagen las coordenadas de las cajas delimitadoras y la verosimilitud por clase. En esta sección, se presentará una de las arquitecturas de un paso usadas para la detección de objetos en este proyecto.

3.2. YOLO: You Only Look Once

Es uno de los algoritmos de detección de objetos en tiempo real más eficaces, que también abarca varias de las mejores ideas a través de toda la literatura de visión por computadora relacionada con la detección de objetos, por lo que resulta ser un Detector de objetos muy eficaz. YOLO [10] plantea la detección de objetos como un único problema de regresión, directamente desde los píxeles de la imagen a las coordenadas de la caja delimitadora y a las probabilidades de cada clase, donde una sola red convolucional predice simultáneamente múltiples cajas delimitadoras y probabilidades de clase para esas cajas. YOLO se entrena en imágenes completas y optimiza directamente el rendimiento de la detección. Este modelo unificado tiene varios beneficios sobre los métodos tradicionales de detección de objetos:

- YOLO es extremadamente rápido. Al enfocar la detección como un problema de regresión, no se necesitan sistemas complejos. Simplemente se ejecuta la red neuronal en una nueva imagen para predecir las detecciones. Además, YOLO alcanza más del doble de la precisión media de otros sistemas en tiempo real.
- YOLO trabaja globalmente sobre la imagen. A diferencia de las técnicas de ventana deslizante y propuestas de regiones, YOLO ve toda la imagen durante el tiempo de entrenamiento y prueba, de manera que implícitamente codifica la información contextual sobre las clases, así como su apariencia.
- YOLO aprende representaciones generalizables de objetos. Cuando se entrena en imágenes naturales, YOLO es altamente generalizable, es menos probable que se descomponga cuando se aplica a entradas inesperadas por lo que supera los métodos de detección más avanzados

A pesar de estas ventajas, YOLO todavía se queda atrás en cuanto a la precisión con respecto a otros sistemas de detección. Aunque puede identificar rápidamente los objetos en las imágenes, tiene problemas para localizar con precisión algunos objetos, especialmente los pequeños. Esto resultó en un problema para este proyecto, debido a que los objetos de interés aparecen en la mayoría de las imágenes como objetos pequeños.

YOLO [10] (Lanzada: 8 Junio 2015) ha ido publicando varias versiones, evolucionando desde la versión inicial, cuya estructura podemos ver en la Figura 19, hasta la última de ellas, YOLOv5 [17] - [17] lanzada el 18 Mayo 2020, la cual es una versión modificada de YOLOv4, pasando por las versiones YOLOv2 [39] (25 Diciembre 2016), YOLOv3 [11] (8 Abril 2018) y YOLOv4 [13] (23 Abril 2020).

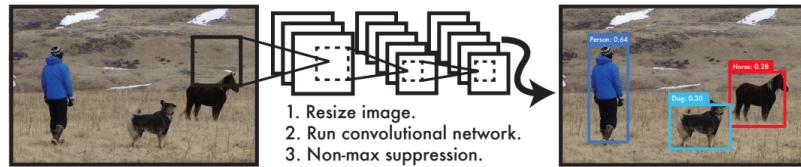


Figura 19: El sistema de detección de YOLO: Procesar imágenes con YOLO es simple y directo. El sistema (1) cambia el tamaño de la imagen de entrada a 448×448 , (2) ejecuta una sola red convolucional en la imagen y (3) establece un umbral para las detecciones resultantes según la confianza del modelo. Fuente: [10]

3.2.1. Funcionamiento de YOLO

La filosofía de las estructuras YOLO consiste en unificar los diferentes componentes de la detección de objetos en una sola red. El funcionamiento global de la inferencia, se puede ver en la siguiente Figura 20.

Los componentes principales para formar el algoritmo de detección de objetos YOLO:

- Intersection over Union

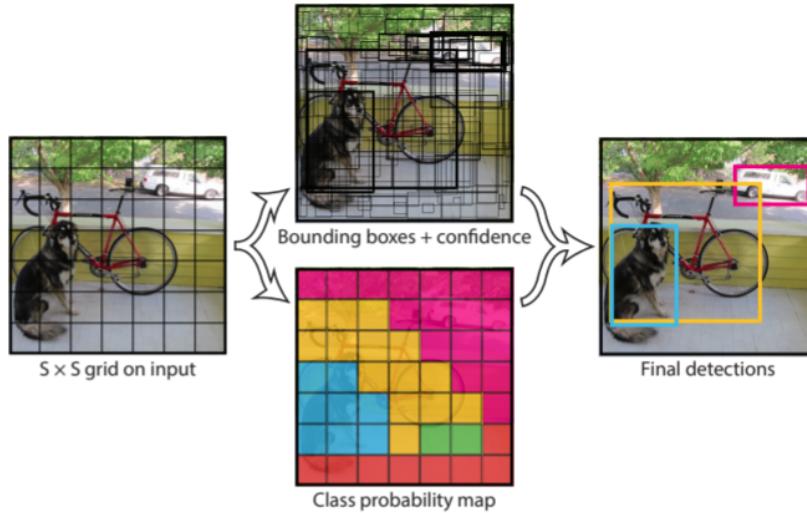


Figura 20: Funcionamiento de YOLO: primero se divide la imagen en cuadrículas más pequeñas, sobre éstas se calculan diferentes cajas delimitadoras o bounding boxes y las probabilidades de las distintas clases que se encuentre en ellas, y finalmente se obtienen las detecciones. Fuente: [10].

- Bounding Box Predictions
- Non-max suppression
- Anchor Boxes

3.2.1.1 Workflow

Para llevar a cabo la detección, utilizando YOLO:

1. Primero, el algoritmo divide la imagen en una cuadrícula de tamaño $S \times S$ celdas (primera imagen de la izquierda Fig.20). Si el centro de un objeto cae en una celda de la cuadrícula, esa celda es la responsable de detectar el objeto.
2. Cada una de las celdas detecta B posibles “bounding boxes” o cajas delimitadoras y calcula el nivel de certidumbre o confianza (probabilidad de la celda P_c) de cada una de las cajas (imagen del centro), que reflejan que tan seguro está el modelo de que esa caja contiene un objeto y qué tan seguro está de que efectivamente la caja se corresponde al objeto que ha detectado, es decir, se calculan $S \times S \times B$ diferentes bounding boxes, la gran mayoría de ellas con un P_c muy bajo.

$$\text{Confianza} = P(\text{Objeto}) * \text{IOU}(\text{truth}, \text{prediction})$$

Si no hay objeto en esa celda, la confianza debería de ser 0, en otro caso debería ser el IOU entre la caja detectada y la leída.

3. Cada una de las cajas delimitadoras está compuesta por 5 predicciones: x , y , w , h y la confianza. Donde (x, y) representa las coordenadas del centro de la caja detectada y (w, h) el ancho y altura respectivamente.
4. Además, cada celda de la cuadrícula también predice C probabilidades condicionales $P(\text{Clase } i \mid \text{Objeto})$, una por cada clase, que representan las probabilidades de que en esa celda se encuentre el objeto de la clase i . Cabe remarcar, que se predice un vector de C probabilidades condicionales por cada celda, independientemente del número de cajas detectadas B .
5. En la inferencia, se multiplican las probabilidades condicionales de cada clase y la confianza de cada caja detectada, obteniendo así las probabilidades específicas de cada clase por cada caja detectada e indicando la probabilidad de que cada clase aparezca en las cajas detectadas y qué tan bien se ajusta dicha caja.

3.2.2. YOLOv3

A continuación se hablará de YOLOv3 [11], que en un principio, fue el modelo elegido para el desarrollar este proyecto como una opción viable de llevarlo a cabo, y con el cual se realizaron las primeras pruebas y evaluaciones de los resultados. Luego se mencionarán brevemente las mejoras que se implementaron en YOLOv3 con respecto a la primera versión YOLO, cuáles mejoras fueron implementadas en YOLOv2 y otras finalmente en YOLOv3.

3.2.2.1 Mejoras

- Normalización de los lotes en las capas convolucionales.
- Clasificador de alta resolución.
- Última capa convolucional con anchor boxes o cajas de anclaje, también llamadas cajas a priori. En vez de predecir directamente las cajas, se predicen los offsets con respecto a unas cajas de anclaje previamente predefinidas. Esto es muy útil por que se pueden definir dichas cajas de anclaje para que se adecuen mejor a la forma de los objetos del dataset.
- Clusters dimensionales: Como se ha mencionado, en muchos problemas de detección, los objetos tienen una forma determinada, por lo que para calcular las K cajas a priori que mejor cubren esta gama de formas en un dataset, se utiliza el algoritmo KMeans[40].
- K-means* es un algoritmo de clasificación no supervisada (clusterización), que agrupa objetos en k grupos, basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster. Se suele usar la distancia cuadrática.
- Predicción directa de la localización

Predicción de cajas delimitadoras

Para la predicción de las cajas delimitadoras utiliza la misma metodología que YOLO9000 [39]. La red predice 4 coordenadas para cada anchor boxes (tx , ty , tw , th), de forma que si la celda de la cuadrícula tiene un offset con respecto a la esquina superior izquierda de la imagen de (cx, xy) y la caja delimitadora a priori tiene anchura (pw , ph), entonces las predicciones finales son Figura 21:

$$\begin{aligned}
 b_x &= \sigma(t_x) + c_x \\
 b_y &= \sigma(t_y) + c_y \\
 b_w &= p_w e^{t_w} \\
 b_h &= p_h e^{t_h}
 \end{aligned}$$

Figura 21: Predicciones de las cajas delimitadoras. Fuente: [11]

En otras palabras, la red predice offset con respecto a cajas de anclaje (anchor boxes). Se puede ver en la siguiente Figura 22 una representación visual de estas ecuaciones. Esto es de mucha utilidad debido a que en muchos problemas de detección, los objetos tienen una forma determinada. Por lo tanto, se definen dichas cajas de anclaje para que se adecuen mejor a la forma de los objetos del dataset. Para calcular las K cajas a priori que mejor cubren esta gama de formas en un dataset, se utiliza el algoritmo K-Means sobre el dataset de entrenamiento.

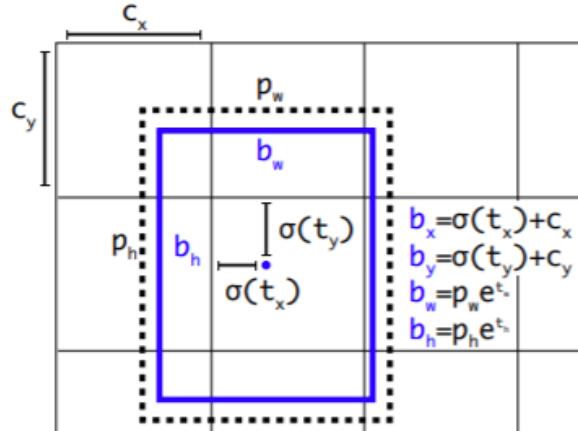


Figura 22: Cajas delimitadoras con dimensiones previas y predicciones. Se predice el ancho y la altura de la caja como offset desde el centro de la caja. Fuente: [11]

Durante el entrenamiento se usa la suma de cuadrados como función de pérdida, de forma que el gradiente viene dado por

$$\hat{t} * -t*$$

donde

$$\hat{t}*$$

es el offset predicho por la red y

$$t*$$

es el offset real.

YOLOv3 predice una puntuación de objeto para cada caja delimitadora usando regresión logística. Si el cuadro delimitador a priori no es la mejor, pero se superpone a un objeto por encima de un umbral (en este caso 0.5), se ignora la predicción. Es decir, después de obtener todas las predicciones, se procede a eliminar las cajas que estén por debajo de un límite, utilizando una medida que ayude a comparar la precisión de las predicciones del modelo ($\text{IoU} > 0.5$).

A las cajas restantes se les aplica un paso de “non-max suppression”, que sirve para eliminar posibles objetos que fueron detectados por duplicado y así dejar únicamente el más exacto de ellos.

Predicción de clases

Cada casilla predice las clases que puede contener la caja delimitadora utilizando la clasificación multi-etiqueta, usando para ello, clasificadores logísticos independientes. Durante el entrenamiento se usa como función de pérdida la entropía cruzada binaria.

Predicciones en diferentes escalas

YOLOv3 realiza predicciones en 3 escalas diferentes. El sistema extrae características de cada una de estas 3 escalas, inspirado y utilizando un concepto similar a las redes piramidales de características o feature pyramid networks [41] (en VGG-16) Figura 23. En la estructura, después de las capas de extracción de características en YOLOv3 se añaden diferentes capas convolucionales, la última de ellas es la encargada de predecir un vector 3D, codificando las predicciones de cajas delimitadoras, la probabilidad de contener un objeto y predicciones de clase. Así, el vector final es de la forma:

$$N * N * (N\text{boxes}(4 + 1 + N\text{classes}))$$

donde $N\text{boxes}$ es el número de cajas predichas en cada escala, 4 se corresponde a los offsets (desplazamiento) de la caja delimitadora, 1 a la predicción del objeto o probabilidad de contener un objeto y $N\text{classes}$ es el número de clases del dataset.

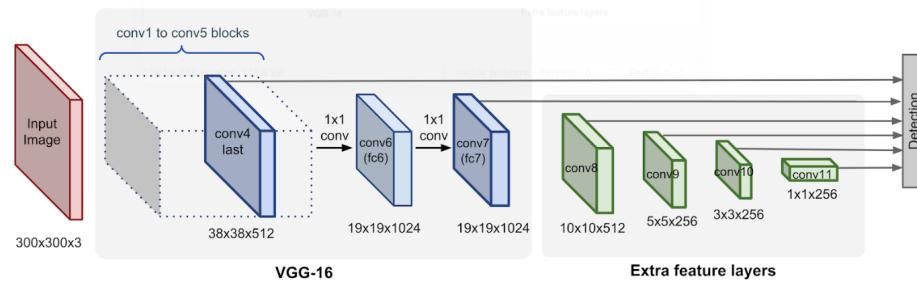


Figura 23: Redes Piramidales de Características. Fuente: [\[12\]](#)

Se toma el feature map o mapa de características de las dos capas anteriores y se aumenta en $2\times$. También se toma un mapa de características de las capas anteriores de

la red y se fusiona con las características aumentadas de capas subsiguientes, usando concatenación. Este método permite obtener información semántica más significativa. Se realiza el mismo proceso una vez más para predecir cajas para la escala final. Así, las predicciones para la tercera escala se benefician de todos los cálculos previos. De esta manera, se ocupa de muchos más candidatos de bounding boxes de diferentes tamaños.

Extractor de características

Se utiliza un enfoque híbrido entre el extractor de características utilizado por YOLOv2 (Darknet-19) [39] y capas residuales. Utiliza capas convolucionales sucesivas de 3×3 y 1×1 , con algunas interconexiones o atajos. Tiene 53 capas convolucionales en total y recibe el nombre de Darknet-53. Se puede ver un resumen de su estructura en la Figura 24.

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	32	1×1	
	64	3×3	
	Residual		128×128
Convolutional	128	$3 \times 3 / 2$	64×64
2x	64	1×1	
	128	3×3	
	Residual		64×64
Convolutional	256	$3 \times 3 / 2$	32×32
8x	128	1×1	
	256	3×3	
	Residual		32×32
Convolutional	512	$3 \times 3 / 2$	16×16
8x	256	1×1	
	512	3×3	
	Residual		16×16
Convolutional	1024	$3 \times 3 / 2$	8×8
4x	512	1×1	
	1024	3×3	
	Residual		8×8
Avgpool		Global	
Connected		1000	
Softmax			

Figura 24: Darknet-53. Fuente: [11]

3.2.2.2 Comparación con otras estructuras

Se puede ver una comparación en los resultados finales con otras estructuras populares para la detección de objetos en la Figura 25.

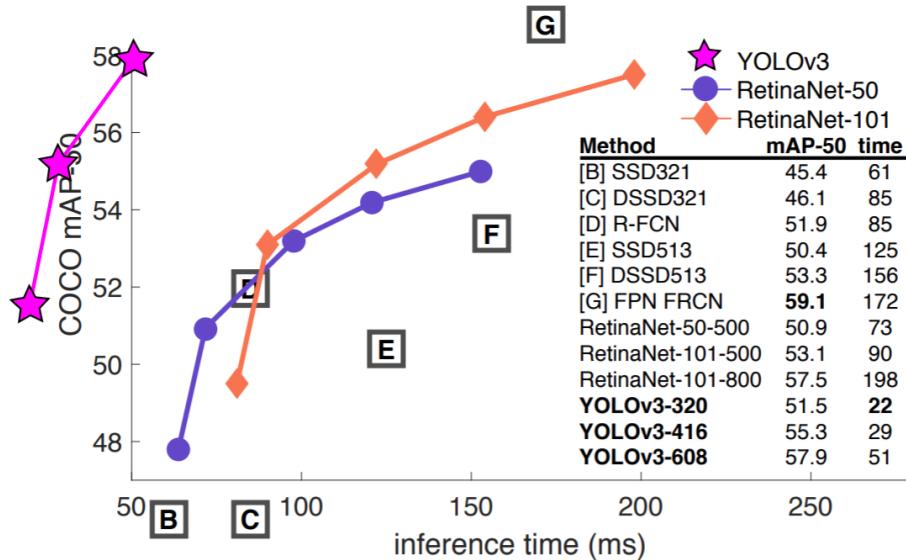


Figura 25: Comparación de distintas estructuras de redes neuronales convolucionales: se observa el balance entre velocidad y precisión para 0.5 IOU, destacando YOLOv3 por su gran velocidad y manteniendo una buena precisión. Fuente[11]

3.2.3. YOLOv4

En esta sección se hablará de YOLOv4 [13], que presenta mejoras considerables con respecto a su predecesor, YOLOv3, tanto en velocidad de inferencia como en precisión de en torno a un 10-12 por ciento, como se observa en la siguiente Figura 26.

Además se encuentra una descripción muy detallada junto a los parámetros que utiliza la red en este enlace [42]. Sin embargo, aunque es una propuesta muy interesante para la detección de objetos en tiempo real por su gran velocidad en cuanto a FPS (Frames per second o Imágenes por segundo) y manteniendo una muy buena precisión con respecto a otros detectores de última generación, al momento de la selección final del modelo a usar, se optó por la elección de YOLOv5, que es la ultima versión disponible de los YOLO, el cual tiene todas las ventajas de YOLOv4, que se mencionarán brevemente a continuación.

3.2.3.1 Mejoras

Nueva Arquitectura

La arquitectura de YOLOv4 se puede dividir en 3 partes:

1. Backbone: Basado en la CSPDarknet53
2. Neck: Basado en SPP y PAN
3. Head: YOLOv3

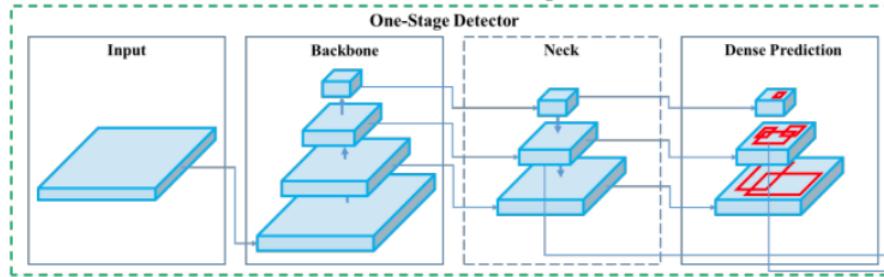


Figura 26: Principales componentes de un Detector de Objetos moderno de una etapa. Fuente: [13]

Esta nueva arquitectura trae nuevos beneficios con respecto a la versión anterior de YOLO, entre ellas se pueden mencionar que:

- Reduce en gran medida la cantidad de cálculo y mejorar la velocidad de inferencia y la precisión.
- Se ocupa de los siguientes tres problemas: Fortalecimiento de la capacidad de aprendizaje de una Red Neuronal Convolutacional (CNN). Eliminación de cuellos de botella computacionales. Reducir los costos de memoria.
- Elimina el requerimiento de imagen de entrada de tamaño fijo.
- Otorga robustez frente a deformaciones de objetos.

- Aumenta el flujo de información propagada a través de la red.
- Mejora la predicción de la anchor boxes.

Bag of Freebies (BoF)

Bolsa de regalos o Bag of Freebies se denomina al conjunto de técnicas o métodos que cambian la estrategia de entrenamiento o el costo de entrenamiento para mejorar la precisión del modelo. Permiten al detector de objetos ser más preciso sin incrementar el costo computacional, un ejemplo clásico es: data augmentation o el aumento de datos, es una técnica que permite crear variaciones artificiales sobre imágenes del dataset para expandir el conjunto de imágenes existentes y aumenta la capacidad de generalización del modelo.

Para ello se hacen distorsiones fotométricas como: cambiar el brillo, la saturación, el contraste y el ruido o hacer distorsiones geométricas de una imagen, como rotar, recortar, etc. Estas técnicas son un claro ejemplo de un BoF, y ayudan a mejorar la precisión del detector.

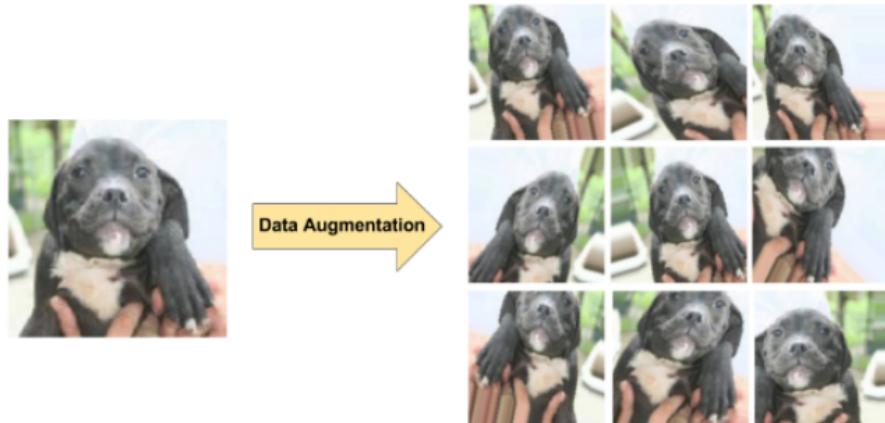


Figura 27: Ejemplo de Aumento de datos [14]

1. CutMix: En lugar de simplemente eliminar píxeles como en Cutout Figura 28, se reemplazan las regiones eliminadas con un parche de otra imagen. Los parches agregados mejoran la capacidad de localización al requerir que el modelo identifique el objeto desde una vista parcial. No quedan píxeles no informativos durante el entrenamiento, lo que hace que el entrenamiento sea más eficiente.
2. Mosaic: Combina 4 imágenes de entrenamiento en una sola, en ciertas proporciones (en lugar de solo dos en CutMix) (Figura 29). Permite que el modelo aprenda a identificar objetos a una escala menor de lo normal y mejorar así la precisión de sus modelos hasta en un 10 %. Combina clases que pueden no verse juntas en su conjunto de entrenamiento.



Figura 28: CutMix



Figura 29: Mosaic representa un nuevo método de aumento de datos. [13]

3. Class label smoothing o Suavizado de etiquetas: Es una técnica de regularización que aborda ambos problemas: overfitting (sobre ajuste) y exceso de confianza. Suavizar las etiquetas evita que la red se vuelva demasiado confiada. Ayuda al modelo a entrenarse en torno a datos mal etiquetados y, en consecuencia, mejorará su solidez y rendimiento. Matemáticamente, se altera el vector Y de ground truth por un factor α :

$$y_k^{LS} = y_k(1 - \alpha) + \alpha/K$$

Bag of Specials (BoS)

Como lo mencionan los autores *Bag of Specials* contiene diferentes módulos y módulos de procesamiento posterior que solo aumentan el costo de inferencia en una pequeña cantidad, pero pueden mejorar drásticamente la precisión del detector de objetos.

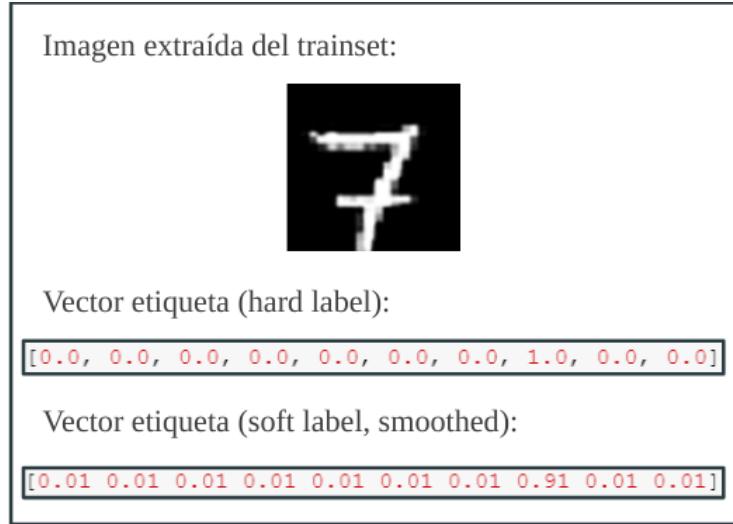


Figura 30: Suavizado de Etiquetas

Hay muchos módulos diferentes presentes, pero en ésta parte solo se hace foco en uno de éstos métodos selectivos que se muestran en la Figura 31 y se responden el por qué de sus beneficios:

1. Módulo de Atención Espacial o Spatial Attention Model (SAM)

Se aplica una máscara de atención espacial a las características de entrada, para obtener un features map refinado: Se destacan así las características más relevantes creadas por las capas convolucionales y se remueven las menos importantes. Producido una mejora en la clasificación y la detección, con un costo computacional bajo. En YOLOv4, una versión modificada de SAM [13] es usada, en la cual no se aplica el maximum ni el average pooling.

3.2.4. YOLOv5

Poco después del lanzamiento de YOLOv4, se presentó YOLOv5 [17], aunque sin documentación oficial al día de la redacción de este informe. Sin embargo, se conoce, por medio de foros y repositorios que lo han implementado, que cuenta las siguientes mejoras:

3.2.4.1 Mejoras

1. Es una implementación de YOLOv4 utilizando un framework en Python llamado Pytorch.
2. Posee muchas optimizaciones algorítmicas y aritméticas.
3. Se reduce sustancialmente el costo computacional de entrenamiento.

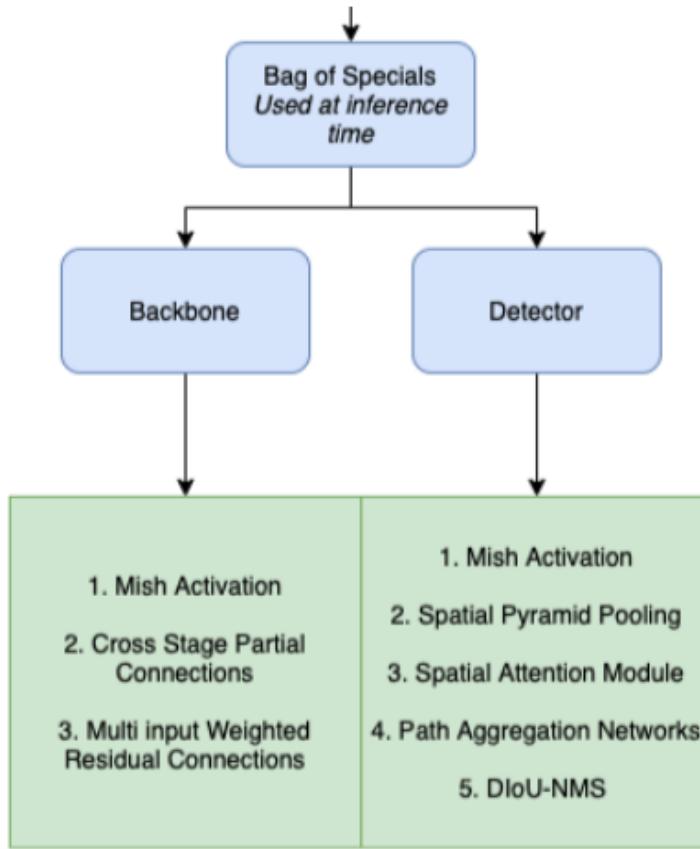


Figura 31: Diferentes métodos presentes en la Bag of Specials. Fuente:[15]

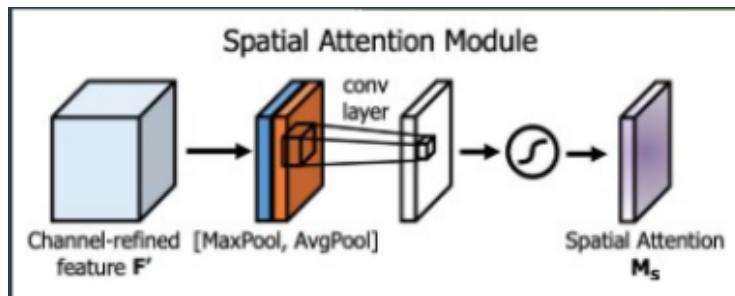


Figura 32: Módulo de Atención Espacial. Fuente: [16]

3.2.4.2 Memoria y FLOPS

Cuando se habla de un modelo que es “liviano” se habla de su uso de la *memoria RAM* necesaria para entrenarlo. La cantidad necesaria viene dada por la cantidad de

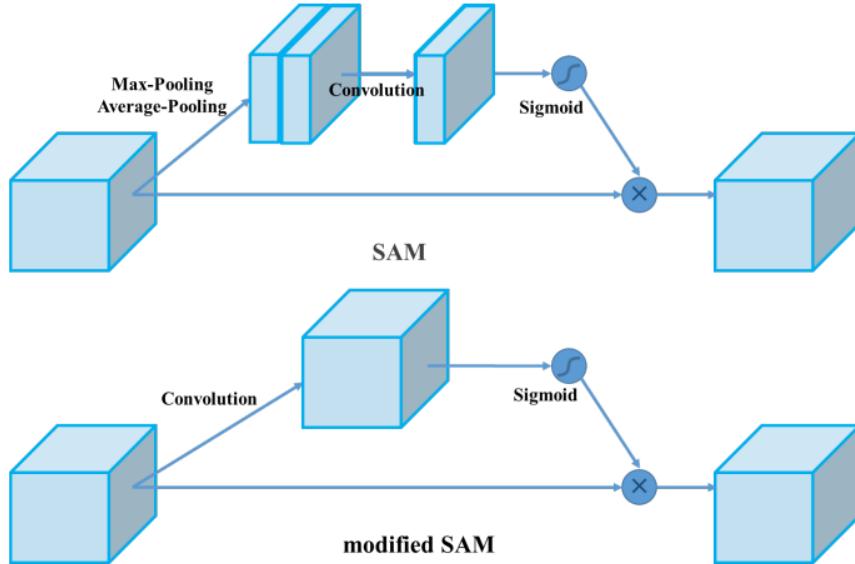


Figura 33: Módulo de Atención Espacial Modificado. Fuente: [13]

parámetros que el modelo seleccionado tiene para optimizar. Si se mantiene constante la arquitectura, a mayor cantidad de parámetros, se suelen obtener mejores predicciones. Si aumenta la cantidad de parámetros a optimizar, entonces también aumentan la cantidad de operaciones necesarias por iteración, lo que se traduce en mayor uso de CPU+GPU y por ende en tiempo de entrenamiento. La unidad de medida para estas operaciones se conoce como *FLOPs* (operaciones de punto flotante).

Comparación: Memoria y FLOPS

En la siguiente tabla se ve claramente cómo el cambio de arquitectura en YOLO v5l (large), con una cantidad de parámetros incluso menor que YOLOv3, arroja resultados más performantes (mayor FPS) y precisos (mayor AP -precisión promedio-) que éste último.

3.2.4.3 Comparación de modelos de YOLOv5

En la siguiente Figura (35), se puede observar como YOLOv5 hace un excelente rendimiento en la detección de objetos, especialmente el YOLOv5S, con más velocidad. El objetivo es producir un modelo de detector de objetos que sea muy eficaz (eje Y) en relación con su tiempo de inferencia (eje X). Los resultados preliminares muestran que YOLOv5 lo hace muy bien con éste fin, en comparación con otras técnicas de vanguardia.

Model	AP_{val}	AP_{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}	params	FLOPS
YOLOv5s	36.6	36.6	55.8	2.1ms	476	7.5M	13.2B
YOLOv5m	43.4	43.4	62.4	3.0ms	333	21.8M	39.4B
YOLOv5l	46.6	46.7	65.4	3.9ms	256	47.8M	88.1B
YOLOv5x	48.4	48.4	66.9	6.1ms	164	89.0M	166.4B
YOLOv3-SPP	45.6	45.5	65.2	4.5ms	222	63.0M	118.0B

Figura 34: Arquitecturas: Memoria y FLOPS. Fuente: [17]

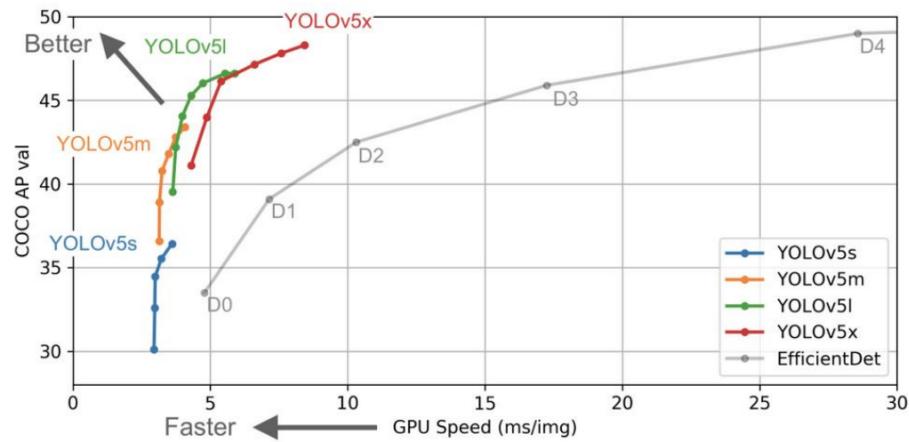


Figura 35: YOLOv5 - Rendimiento. Fuente: [17]

En resumen, YOLOv5 obtiene la mayor parte de su mejora de rendimiento de los procedimientos de entrenamiento de PyTorch, mientras que la arquitectura del modelo permanece muy cercana a YOLOv4.

Chapter 4

Análisis del Diseño del Sistema

4. Introducción

Luego de haber adquirido una cantidad de conocimiento sobre Detección de Objetos en general y sobre el algoritmo de YOLO en particular, el presente capítulo investiga los problemas que tiene el área agropecuaria y la necesidad de detección y de localización de las silobolsas y mecanismos de riego por pivotes. Para esto, se realizó una entrevista con las partes interesadas o stakeholders en hacer uso de la aplicación resultante de este trabajo final, donde manifestaron su importancia. Luego se procede a analizar si las soluciones propuestas a las cuales se llegaron en la entrevista representan un caso de uso de Detección de Objetos. Por último, se especifican Componentes, Requerimientos y Comportamiento del sistema a implementar.

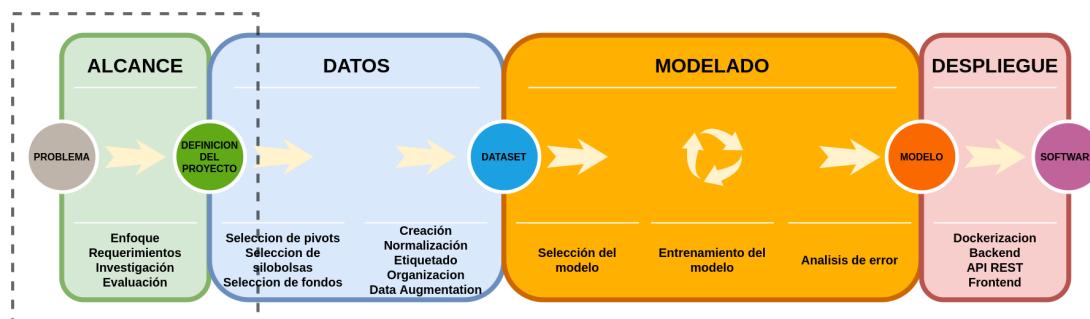


Figura 36: Workflow: Alcance

4.1. Entrevistas

Las partes interesadas, como conocedores del área de las imágenes satelitales y del sector agropecuario, fueron entrevistadas con el objetivo de:

- Entender las primeras especificaciones para el prototipo planteado en la sección 5.
- Comprender mejor los problemas del sistema de riego que se usa en el campo, como así también el almacenamiento de los cereales, legumbres, entre otros.
- Hacer uso de éste prototipo y sus beneficios como parte de la solución de estos problemas en un futuro.

De la charla, se plantearon los siguientes problemas:

- Deforestaciones ilegales.
- Asentamientos ilegales y expropiaciones.
- Evasión impositivas en el rubro agropecuario.
- Abuso de los recursos hídricos.

La solución propuesta que surgió de la entrevista fue:

- Debido a la fuerte influencia de algoritmos de Machine Learning de detección de objetos, se propuso hacer uso de ésta tecnología para desarrollar una aplicación que provea lo siguiente:
 - Detectar múltiples objetos, es este caso, las silobolsas y los pivotes de riego.
 - Dar la posición X e Y del objeto en la imagen (o su centro) y dibujar un rectángulo a su alrededor.
 - La posibilidad de detectar “a tiempo”. Ésta es una característica que se debe tener en cuenta si por ejemplo se quiere hacer detección en tiempo real sobre vídeo. Ésto es posible gracias a la velocidad y precisión para la inferencia que los detectores de objetos proveen.

4.2. Motivación e Importancia del proyecto

Ante las preocupaciones anteriormente planteadas, se propone una solución para satisfacer esas necesidades que no estaban cubiertas, por lo cual se enumeran a continuación las motivaciones que se presentaron para llevarlo a cabo:

1. Elevar la inteligencia de la toma de acciones automáticas, por ejemplo, en un sistema de Dirección General de Rentas.
2. Incorporar tecnologías novedosas y altamente populares en la industria al ámbito Agropecuario público.
3. Ayudar a combatir irregularidades o deforestaciones detectándolas en tiempo real.

4.3. Casos de uso de Machine Learning - Detección de objetos

Desde el éxito de Machine Learning y el nacimiento de numerosas aplicaciones que la usan, como el reconocimiento facial en tiempo real (celulares), sugerencias de compras, autos autónomos, entre otras, están, hoy en día, todas al alcance de cualquiera. Es una tecnología que está madura y es adoptada en la vida cotidiana.

Un claro ejemplo del alcance de la Inteligencia Artificial, es el motor de Google con aplicaciones de sistemas de inteligencia artificial y aprendizaje profundo, el cual ofrece respuestas adecuadas, tales como son las traducciones automáticas, las recomendaciones de videos, el reconocimiento de imágenes y su posterior geo-localización, y hasta la detección de SPAM (correo electrónico masivo no solicitado) en el correo electrónico.

Otro impacto de esta tecnología, es en el futuro de la conducción con Piloto automático en coches Tesla [43], por ejemplo, que vienen de serie con una combinación de hardware y software avanzado capaz de ofrecer las funciones de Piloto automático y capacidades de conducción autónoma total basado en una IA (Inteligencia Artificial) para la visión y la planificación.

4.4. Funcionalidades

Luego de la entrevista y de la evaluación del caso de uso, se consideran necesarias las siguientes funcionalidades para el prototipo:

- El prototipo debe consistir de una API con una interfaz amigable para que cualquier usuario pueda procesar una imagen.
- El prototipo debe ser capaz que recibir imágenes de diferentes tamaños y características, ya sea por unidad o por cantidad en archivos comprimidos. A las que se le va aplicar el algoritmo de detección.
- El prototipo debe proporcionar como resultado del procesamiento, una imagen con los objetos detectados remarcados con cuadros delimitadores y su localización en coordenadas (x,y).

Este prototipo se realiza con una API de REST, o API de RESTful [24], la cual es una interfaz de programación de aplicaciones (API o API web), que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful.

En otras palabras, las APIs le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

4.5. Definición de los Componentes

Con el fin de implementar las funcionalidades descritas en la sección anterior, en primer lugar se definen los componentes del prototipo en su totalidad.

1. **Modelo de Machine Learning:** Se necesita de un Modelo de Machine Learning que sea re-entrenable para detectar las clases de pivotes y de silobolsas. Que tenga un gran rendimiento en la detección de objetos para así cumplir con las exigencias en los cortos tiempos de inferencia deseados. Además, que sea liviano, para poder ser desplegado en un amplia variedad de sistemas, desde embebidos hasta clústeres.
2. **Desarrollo de una API:** Se requiere de un proceso de continuo funcionamiento que sea capaz de recibir consultas, procesar los datos recibidos, disparar la ejecución del modelo para la inferencia y enviar mensajes al usuario final, ya sea con los resultados de detección de su consulta o bien notificar sobre la ocurrencia de algún error. En éste sentido, optar por el desarrollo de un API REST fue lo más óptimo, debido a su facilidad en la implementación y la amplia documentación disponible, sea cual sea el framework y/o lenguaje seleccionado para su desarrollo.
3. **Desarrollo de una GUI:** Se precisa de una interfaz gráfica, que sea amigable para cualquier usuario y que sea capaz de generar consultas con los requerimientos del usuario y comunicarlas con el backend para su posterior procesamiento. A su vez, debe ser capaz de transmitir la información recibida por el backend de manera clara al usuario. Es por eso, que se optó por desarrollar un Frontend, representado por una página web, una vez más, beneficiándose en la rapidez en su implementación y el acceso a documentación detallada, sea cual sea el framework y/o lenguaje seleccionado para su desarrollo.
4. **Diagrama general de la arquitectura**

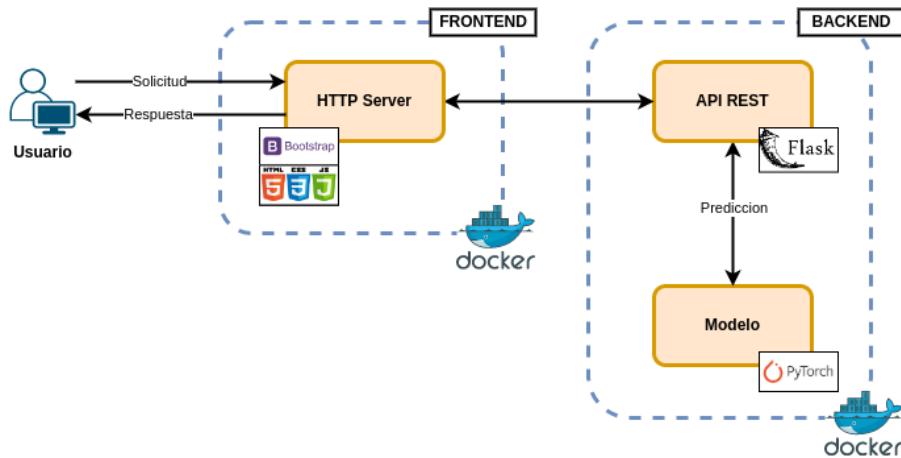


Figura 37: Arquitectura del Software Implementado

4.5.1. Selección de los Componentes

- Elección del Modelo de Machine Learning:** Luego del análisis y comparaciones entre los rendimientos de los diferentes modelos de ML más usados, se optó por una primera elección de YOLOv3 [11]. Se realizaron pruebas con diferentes sub-versiones del mismo (ej.: tiny, medium, large), donde variaba la profundidad de su arquitectura de red neuronal, afectando la precisión y los tiempos en las inferencias. Para éste modelo, se tomó mucho tiempo para re-entrenarlo y tratar de mejorar su precisión: aplicando diferentes técnicas de data augmentation sobre el dataset; modificando los hiper-parámetros del modelo; y hasta modificando secciones de su arquitectura de red neuronal, pero sin obtener los resultados esperados. Se tenían problemas para detectar objetos muy pequeños (silobolsas) junto a los objetos grandes (pivotes), o con diferentes texturas y colores, por lo que se optó por continuar con una versión más reciente del mismo: YOLOv5 [17], que contaba con mejoras respecto a la versión 3, gracias a que se trata de un proyecto muy activo y en continuo desarrollo. Finalmente, éste fue el modelo elegido luego de realizar varias pruebas y mejoras con cambios en la configuración, modificando hiper parámetros con el fin de obtener mejores resultados, otorgando un excelente rendimiento y precisión en la detección de objetos, en comparación a sus alternativas (otras versiones de YOLO u otros modelos de ML).
- Elección del lenguaje de programación:** El framework empleado para el entrenamiento y en el cual está basado el proyecto oficial de YOLOv5 [17], es Python + Pytorch. En la selección del lenguaje con el cual se desarrollarían los scripts de diversa índole que fueran surgiendo en el desarrollo del proyecto, los lenguajes de programación conocidos, tales como C/C++ o JAVA, fueron descartados, ante la opción de Python, dada la mayor familiaridad y experiencia con dicho lenguaje, y a la amplia variedad de librerías orientadas al tratamiento de imágenes y datos que éste ofrece.
- Elección para el desarrollo de una API:** Para el desarrollo del Backend, se optó por modificar y ampliar las funcionalidades que el repositorio oficial de

YOLOv5 brindaba. El mismo, esta basado en Python+Flask, y la familiaridad y el conocimiento previo con dichos lenguaje y framework, la hacían una gran opción sobre la cual seguir desarrollando. Originalmente, ofrecía una implementación muy limitada para la clase de uso que se pretendía dar, y es por estos motivos, que se decidió trabajar sobre ésta base y agregarle las funcionalidades necesarias para cumplir con los objetivos del proyecto. En cuanto al servidor web necesario para desplegar y exponer un servicio continuo de comunicación, se optó por desarrollar el software en JavaScript. Ésta elección fue fundada en el hecho tener experiencia previa trabajando con dicho lenguaje, y las bondades que éste ofrece, para el desarrollo de éste tipo de aplicaciones web, frente a sus alternativas (Python o JAVA). JavaScript es, entonces, el código cliente que invoca al servidor, pero para que sea accesible a través de Internet es necesario continuar con el desarrollo de una API REST que permita llamar las funciones con métodos HTML. La herramienta elegida para proseguir con esa tarea es Node.js, que es un entorno en tiempo de ejecución multiplataforma para la capa del servidor basado en JavaScript, sobre el cual se tiene experiencia previa y se dispone de mucha documentación detallada online. Node.js es, a su vez, un entorno controlado por eventos, diseñado para crear aplicaciones escalables, permitiendo establecer y gestionar múltiples conexiones al mismo tiempo y flexibilidad para aplicaciones web y APIs, convirtiéndolo en el candidato más óptimo para el desarrollo de esta funcionalidad de la API.

4. **Elección para el desarrollo de una GUI:** La API, públicamente expuesta a Internet, necesita un Frontend, o una interfaz web, que permita acceder a ella. Su funcionalidad consiste en obtener información de la API y mostrarla de forma entendible para el usuario, como también, obtener datos ingresados por el mismo y enviarlos para su procesamiento a la API. Para lograr eso, frente a la falta de experiencia específica previa en el desarrollo de GUIs, a excepción de HTML básico, se decidió utilizar Bootstrap [26], por su facilidad de uso y amplia disponibilidad de documentación y guías online. Usando ésta herramienta, las páginas adaptan su contenido de forma dinámica a medida que reciben entradas del usuario, en vez de descargar páginas nuevas de un servidor. Además, proporciona plantillas para CSS y HTML, que facilitan la colocación y el diseño de la página, las fuentes, los botones y los elementos de navegación, de modo que implementar con ella un diseño web moderno resulta sencillo.

4.6. Definición de los Requerimientos

Para especificar el funcionamiento general del sistema, se va a hacer uso del lenguaje de modelado UML. Si bien se trata del desarrollo de un sistema distribuido con una variedad de protocolos y tecnologías, el prototipo final es un producto de software, por lo cual un modelado en SysML [44] no se consideró.

Antes de plantear los requerimientos, primero es necesario definir un diagrama de casos de uso, como se ve en la Figura 38. En el diagrama se puede observar que el objetivo principal es brindarle al usuario un sistema de detección de objetos sobre imágenes satelitales.

4.6.1. Diagrama de casos de uso

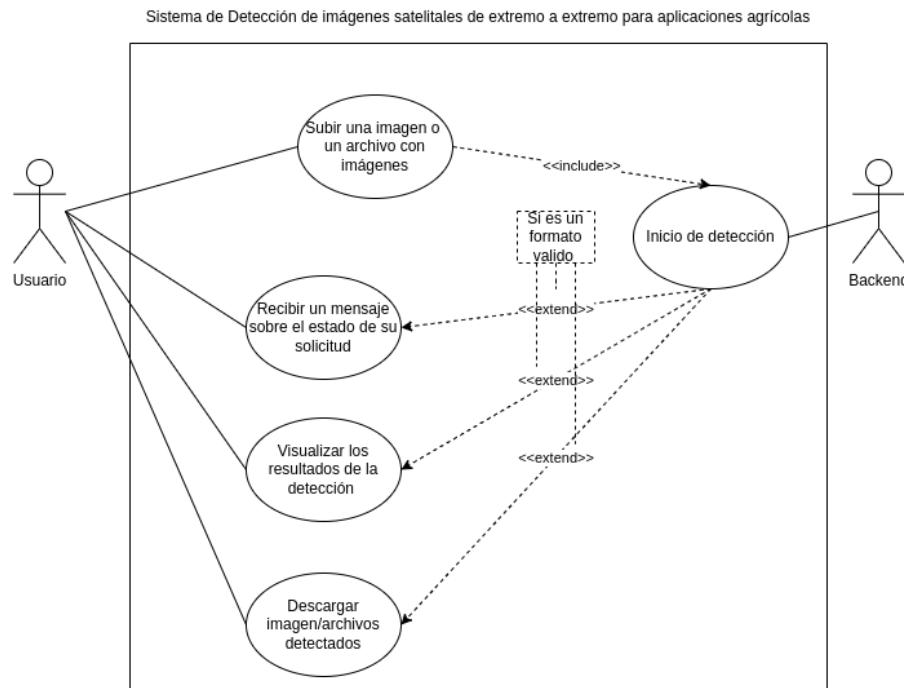


Figura 38: Diagrama de Caso de usos

Con el diagrama de caso de uso elaborado se pueden definir los siguientes requerimientos de usuario, definidos en el Cuadro 1:

R-01	Cargar una imagen nueva al sistema.
R-02	Verificar la presencia de una imagen cargada.
R-03	En caso de que sea un formato erróneo, mostrar un mensaje del error y debe ser posible volver a cargar una nueva imagen.
R-04	Visualizar todas las imágenes cargadas y posteriormente un historial de las analizadas.
R-05	Analizar la imagen cargada, obtener información de la misma y el sistema debe informar si se encontró algún objeto o no.
R-06	Descargar la imagen ya analizada.

Cuadro 1: Requerimientos de Usuario

Luego es posible definir los requerimientos funcionales para cada uno de los componentes descritos anteriormente: la API y la interfaz gráfica. Los cuales se encuentran en los Cuadros 2 y 3 respectivamente. Los Requerimientos No funcionales se definieron en el Cuadro 4.

RF-API-01	Disponer de un endpoint para acceder a los servicios de detección.
RF-API-02	Soportar una imagen y/o un archivo comprimido con imágenes.
RF-API-03	Realizar la detección de silobolsas y pivotes de riego sobre los archivos recibidos.
RF-API-04	Retornar un/archivos con la/s imagen/es con los objetos detectados.
RF-API-05	Retornar información adicional sobre la detección y el sistema donde fue ejecutado.

Cuadro 2: Requerimientos Funcionales de la API

RF-FE-01	Mostrar información sobre el sistema y sus desarrolladores.
RF-FE-02	Admitir carga de archivos.
RF-FE-03	Mostrar la imagen con los objetos detectados.
RF-FE-04	Mostrar información de la detección.
RF-FE-05	Visualizar historial de las detecciones anteriores.
RF-FE-06	Permitir la descarga de archivos.

Cuadro 3: Requerimientos Funcionales del Frontend

RNF-01	Tiempo de detección menor a 100ms por imagen.
RNF-02	Presentar una interfaz amigable para cualquier usuario.
RNF-03	Soportar diferentes formatos de archivos.
RNF-04	Capacidad de ser desplegado fácilmente en diferentes sistemas
RNF-05	La API debe desarrollarse en Node.js y para la interfaz gráfica se debe usar CSS, HTML y JavaScript.

Cuadro 4: Requerimientos No Funcionales del Sistema

Para comprender mejor cual es la relación estructural entre las diferentes tecnologías empleadas en el prototipo, se ofrece un diagrama de componentes, realizado en la Figura 39. Se distinguen 3 componentes principales:

- El Frontend donde se encuentra nuestra página web.
- La aplicación - API REST.
- El Modelo de Machine Learning - YOLOv5, donde los últimos dos componentes conforman el Backend.

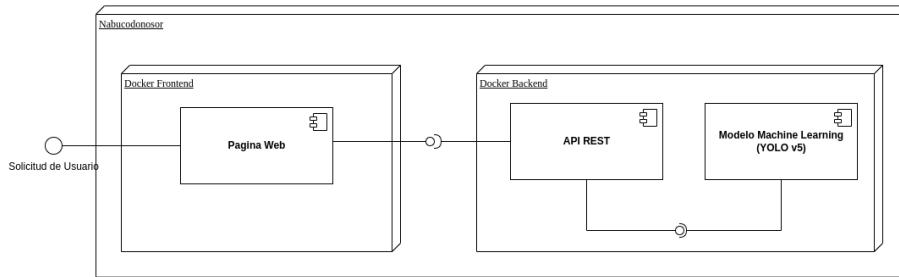


Figura 39: Diagrama de Componentes

4.7. Definición del Comportamiento

Para entender cómo interactúan los componentes diagramados, se emplea un diagrama de secuencia en la Figura 40, en el cual se muestra el flujo de mensajes para el caso de subir una imagen a analizar y su posterior descarga, como también para verificar la validez del formato de una imagen y en caso de ser válido, proveer los resultados de las detecciones correspondientes.

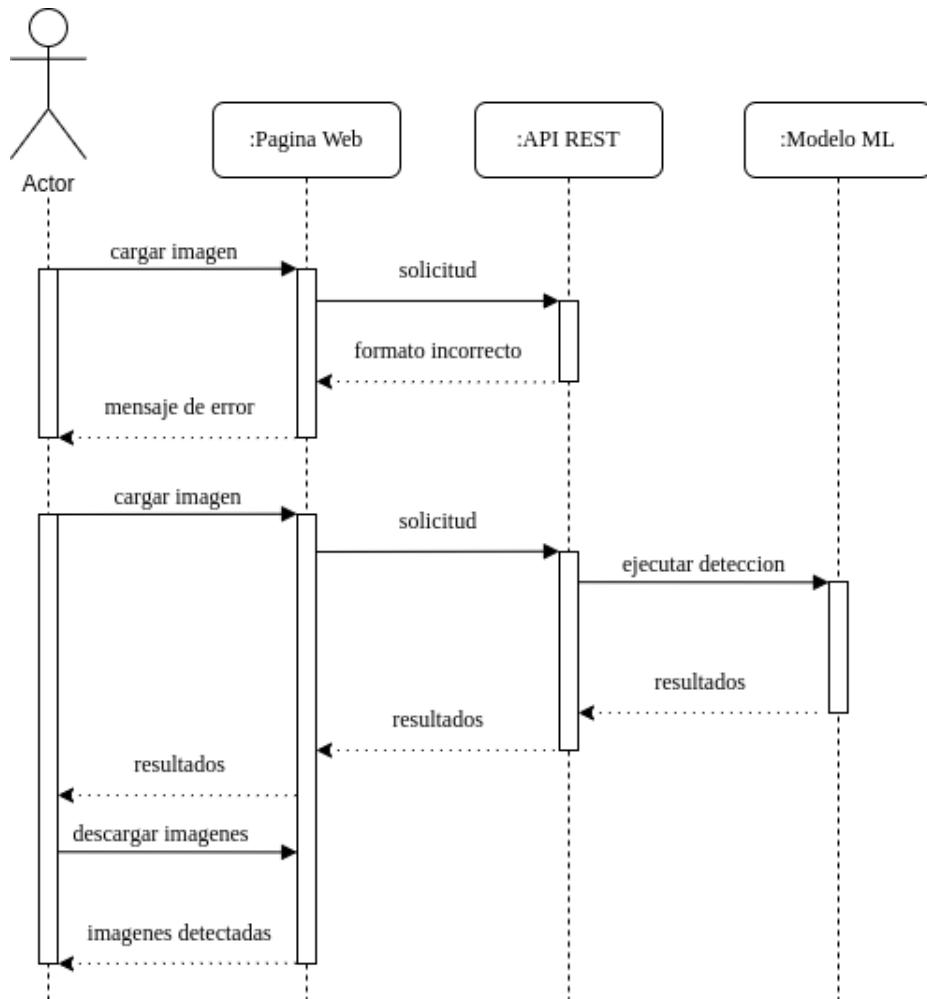


Figura 40: Diagrama de Secuencia

4.8. Riesgos

La evaluación de riesgos de un proyecto busca detectar posibles eventos no deseados que puedan perjudicar o amenazar al proyecto, con el objetivo de minimizarlos o controlar su impacto sobre el proyecto.

En la presente sección se establecen diferentes criterios para identificar posibles riesgos, asignar prioridades a los mismos, evaluar su probabilidad y encontrar estrategias que permitan resolver los mismos o minimizar sus efectos.

4.8.1. Criterios

Los riesgos se clasifican según dos criterios: su probabilidad y la gravedad en el caso de su ocurrencia. A continuación, figuran las probabilidades y gravedades que se tuvieron en cuenta según Ian Sommerville en su libro Software Engineering [45]:

1. Riesgo muy bajo (menor a 10 %)
2. Riesgo bajo (10 - 25 %)
3. Riesgo moderado (25 - 50 %)
4. Riesgo alto (50 - 75 %)
5. Riesgo muy alto (mayor a 75 %)

1. Efectos insignificantes
2. Efectos tolerables
3. Efectos graves
4. Efectos catastróficos

La combinación de ambos criterios permite detectar y priorizar los riesgos teniendo en cuenta ambos factores, y se consideran a los riesgos dentro del área verde como los de menor prioridad, los riesgos en el área amarilla como de prioridad intermedia y los riesgos en el área roja como de prioridad alta, como se muestra en el Cuadro 5.

Probabilidad-Efecto	Insignificante	Tolerable	Grave	Catastrófico
Muy baja				
Baja				
Moderada				
Alta				
Muy alta				

Cuadro 5: Probabilidad de Riesgos Vs Efecto

4.8.2. Identificación de Riesgos

En la etapa de la gestión de riesgos, se buscan los posibles riesgos relacionados al proyecto escritos en el Cuadro 6, distinguiendo las siguientes categorías :

- **Riesgos de tecnología:** Riesgos relacionados la hardware o software con el que se está desarrollando el presente proyecto.

- **Riesgos personales:** Relacionados con lo personal del equipo de desarrollo.
- **Riesgos de requerimientos:** Surgen de modificaciones de los requerimientos.
- **Riesgos de estimación:** relacionados a la estimación de recursos requeridos o estimación de tiempo para el desarrollo del proyecto.

Código	Descripción	Tipo de Riesgo
Riesgo-01	Problemas de funcionamiento o fuera de servicio de la Máquina donde se desarrolla el proyecto	Tecnológico
Riesgo-02	Incompatibilidad de versiones o librerías instaladas en la máquina de desarrollo con los frameworks usados.	Tecnológico
Riesgo-03	Demoras durante el desarrollo del proyecto debido a trabajo u otros inconvenientes personales.	Personales y Estimación
Riesgo-04	Errores en la implementación de los frameworks elegidos.	Tecnológico, Requerimientos y Estimación
Riesgo-05	Falta de conocimiento o documentación para el uso de los frameworks, del software o los lenguajes de programación elegidos.	Estimación
Riesgo-06	Rechazo de la implementación por parte del usuario final.	Estimación

Cuadro 6: Riesgos identificados para el proyecto

4.8.3. Análisis de Riesgos

En el Cuadro 7 se ordenaron los riesgos según su importancia, probabilidad y efecto.

Código	Riesgo	Probabilidad	Efecto	Importancia
Riesgo-02	Incompatibilidad de versiones o librerías instaladas en la máquina de desarrollo con los frameworks usados.	Moderada	Insignificante	Prioridad baja
Riesgo-01	Problemas de funcionamiento o fuera de servicio de la Máquina donde se desarrolla el proyecto	Baja	Grave	Prioridad Intermedia
Riesgo-05	Falta de conocimiento o documentación para el uso de los frameworks, del software o los lenguajes de programación elegidos.	Alta	Tolerable	Prioridad Intermedia
Riesgo-04	Errores en la implementación de los frameworks elegidos.	Alta	Tolerable	Prioridad intermedia
Riesgo-03	Demoras durante el desarrollo del proyecto debido a trabajo u otros inconvenientes personales.	Alta	Grave	Prioridad alta
Riesgo-06	Rechazo de la implementación por parte del usuario final.	Alta	Grave	Prioridad alta

Cuadro 7: Riesgos según su probabilidad y efecto

4.8.4. Planificación de Riesgos

En la presente sección se describen las estrategias para manejar cada riesgo identificado anteriormente, y se detallan en los Cuadros 8 y 9.

Código	Riesgo	Estrategias de Manejo del riesgo
Riesgo-01	Problemas de funcionamiento o fuera de servicio de la Máquina donde se desarrolla el proyecto	<ul style="list-style-type: none"> ■ Implementación de un respaldo de información en la nube con repositorios de Github. ■ Realización de la operación <i>git push</i> al repositorio diariamente o ante algún cambio.
Riesgo-02	Incompatibilidad de versiones o librerías instaladas en la máquina de desarrollo con los frameworks usados.	<ul style="list-style-type: none"> ■ Lectura cuidadosa de tutoriales de instalación y pre-requisitos. ■ Uso de Docker para ganar mayor flexibilidad con versiones de librerías o lenguajes de programación.
Riesgo-03	Demoras durante el desarrollo del proyecto debido a trabajo u otros inconvenientes personales.	<ul style="list-style-type: none"> ■ Comunicación fluida con director y codirector del Proyecto. ■ Reducción de obligaciones laborales y extracurriculares. ■ Definición clara del alcance del proyecto para minimizar las posibles demoras de tiempo y trabajo.
Riesgo-04	Errores en la implementación de los frameworks elegidos.	<ul style="list-style-type: none"> ■ Implementación de métodos descritos en la documentación o páginas de seguimientos, como también foros oficiales. ■ Modificación o adaptación del requerimiento y su implementación.

Cuadro 8: Estrategias de manejo de riesgos - Parte 1

Código	Riesgo	Estrategias de Manejo del riesgo
Riesgo-05	Falta de conocimiento o documentación para el uso de los frameworks, del software o los lenguajes de programación elegidos.	<ul style="list-style-type: none"> ■ Lectura de tutoriales y realización de cursos cortos para lograr una introducción en temas cuyo desconocimiento impida el avance del proyecto. ■ Empleo y modificación/adaptación de ejemplos previstos para el aprendizaje. ■ Búsqueda de explicaciones adicionales en foros de usuarios como StackOverflow [46]. ■ Consultas a programadores o profesionales experimentados en los campos en cuestión.
Riesgo-06	Rechazo de la implementación por parte del usuario final.	<ul style="list-style-type: none"> ■ Explicación de la tecnología empleada y proporcionar manual de usuario para su uso. ■ Prueba de piloto para identificar mejoras en cuanto a la usabilidad y posibilidad de mejoras en un futuro.

Cuadro 9: Estrategias de manejo de riesgos - Parte 2

4.9. Control de Versiones

Git es una herramienta para mejorar la colaboración entre varios programadores. Usar un repositorio permite tener acceso remoto al proyecto desde cualquier equipo, teniendo así una copia de seguridad del proyecto completo en la nube. A su vez, se mantiene un historial de versiones y en caso de ser necesario, volver a una versión anterior o comparar las diferencias entre dos versiones distintas.

Para el control de versiones, se utilizó un repositorio de Github disponible bajo la URL: https://github.com/gianfrancob/detector_pivotes_silobolsas

Chapter 5

Implementación

5. Introducción

En este capítulo se describen los procedimientos de implementación que se realizaron, expuestos de forma cronológica, es decir a medida que se avanzaba con el proyecto, comenzando con la configuración del ambiente de trabajo, después el armado del dataset a partir de las imágenes/datos obtenidos, para luego entrenar el modelo con la correcta monitorización de los resultados y por último, desplegar el software final.

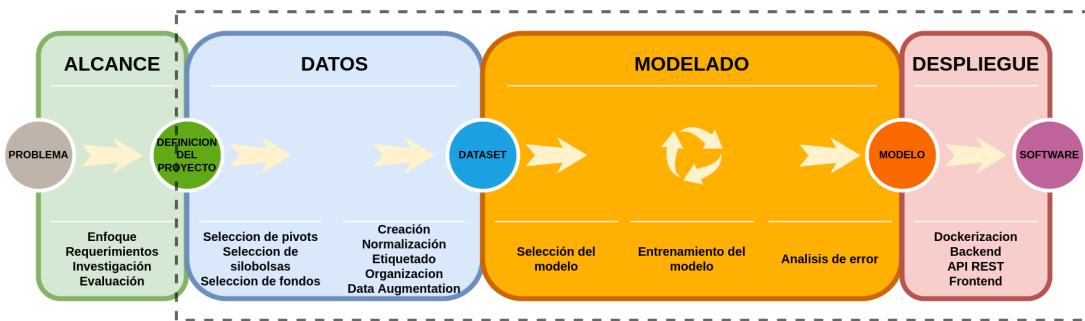


Figura 41: Workflow: Implementación

5.1. Entornos de trabajo

Para el desarrollo de este proyecto, se utilizaron dos computadoras con el sistema operativo Linux instalado (distintas distribuciones de Ubuntu), con conexión a internet, además de la súper computadora de la UNC (Universidad Nacional de Córdoba) Nabucodonosor (*Nabu*) [47] para el entrenamiento. A continuación se detalla el hardware que conforma el clúster:

- **CPU:** Xeon E5-2680v2 (x2)
- **RAM:** 64 GB
- **GPU:** NVIDIA GTX 1080Ti (3x)
- **Almacenamiento:** 3 TiB + 240 GiB

El proyecto se realizó en gran parte de forma remota, usando las computadoras personales para conectarse a *Nabu* usando un túnel SSH [48], lo que brindaba la posibilidad de disponer de un gran poder de computo sin largas colas de espera ni limitaciones de tiempo, desde cualquier lugar. Sin embargo, surgieron algunas limitaciones, sobre todo

relacionadas a la ausencia de una interfaz gráfica al trabajar desde una consola; por lo que para ciertas tareas fue necesario usar las PCs locales. A continuación se detallan las tareas del proyecto que se llevaron a cabo en cada entorno:

Entorno PC

- Generación del dataset básico: Usando software para acceder a fuentes de imágenes satelitales *QGIS*, *Google Earth*, editor de imágenes *Gimp* y etiquetador manual *YOLO Annotation Tool*.
- Programación de código fuente: Se utilizó un editor de texto que modificara remotamente los archivos en *Nabu*.
- Prueba del BackEnd: Se hizo uso de navegadores web y del software *Postman* para probar el backend, haciendo uso del *re-direccionamiento de puertos*.
- Diseño y prueba de FrontEnd: Se accedía a la página web (el frontend de este proyecto) haciendo uso del re-direccionamiento de puertos.

Entorno Nabucodonosor: *Las siguientes tareas se ejecutaban en Nabu a través de una consola conectada por un túnel SSH.*

- Generación del dataset aumentado: Usando un script en Python personalizado se extendió el dataset básico “manual” con el objetivo de mejorar la performance del modelo.
- Entrenamiento del modelo: Se ejecutó el entrenamiento del modelo usando diferentes versiones de dataset e hiper-parámetros hasta lograr resultados satisfactorios.
- Ejecución de la Detección: Todas las detecciones se ejecutan en el clúster, ya sea para pruebas manuales como para ejecuciones disparadas por el Backend.
- Ejecución del BackEnd: Se dejó corriendo un contenedor Docker con el backend basado en Python usando el framework *Flask* el cual recibía consultas con imágenes para ser analizadas por el modelo.
- Ejecución del FrontEnd: Se dejó corriendo otro contenedor Docker con el frontend basado en JavaScript usando el framework *Bootstrap* y HTML. El mismo dejaba corriendo un servidor con el frontend para ser accedido por cualquiera (teóricamente) y éste realizaba las consultas al backend con el modelo en ejecución. *Aclaración:* Debido a limitaciones por seguridad del clúster este frontend solo fue accedido desde las PCs usando *re-direccionamiento de puertos*.

5.1.1. Configuración Nabucodonosor

Lo principal para comenzar a desarrollar y trabajar sobre un modelo de Machine Learning es disponer del firmware y software para hacer uso del poder de computo, en particular, de las tarjetas gráficas. Para eso, un enfoque práctico es hacer uso de un contenedor *Docker* sobre el cual instalar librerías y dependencias que permitan realizar el desarrollo sin alterar el estado *por defecto* del clúster para que el resto de los usuarios del mismo no se vean afectados. De esta forma se trabajó de manera encapsulada dentro de diferentes contenedores Docker obtenidos en *DockerHub* [49], configurados de tal manera que cumplieran diferentes propósitos particulares.

5.1.1.1 Machine Learning & Backend Docker

Posee la siguiente configuración:

- Driver Nvidia: Para hacer uso de los núcleos *CUDA* [50] disponibles en las 3 tarjetas gráficas.
- Python 3: Se le sumaron a las librerías pre-instaladas en el contenedor las siguientes librerías y frameworks:
 - Anaconda: Framework basado en Python orientado a Machine Learning.
 - Librerías:

Descripción o Uso	Librerías
Para Machine Learning	<ul style="list-style-type: none"> ◦ matplotlib ◦ numpy ◦ opencv ◦ Pillow ◦ PyYAML ◦ requests ◦ scipy ◦ torch ◦ torchvision ◦ tqdm
Para inicio de sesión o Logueo	<ul style="list-style-type: none"> ◦ tensorboard
Para Gráficas	<ul style="list-style-type: none"> ◦ seaborn ◦ pandas
Para Backend	<ul style="list-style-type: none"> ◦ flask ◦ pyunpack ◦ flask_cors
Para Monitoreo de Recursos	<ul style="list-style-type: none"> ◦ htop

Cuadro 10: Librerías Backend

Además, para usar el mismo contenedor para ejecutar diferentes tareas con librerías incompatibles, se hizo uso de entornos virtuales usando *VirtualEnv* [51]. Se configuraron los siguientes entornos virtuales:

- *yolo-v5*: Para correr el modelo (entrenamiento y detección).
- *yolo-annotation-tool*: Para correr el script que levantaba la interfaz gráfica para etiquetar imágenes manualmente.

5.1.1.2 Frontend Docker

El contenedor en donde se despliega la interfaz gráfica posee la siguiente configuración:

- JavaScript
- CSS
- HTML
- Bootstrap

5.2. Dataset

En esta sección se detallan las tareas y mecanismos utilizados para generar el dataset con el que se entrenó el modelo.

5.2.1. Data Augmentation

Las redes neuronales convolucionales necesitan una enorme cantidad de imágenes para entrenar de forma efectiva el modelo. Esta técnica permite incrementar el rendimiento y robustez del modelo [14] y se realiza aplicando diferentes transformaciones a las imágenes como invertirlas, zoom, rotaciones entre otras dando una mayor generalización y una reducción del overfitting o sobreajuste.

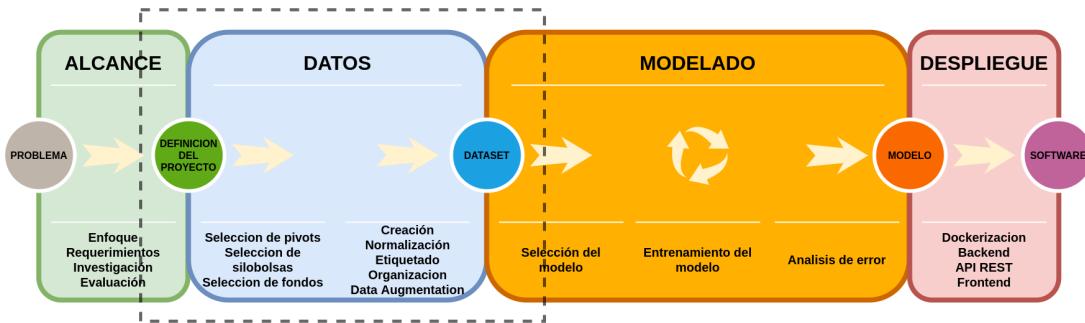


Figura 42: Workflow: Etapa de Datos

Para el dataset se tomaron imágenes de silobolsas y pivotes de riego, y a mano se recortaron solo los objetos de las mismas, para así crear plantillas con fondo transparente en formato PNG (RGBA, donde A viene de *Alpha* y se corresponde con el nivel de transparencia). Por otro lado se obtuvieron imágenes satelitales de diversos lugares del territorio argentino sin una determinada lógica, con el objetivo de conformar un conjunto de “fondos” sobre los cuales luego se combinarían con las plantillas de objetos. Para la generación de un dataset aumentado se programó un script en Python, el cual, toma todas las plantillas, las rota en un ángulo aleatorio y las posiciona en diversos lugares de un determinado fondo. De la combinación lineal de todos los fondos y todas las plantillas y sus alteraciones, surge un dataset lo suficientemente baste como para poder entrenar el modelo YOLO. Este script, además, se encarga de generar automáticamente los archivos de etiquetas para cada imagen nueva.

5.2.2. Formatos

Uno de los modelos con los que se trabajó en un principio, admitía el dataset de entrenamiento en formato *TFRecord*. El mismo se generaba a partir correr un script en Python: *YOLOToTFRecords* [52]. Este código fuente fue modificado para admitir archivos formato CSV y, a partir del mismo, generar el archivo TFRecord. Este es un formato especial creado por *TensorFlow* para almacenar el dataset en formato binario.

Finalmente en el modelo final YOLOv5, solo bastaba con configurar el directorio donde se ubicaban las imágenes y sus etiquetas para poder entrenarlo.

5.3. Entrenamiento

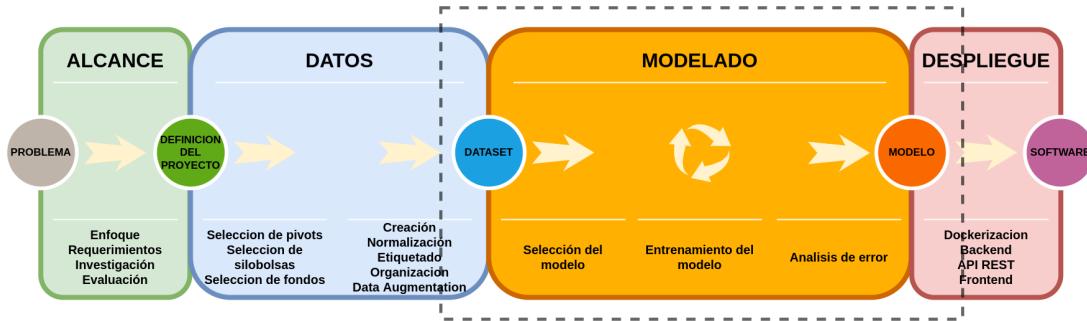


Figura 43: Workflow: Etapa de Modelado

En esta etapa se procedió a entrenar el modelo utilizando el dataset generado. El proceso de entrenamiento consta de iterar una inmensa cantidad de veces sobre el dataset de entrenamiento, para que así, la red neuronal por la que esta compuesta el modelo, “aprenda” a detectar correctamente. Es un proceso repetitivo, en el cual, poseer un gran poder de computo reduce los tiempos de entrenamiento hasta llegar a un resultado aceptable. Para determinar cuándo un resultado es lo suficientemente bueno, es en donde se utilizan diferentes métricas para realizar pruebas.

5.4. Pruebas

En esta sección se expondrán y explicarán diferentes diagramas y gráficos, en base a las pruebas de validación y testeo que se realizaron sobre el modelo de Machine Learning. Cabe destacar que esta etapa conforma, junto con el entrenamiento, un proceso cíclico, el cual se repite hasta lograr los resultados esperados. Se detalla a continuación algunos conceptos para comprender las métricas seleccionadas para valorar el funcionamiento del modelo, junto con sus correspondientes diagramas:

5.4.1. Precisión

Es la proporción de observaciones positivas predichas correctamente (verdadero positivo) sobre el total de predicciones realizadas. Su fórmula es:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

Donde:

TP: Truth Positive - Verdadero Positivo

FP: False Positive - Falso Positivo

Es decir, la precisión mide que tan preciso son las predicciones de nuestro modelo en base a un porcentaje de las predicciones correctas.

Ejemplo: La precisión representa el ratio entre verdaderos positivos y número total de positivos predichos. Por ejemplo, si el modelo detectó 5 pivotes de riego de los cuales 4 realmente lo eran, la precisión ha sido de $4 / 5 = 0.8$.

El clasificador ideal tendría una precisión de 1, ya que todas las predicciones positivas serían verdaderas. De forma equivalente, el peor clasificador posible, tendría una precisión de 0, ya que todas las predicciones positivas serían falsas.

Resultado obtenido

Para comprender la curva de la Figura 44, es necesario saber que el eje de ordenadas representa la Precisión de las inferencias realizadas por el modelo, expresada en un rango de 0 a 1; mientras que en el eje de abscisas, se tiene la Confianza, es decir, qué tan seguro está el modelo de la predicción realizada.

Se observa, entonces, en la Figura 44, la curva de precisión obtenida para la detección de Silobolsas y Riegos por Pivot desde la Epoch (ciclo de entrenamiento) 1 al 100 y de la Epoch 101 a la 700.

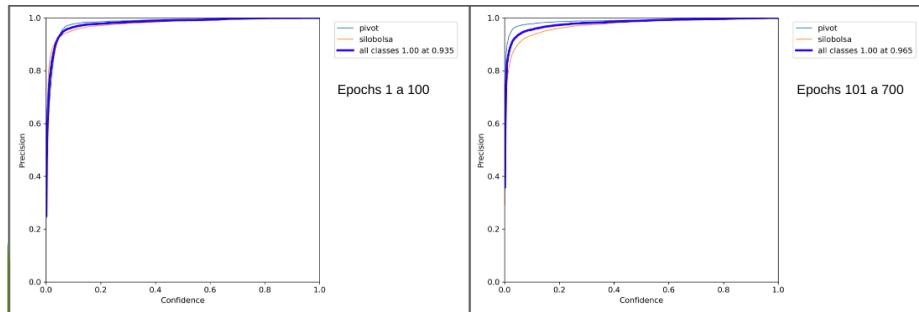


Figura 44: Precisión

5.4.2. Recall o Sensibilidad

Es la proporción de observaciones positivas predichas correctamente con respecto a todas las observaciones en la clase real, es decir, mide la capacidad del modelo de detectar casos positivos, en este caso, objetos que pertenezcan a alguna de las clases del dataset, ya sea Pivot o Silobolsa. Su fórmula es:

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

Por lo tanto **recall** es un porcentaje de los casos positivos correctamente acertados.

Ejemplo: La exhaustividad (recall en inglés, aunque también se conoce como sensibilidad o sensitivity, y TPR o True Positive Rate) de una clasificación es el ratio entre los verdaderos positivos (los que se han detectado) y los positivos reales (los hayamos detectado o no).

La exhaustividad sería el número de pivotes de riego detectados, dividido por el número de pivotes de riego reales totales: si hemos detectado 3 y había 5, la exhaustividad ha sido de $3 / 5 = 0.6$.

El clasificador ideal tendría una exhaustividad de 1, pues todos los positivos reales serían detectados como positivos (verdaderos positivos).

El peor clasificador posible tendría una exhaustividad de 0, pues ninguno de los positivos reales sería identificado como positivo.

Resultado obtenido

Similarmente a la curva de Precisión, en la Figura 45 se tiene a la Recall de las inferencias del modelo en el eje de ordenadas, expresada en un rango de 0 a 1; mientras que en el eje de abscisas, se tiene la Confianza.

Se observa, entonces, en la Figura 45, la curva de Recall obtenida para la detección de Silobolsas y Pivot desde la Epoch 1 al 100 y de la Epoch 101 a la 700.

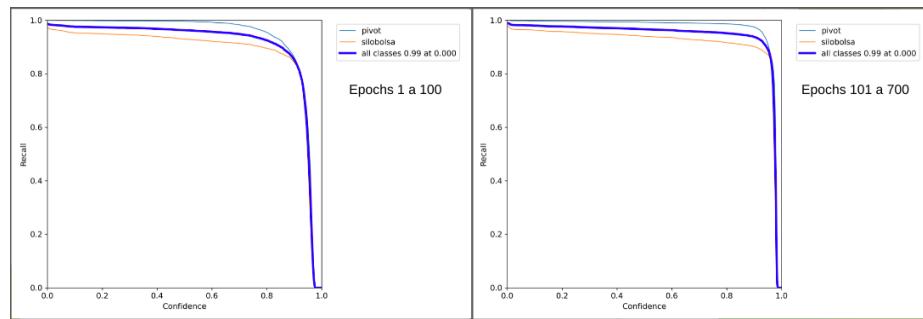


Figura 45: Recall

5.4.3. Precisión/Recall

De acuerdo a lo mencionado anteriormente, podemos resumir que:

Precisión: ¿Cuántas veces, lo que mi modelo dice, es realmente cierto?

Recall: ¿Cuántas veces mi modelo es capaz de identificar la verdad?

Al relacionar ambas métricas, podemos decir que, la Precisión se centra en lo que modelo dice y luego lo compara con la realidad. Por otro lado, Recall parte de la realidad, y después evalúa que tan bueno es el modelo para reconocerla.

Resultado obtenido

En la Figura 46 se observa la curva de Precisión/Recall obtenida para la detección de Silobolsas y Pivot desde la Epoch 1 al 100 y de la Epoch 101 a la 700. Precisión se ubica en el eje de ordenadas y Recall en el de abscisas.

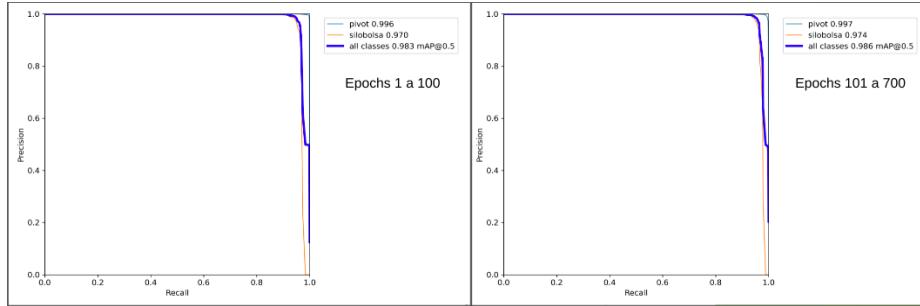


Figura 46: Precisión/Recall

5.4.4. F1 Score

Es el promedio ponderado de la Precisión y el Recall. Por lo tanto, esta métrica, tiene en cuenta tanto falsos positivos como falsos negativos. Su fórmula es:

$$F1Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

Resultado obtenido

Similarmente a la curva de Precisión o a la de Recall, en la Figura 47, se tiene a la métrica F1 Score de las inferencias del modelo en el eje de ordenadas, expresada en un rango de 0 a 1; mientras que en el eje de abscisas, se tiene la Confianza.

Se observa, entonces, en la Figura 47, la curva de F1 Score obtenida para la detección de Silobolsas y Pivot desde la Epoch 1 al 100 y de la Epoch 101 a la 700.

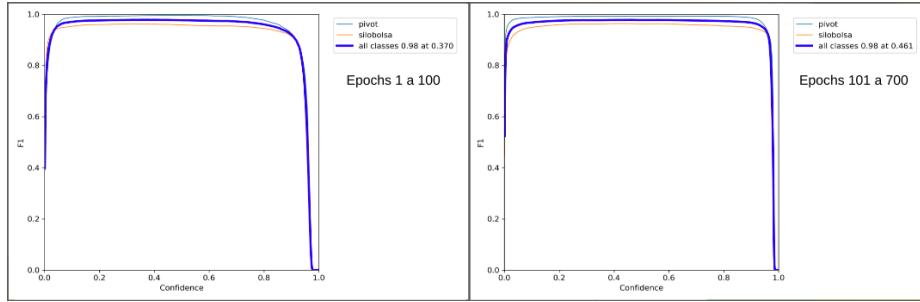


Figura 47: Curva F1 Score

5.4.5. Loss Function o Función de Pérdida y Precisión Media Promedio (mAP, Mean Average Precision)

En las siguientes imágenes se exponen los valores que toman las tres loss functions del modelo (bounding box, detección de objeto, clasificación) para los conjunto de validación y testeo; así como también las curvas de precisión, recall y mAP generales (ver figuras 48 y 49).

Antes de comenzar a con los resultados, vamos a definir lo que es un **epoch**, término que vamos a utilizarlo para explicar los resultados obtenidos.

Epoch: Dentro del contexto del aprendizaje automático, epoch se puede describir como un ciclo completo a través de todo el conjunto de datos de entrenamiento e indica la cantidad de pasadas que el algoritmo ha completado durante ese entrenamiento. Es decir, un epoch se compone en última instancia de lotes de datos e iteraciones, cuya suma en última instancia equivaldrá a una época.

Luego de las primeras 100 epochs (o épocas) de entrenamiento, se observan los siguientes resultados:

- En cuanto a métricas, los resultados ya eran muy buenos al finalizar la epoch 100.
- Las tres loss functions de validación continuaban bajando al finalizar el entrenamiento, lo cual indicaba que aún había lugar para continuar mejorando los resultados, sin overfitting o sobreajuste.
- La loss de validación de clasificación, sin embargo, parecía haber llegado a un mínimo.

Al finalizar las siguientes 600 epochs, se observan los siguientes resultados:

- La precisión y recall oscilaron entre valores altos, superiores al 96 %.
- El mAP continuó aumentando. En particular fue muy positiva la mejora en el mAP con barrido de IoU (0.5 a 0.95).
- Las loss functions de clasificación y validación continuaron bajando, pero sus valores ya eran extremadamente bajos. Quizás quedó algo de lugar para continuar con el entrenamiento, pero se optó por continuar con los siguientes objetivos del proyecto, ya que los resultados cumplían con las expectativas.

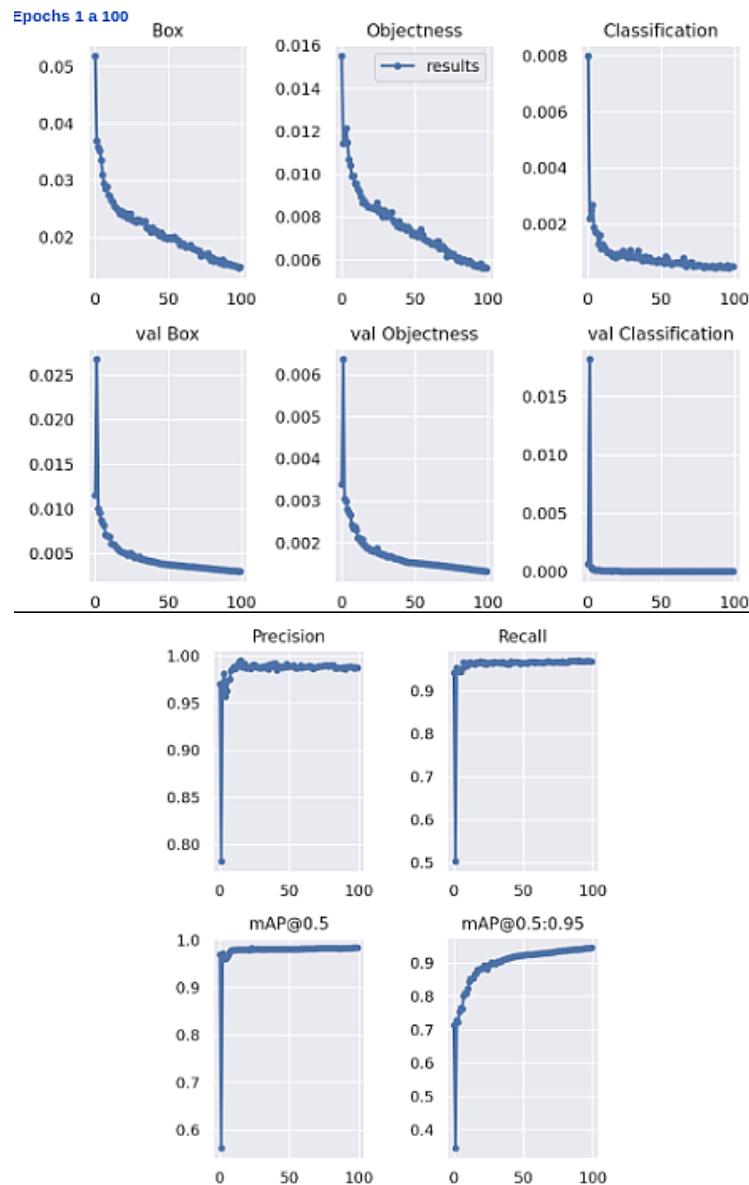


Figura 48: Loss Function y mAP. Primeras 100 epochs.

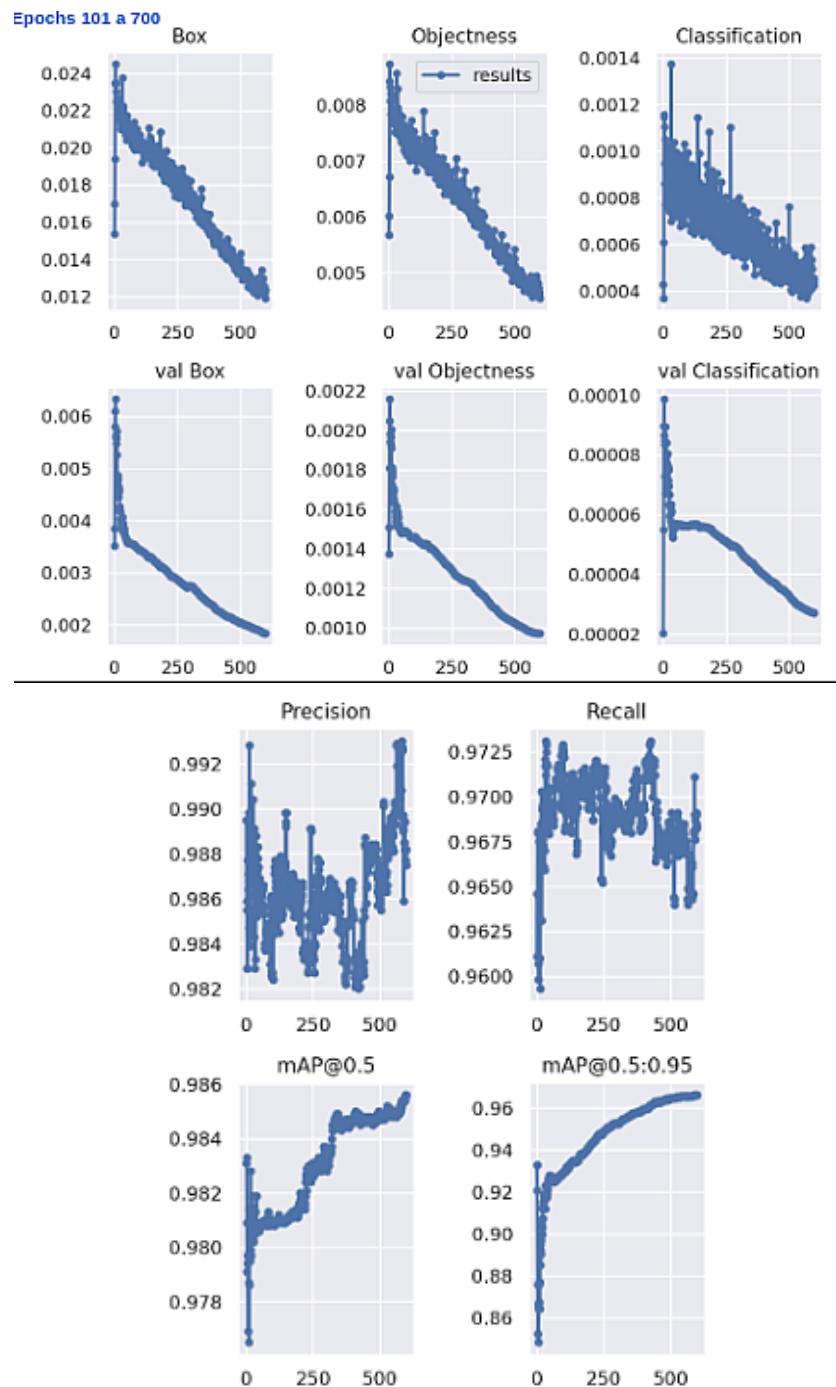


Figura 49: Loss Function y mAP. Epochs 101 a 700.

5.4.6. Matriz de Confusión

La siguiente matriz de confusión muestra el porcentaje de objetos que el modelo predijo correctamente contra los incorrectos y contra las omisiones.

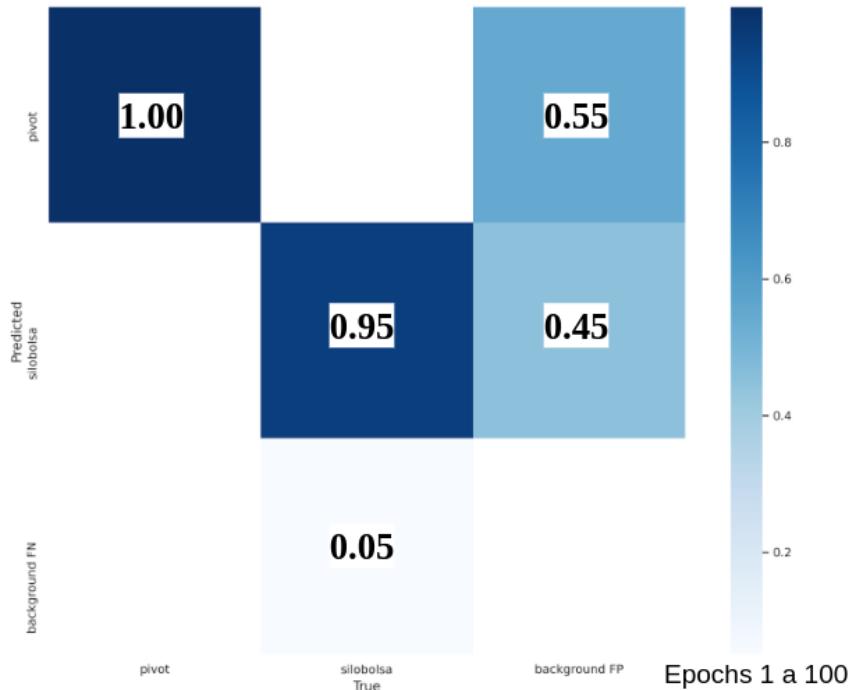


Figura 50: Matriz de Confusión. Primeras 100 epochs.

Al finalizar las primeras 100 epochs, se ve en el gráfico de la izquierda que las detecciones de pivotes son excelentes, es decir, cada pívote fue reconocido, por eso el resultado 1.00. Sin embargo, de las falsas predicciones (FP), más de la mitad de las mismas eran de pívote (55 %). Por otro lado, en el caso de las silobolsas, se detectaron correctamente el 95 % de las mismas, con una proporción de FP de 45 %.

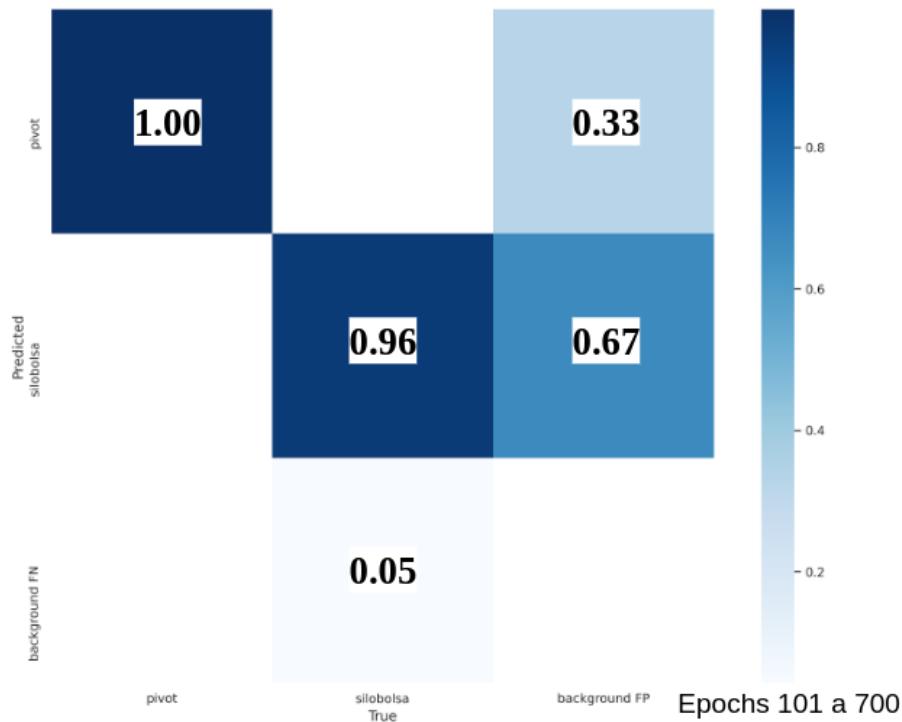


Figura 51: Matriz de Confusión. Epochs 101 a 700.

Al finalizar la segunda etapa de entrenamiento de 600 epochs, vemos que el porcentaje de silobolsas detectado subió un 1%, mientras que la mayor diferencia se ve en la distribución de FP, con solo 33 % de pivotes mal detectados y un 67 % de silobolsas. En este caso se aprecia claramente un aumento de la precisión a costa de una disminución de la recall.

5.5. Despliegue

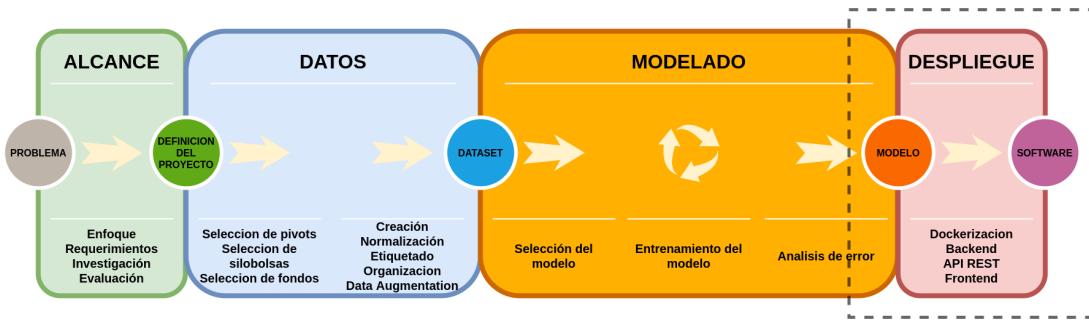


Figura 52: Workflow: Etapa de Despliegue

Para la etapa de despliegue se optó por hacer uso de los dos contenedores Docker descriptos líneas arriba: uno corriendo el Backend y otro el Frontend. Este enfoque permite que el software en su totalidad sea fácilmente portado a diferentes sistemas.

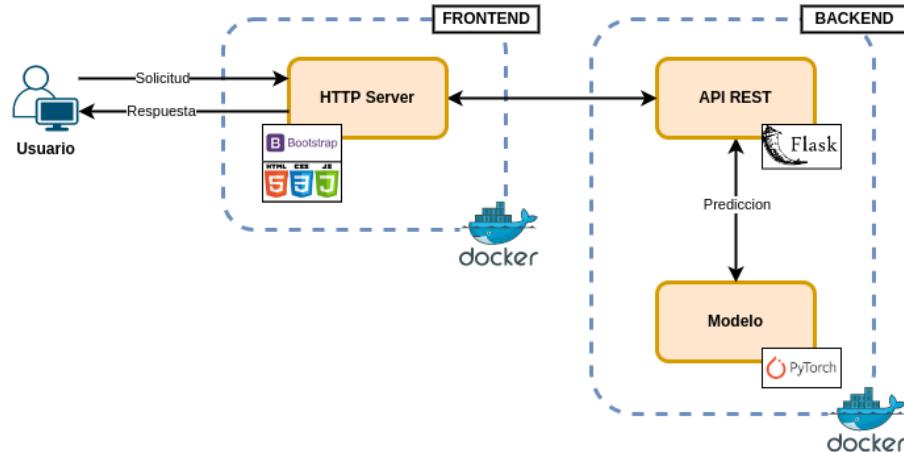


Figura 53: Arquitectura del Software Implementado

Chapter 6

Validación

6. Introducción

En el presente capítulo se describen los procedimientos de prueba para los requerimientos de usuario mencionados en el Cuadro 1. Cada cuadro describe el objetivo de la prueba, el procedimiento, el resultado esperado y el estado final indicando si la prueba se realizó de forma exitosa o no. Por último, se describe una matriz de trazabilidad que permite relacionar los requerimientos de usuario con los requerimientos funcionales y asociar cada uno con su caso de prueba correspondiente.

6.1. Validación de los requerimientos de usuario

A continuación en esta sección se detallarán las validaciones realizadas para mostrar el cumplimiento de los Requerimientos de Usuario en los siguientes Cuadros 11, 12, 13.

TC-01	Validar que el usuario pueda cargar una nueva imagen al sistema, desde la página principal.
Objetivo	Lograr que el usuario pueda cargar una imagen de diferentes formatos de forma exitosa.
Procedimiento	<ol style="list-style-type: none">El usuario debe dirigirse hacia la página principal y hacer clic en el cuadro con el título “Seleccionar archivo”.Luego debe seleccionar la imagen o un archivo de imágenes a ser analizadas desde su computadora.Después se indicará si la imagen fue subida exitosamente o no.
Resultados esperados	Éxito de la operación al subir una imagen.- Figuras 54 55 56
Estado	APROBADO

Cuadro 11: Caso de Prueba del Requerimiento de Usuario R-01

Proyecto Integrador Ingeniería en Computación - UNC

[IA Imágenes Satelitales](#) Home Quienes somos Subir imágenes Resultados



IA PARA DETECTAR PIVOTES DE
RIEGO Y SILOBOLSAS EN IMÁGENES
SATELITALES

Computer Vision - Usando Yolo v5

Figura 54: Página principal

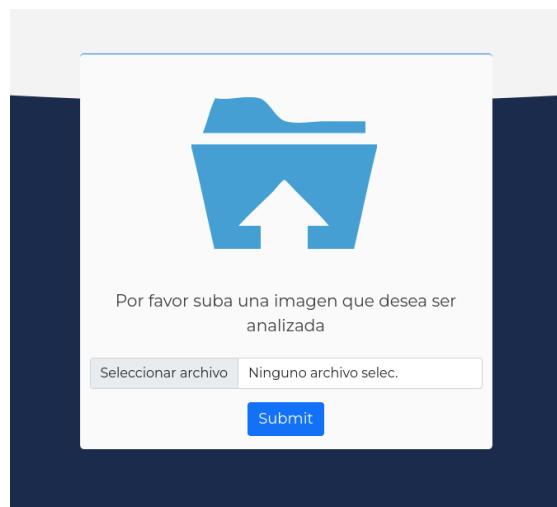


Figura 55: Subir una imagen para analizar

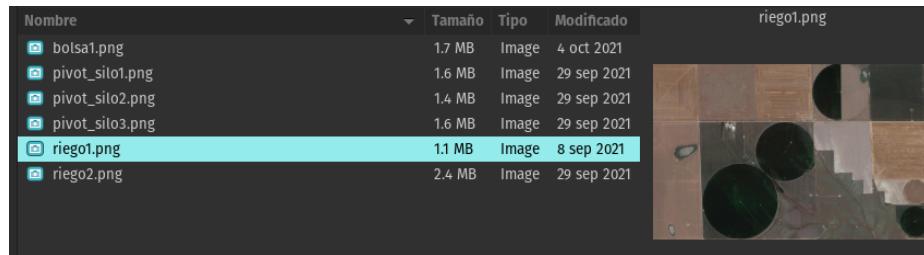


Figura 56: Seleccionar una imagen desde la PC-Usuario

TC-02	Validar la presencia de una imagen cargada correctamente.
Objetivo	Obtener información sobre la validez o invalidez de la imagen subida por el usuario.
Procedimiento	<ol style="list-style-type: none"> El usuario debe dirigirse a la página y luego de subir una imagen debe observar si el formato es aceptado. Luego debe ver habilitado el botón “Submit” para dar inicio a la Detección.
Resultados esperados	Éxito en la presencia de la imagen subida.- Figura 55, 56 y 57
Estado	APROBADO

Cuadro 12: Caso de Prueba del Requerimiento de Usuario R-02

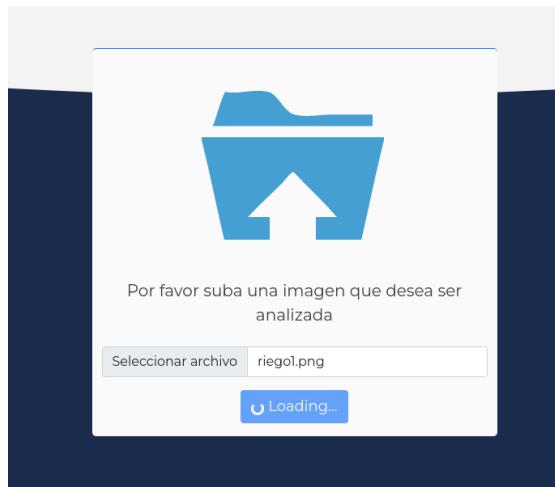


Figura 57: Inicio de la detección - Loading

TC-03	Validar cuando la imagen es de un formato erróneo, mostrar un mensaje del error y debe ser posible volver a cargar una nueva imagen.
Objetivo	Al obtener un mensaje sobre la invalidez de la imagen subida, volver a habilitar el botón para subir una nueva imagen.
Procedimiento	<ol style="list-style-type: none"> El usuario debe dirigirse a la página y luego de subir una imagen errónea debe observar un mensaje de imagen inválida. Luego debe ver habilitado el botón para volver a cargar una imagen nueva.
Resultados esperados	Éxito en la presencia de un mensaje de formato inválido.- Figura 58 muestra el mensaje de error.
Estado	APROBADO

Cuadro 13: Caso de Prueba del Requerimiento de Usuario R-03

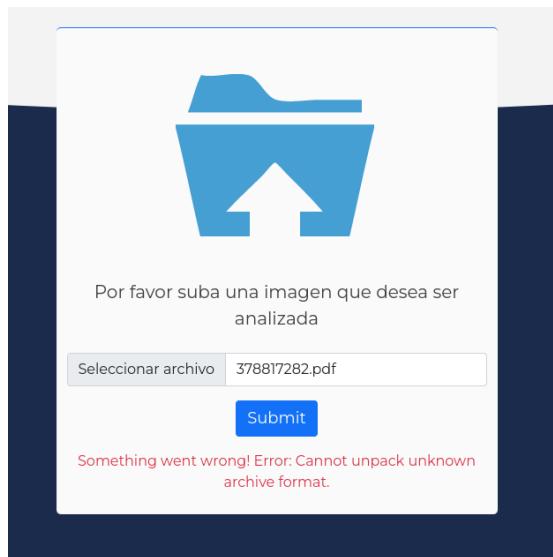


Figura 58: Error al subir una archivo de formato invalido

TC-04	Validar que debe ser posible visualizar en la página la/s imágenes cargadas y un historial con información de las analizadas.
Objetivo	El usuario debe visualizar la información sobre todas las imágenes analizadas anteriormente y la actual.
Procedimiento	<ol style="list-style-type: none"> 1. El usuario debe dirigirse a la página, luego de subir/seleccionar una imagen y hacer clic en el botón de 'Submit', iniciará la detección automáticamente, 2. Luego, como resultado debe ver la imagen resultante con las detecciones encontradas. 3. Además debe observar una tabla con el historial de todas las detecciones y los detalles de las mismas.
Resultados esperados	Éxito en la presencia de la imagen analizada y de una tabla con el historial de todas las imágenes analizadas hasta el momento.- Figuras 59 y 60.
Estado	APROBADO

Cuadro 14: Caso de Prueba del Requerimiento de Usuario R-04

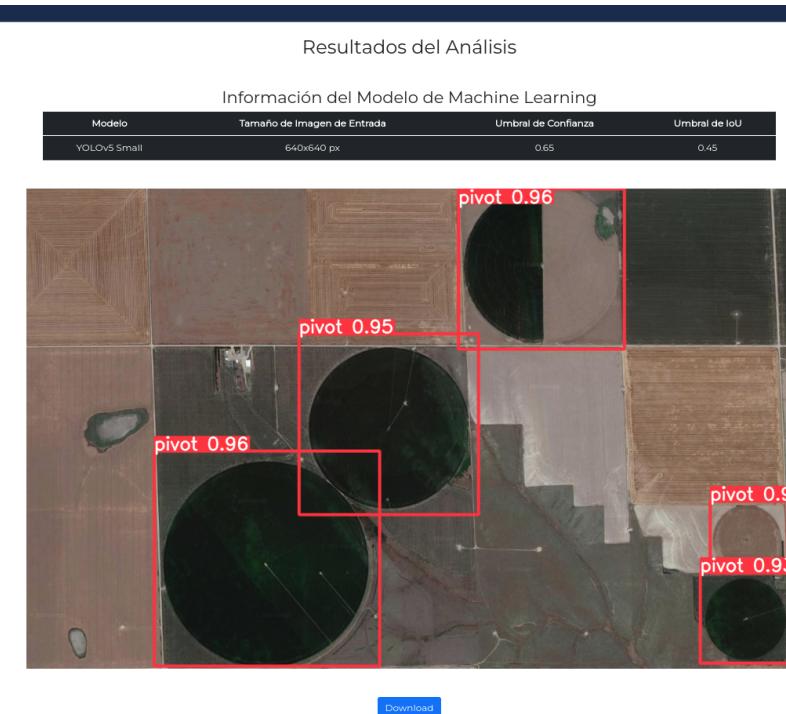


Figura 59: Fin de la detección - Resultado



Figura 60: Tabla de Información de detecciones - Resultado

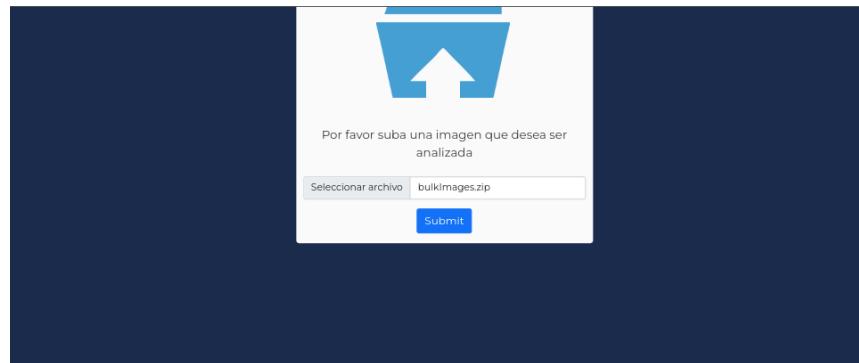
TC-05	Validar que es posible analizar la imagen cargada, obtener información de la misma y que el sistema debe informar si se encontró algún objeto o no.
Objetivo	El usuario debe visualizar la imagen ya analizada y el resultado de la detección, remarcando donde se encuentran los objetos detectados y a qué tipo corresponde: silobolsa o riego por pivot.
Procedimiento	<ol style="list-style-type: none"> 1. El usuario luego de solicitar que se analice una imagen, debe visualizar el resultado. 2. La imagen resultante debe contener cuadros delimitadores encerrando a los objetos detectados e indicando su tipo.
Resultados esperados	Éxito en la presencia del resultado de la detección con información adicional .- Figura 61 detalla el historial y los resultados de la detección. Figura 60 muestra el resultado en caso de haber subido una sola imagen.
Estado	APROBADO

Cuadro 15: Caso de Prueba del Requerimiento de Usuario R-05

6.1.1. Matriz de trazabilidad

En la matriz de trazabilidad presentada a continuación se relacionan todos los requerimientos planteados con los casos de test descritos en la sección anterior, con la finalidad de obtener una mejor visión general que todos los requerimientos fueron testeados exitosamente.

Los requerimientos no funcionales RNF-04 (Capacidad de ser desplegado fácilmente en diferentes sistemas) y RNF-05 (La API debe desarrollarse en Node.js y para la



Resultados del Análisis

Información del Modelo de Machine Learning

Modelo	Tamaño de Imagen de Entrada	Umbral de Confianza	Umbral de IoU
YOLOv5 Small	640x640 px	0.65	0.45

[Download](#)

Información del Análisis

Nombre	Tiempo Total	Tiempo de Inferencia	Tamaño de la Imagen	Pivotes Detectados	Silobolas Detectados
bolsa1.png	54.2ms	0.053s	320x640 px	0	0
pivot_silo1.png	54.2ms	0.053s	352x640 px	1	0
pivot_silo2.png	54.2ms	0.054s	384x640 px	0	0
pivot_silo3.png	54.2ms	0.052s	352x640 px	0	0
regol.png	54.2ms	0.054s	416x640 px	5	0
regol2.png	54.2ms	0.060s	512x640 px	3	0

Figura 61: Detección de un archivo con imágenes comprimidas

TC-06	Validar que es posible descargar la imagen ya analizada.
Objetivo	El usuario debe poder visualizar y descargar la/s imagen/es analizada/s.
Procedimiento	<ol style="list-style-type: none"> El usuario una vez que pueda visualizar la imagen analizada, debe tener la opción de descargarla a su máquina local. La imagen analizada y descargada debe encontrarse en la máquina local del usuario.
Resultados esperados	Éxito en la presencia de la imagen o un archivo con varias imágenes analizadas en la máquina local del usuario.- Figura 62.
Estado	APROBADO

Cuadro 16: Caso de Prueba del Requerimiento de Usuario R-06

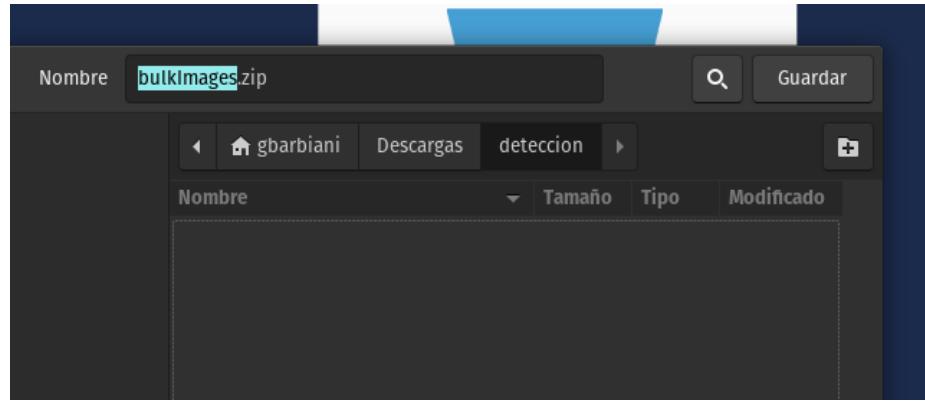


Figura 62: Descarga de archivo con imágenes detectadas - Resultados

interfaz gráfica se debe usar CSS, HTML y JavaScript) no figuran en la matriz de trazabilidad, debido a que establecen metodologías y herramientas de trabajo.

	RF-API1-01	RF-API1-02	RF-API1-03	RF-API1-04	RF-API1-05	RF-FE-01	RF-FE-02	RF-FE-03	RF-FE-04	RF-FE-05	RF-FE-06	RNF-01	RNF-02	RNF-03
TC-01	✓					✓	✓					✓	✓	✓
TC-02	✓						✓					✓	✓	✓
TC-03	✓						✓					✓	✓	✓
TC-04	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓
TC-05	✓		✓	✓			✓	✓	✓	✓		✓	✓	
TC-06	✓			✓		✓	✓			✓		✓		

Cuadro 17: Matriz de Trazabilidad

Chapter 7

Conclusión

7. Introducción

En el presente proyecto, se realizó un estudio extenso sobre Machine Learning, sus conceptos más importantes como también sus diferentes aplicaciones y casos de uso. Se comparó y probó diferentes algoritmos de Machine Learning. Luego, se eligió el modelo YOLOv5 de los analizados para probar su funcionamiento en un caso práctico de Computer Vision. Se investigó la utilidad de este modelo para detección de imágenes en tiempo real y se llegó a la conclusión de que es muy útil y puede ser aplicado a diferentes problemas. Como lo es en nuestro caso el análisis de imágenes satelitales y así detectar el abuso de los recursos hídricos, evasión impositivas en el rubro agropecuario, deforestaciones, asentamientos y expropiaciones ilegales, entre otros, mediante la detección de pivotes de riego y silobolsas en campos.

El conocimiento adquirido a lo largo de la carrera, en las materias tales como Arquitectura de Computadoras y Sistemas Operativos, formó una base de conocimiento imprescindible para el desarrollo del proyecto. Los problemas que se plantearon pudieron ser abarcados gracias a la preparación recibida en la facultad, a la formación e investigación realizada sobre Deep Learning, Computer Vision, Redes Neuronales, entre otros, junto con paciencia, perseverancia y curiosidad.

El objetivo era desarrollar y aplicar estos conocimientos para ayudar al medio ambiente o proveer alguna herramienta que sea útil para el mismo, y que pueda ser usado por cualquier persona interesada sin costo alguno y hacer un aporte de Machine Learning al ámbito público.

7.1. Problemas y Limitaciones

A continuación, se elaboran algunos problemas y limitaciones que se presentaron durante el desarrollo del proyecto.

■ *Investigación:*

Antes de elegir el modelo adecuado en base a una opinión apta, se debió contar con suficiente información y conocimiento en materia de Machine Learning, para comprender como funcionan las Redes Neuronales y todo lo que hay detrás de ellas para su funcionamiento. Conocimiento que al momento de empezar este proyecto no se contaba a fondo. El camino comenzó con un curso de Coursera [53], dictado por el Dr. Andrew Ng, el cual allanó el camino para conocer en detalle qué es y cómo funciona Machine Learning. También introdujo los primeros conceptos a cerca de los cimientos de YOLO. Se profundizó el estudio con la lectura de los papers oficiales de YOLOv1, YOLOv2 y YOLOv3, (y más adelante, YOLOv4 y YOLOv5). Se complementó la investigación con el estudio de técnicas para mejorar los resultados de inferencia: purga de dataset, data augmentation, transfer learning, seteo de hiper-parámetros, recomendaciones en cuanto a la duración de las epochs, entre otros. Por otro lado, se investigó en materia de métricas y creación y análisis

de gráficas para detectar tempranamente posibles puntos de overfit o bien detectar la posibilidad de continuar entrenando. El estudio de métricas tales como precisión, recall, F1 Score, Mean Average Precision (mAP), mAP con barrido de IoU, matriz de confusión y correlograma de etiquetas, por nombrar algunos. La selección de las métricas a utilizar no fue trivial, se compararon varias métricas para medir la performance del modelo y dada la naturaleza del dataset se hizo un descarte de las métricas que no aportaban mucha información.

A continuación se realizó una lectura y comparación de los distintos frameworks que se disponían en Python para basar el modelo de Redes Neuronales y demás algoritmos y optimizadores que componen un modelo de Machine Learning completo, tales como: PyTorch, TensorFlow y TensorFlow 2.0; optando por Tensorflow en su versión vainilla dada su mayor trayectoria en la industria, educación e investigación y la abundante documentación disponible en la web. Sin embargo, más avanzado en el proyecto y obligados a cambiar de modelo, al migrar a la versión más reciente de YOLOv5, gran parte de su mejora de performance se debía a su implementación en PyTorch, framework el cual, en el transcurso de los meses que pasaron desde el comienzo de este proyecto y el momento de su selección, había madurado considerablemente y resultaba una opción más que válida.

- *Preparación del entorno de trabajo:*

El proyecto fue desarrollado principalmente en la súper computadora (cluster) de la Facultad de Ciencias Exactas, Físicas y Naturales denominada: Nabucodonosor, ya que ofrecía suficiente poder de cómputo y almacenamiento para soportar los entrenamientos requeridos para obtener los resultados deseados, así como también la disponibilidad 24/7 para dejar corriendo scripts de diferente índole (ej.: data augmentation). No obstante, la computadora ofreció tempranamente algunos problemas al momento de setear el entorno de trabajo. Se tuvieron limitaciones para la conexión vía túnel SSH para hacer uso de la consola, conexión la cual una vez lograda, tenía la limitación de no ofrecer interfaz gráfica para algunas features que ofrecían los modelos descargados. Por otro lado, también hubo dificultades para lograr configurar un contenedor Docker el cual permitiera el uso completo del hardware que el clúster tenía disponible. Finalmente, por cuestiones de seguridad, el acceso a la página web con la interfaz visual para este proyecto se vio limitado a ser vía túnel SSH, esfumando la posibilidad de ofrecer un endpoint de público acceso. Otro tema que causó algunas limitaciones fueron los inesperados cortes de luz que sufría la facultad, lo cual dejaba inaccesible el clúster hasta su reinicio de manera manual. Impactando la disponibilidad, también se vio agregado la cola de espera para hacer uso de la maquina, dado que es un recurso compartido por todo el ámbito de la educación e investigación del país.

- *Elaboración del dataset:*

Este paso fue uno de los más importantes, por que influye significativamente en el entrenamiento del algoritmo, por que se alimenta y aprende a través de las imágenes que se le entregan. Es por eso que las mismas deben ser cuidadosamente elegidas, haciendo foco en el objeto que se quiere detectar y con una gran variedad de los mismos. En este paso, se cometieron algunos errores al momento de elegirlos debido a la diferencia de tamaño que hay entre los objetos en cuestión, al principio no fueron lo suficientemente grandes o centrados, y dicha mala elección se vio reflejada en los resultados de detección que se obtenían. Es por eso que se invirtió suficiente tiempo en preparar el conjunto de datos, con el tamaño correcto

de la imagen y el objeto en la misma, diferencias de colores, fondos y texturas para hacer mas variado el conjunto.

- *Análisis y elección del algoritmo:*

Al momento de la realización de este trabajo se investigó sobre los algoritmos de Computer Vision en tiempo real más populares y que sea aplicable a éste proyecto. Al principio se optó por el algoritmo YOLOv3, el cual era el estado del arte al comenzar el proyecto. El modelo prometía tiempos de inferencia muy rápidos, categorizándolo como un algoritmo de detección en tiempo real, la posibilidad de hacer transfer learning, y ajustar el modelo a un dataset personalizado. Tras familiarizarse con el modelo y la especificación para el etiquetado, se realizaron las primeras pruebas preliminares con un dataset personalizado que incluyera dos clases. Esto arrojó en principio resultados prometedores, sin embargo, aun lejos de lograr la precisión deseada. Se procedió, entonces, a incrementar el tamaño del dataset, purgarlo y diversificarlo, pero sin lograr los resultados esperados. Se realizaron barridos en los hiper-parámetros que el modelo hacia disponible, pero sin grandes mejoras. Se prosiguió con una alteración directa al código fuente del modelo, agregando nuevos y variados optimizadores para mejorar la performance de la función de costo. Se calcularon nuevos anchor boxes, tanto elegidos a mano como mediante el uso de el algoritmo K-Means. Ambos ensayos tuvieron un impacto casi nulo en la performance del modelo. Tras meses de ensayos, se tomó la decisión de buscar otro código fuente o bien otro modelo, lo que dio como resultado el descubrimiento de una nueva y mejorada versión de YOLOv3: YOLOv5. El mismo otorgó, desde las primeras pruebas, mayor facilidad para el seteo de su entorno, soportaba el mismo dataset que se venia utilizando en YOLOv3 y brindó resultados excelentes, llegando incluso a superar las expectativas.

7.2. Trabajos futuros

La aplicación final presentada como una página web, permite al usuario final hacer uso de un modelo funcional mediante una interfaz gráfica familiar. Los resultados finales son aceptables pero distan de la excelencia, aun así, conforman una base sólida para ser mejorados en proyectos futuros, por ejemplo, mejorando el dataset y re-entrenando por más tiempo, realizando un cambio de modelo base, por nombrar algunas ideas. Por otro lado, el software presentado como un backend y frontend permiten ser modificados con facilidad por los futuros desarrolladores que continúen con este trabajo.

7.3. Aporte personal

El conocimiento adquirido a lo largo de la carrera Ingeniería en Computación, en especial en materias tales como Redes de Computadoras, Sistemas Operativos y Sistemas de Computación, fueron fundamentales para formar una base sólida para desarrollar este proyecto. Si bien el tema principal no es abarcado como una materia tal en la carrera, la problemática que se planteó pudo ser resuelta gracias a la preparación recibida por la facultad y nuestros profesores, junto con la curiosidad, paciencia y perseverancia, se logró alcanzar los objetivos propuestos.

7.4. Agradecimientos

En primer lugar queremos expresar nuestro agradecimiento al Director de éste Proyecto Integrador: Mg. Ing. Miguel Angel Solinas, Codirector Dr. Ing. Miguel Solinas Jr. y a su esposa la Dra. Ing. Margot Selosse por su incondicional apoyo durante todo el desarrollo del mismo; transmíténdonos sus conocimientos y experiencias, así como también supervisando nuestros avances hasta el final. En segundo lugar agradecer a la Facultad y todos sus docentes que nos brindaron las herramientas, conocimientos y recursos para llegar a esta instancia, en particular al Ing. Aldo Algorry y al Dr. Ing. Gustavo Wolffmann por su disposición y soporte brindado para el uso del clúster Nabucodonosor, clave en el desarrollo del proyecto. Finalmente queremos agradecer a nuestras familias y amigos, que con su apoyo y ánimos brindados logramos cumplir nuestra meta de ser Ingenieros. No quisieramos finalizar esta sección de agradecimientos, sin antes mencionar que el siguiente proyecto no hubiera sido posible sin el gran trabajo realizado por Ultralytics que desarrolló el modelo de YOLOv5 en Pytorch: <https://github.com/ultralytics/yolov5>.

Chapter 8

Apéndice

8. Imágenes Dataset

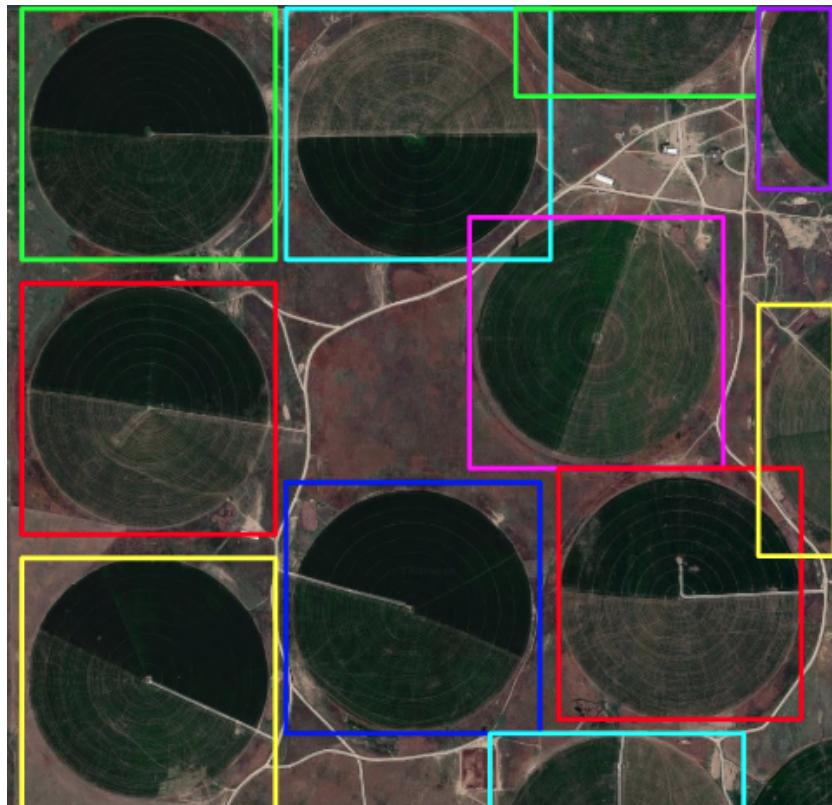


Figura 63: Pívot etiquetado manualmente

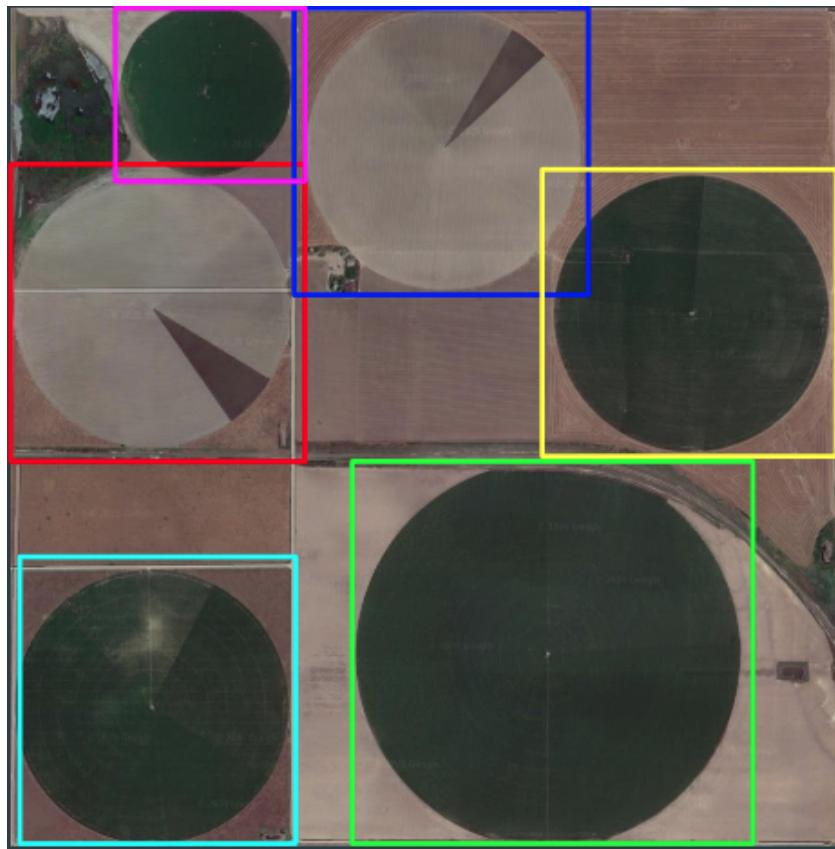


Figura 64: Pívor etiquetado manualmente



Figura 65: Silobolsa etiquetado manualmente



Figura 66: Silobolsa etiquetado manualmente



Figura 67: Silobolsa etiquetado manualmente



Figura 68: Silobolsa etiquetado manualmente



Figura 69: Pívot generado con data augmentation



Figura 70: Pívot generado con data augmentation

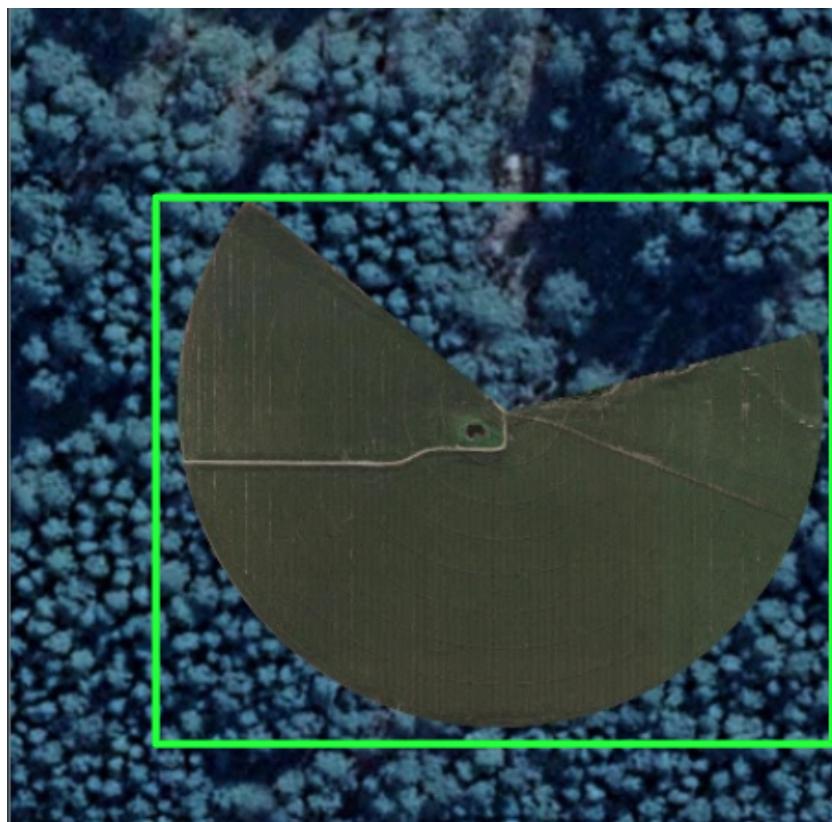


Figura 71: Pívot generado con data augmentation



Figura 72: Silobolsa generado con data augmentation



Figura 73: Silobolsa generado con data augmentation

9. Manual de Usuario

9.1. Introducción

En esta sección se detallan los pasos a seguir para el uso adecuado del software Detector de Pivotes de Riego y Silo bolsas en Imágenes Satelitales para aplicaciones agrícolas.

9.2. Pre-requisitos

Contar con Docker instalado en la computadora.

En el caso de no optar por el uso de Docker, se puede correr localmente de contar con los siguientes requerimientos:

- Distribución Linux: Debian o Ubuntu preferentemente (o sus derivados).
- Python 3.7 o superior.
- Virtualenv.

9.3. Instalación

1. Descargar o clonar el repositorio desde Github: https://github.com/gianfrancob/detector_pivotes_silobolsas
2. Inicializar el contenedor Docker corriendo:
 - `docker build -t detector_pivotes_silobolsas`
 - `docker run -p 5000:5000 -name test -it detector_pivotes_silobolsas`
3. En el caso de no optar por el uso de Docker, se puede correr localmente de la siguiente manera:
 - Crear entorno virtual usando virtualenv: `virtualenv venv`
 - Activar el entorno: `source ./venv/bin/activate`
 - Instalar las dependencias en el entorno: `pip install -r ./requirements.txt`
4. Iniciar el servicio backend: `python ./utils/flask_rest_api/restapi_pivot_silobolsa.py -port 5000`
5. Ingresar a la página abriendo en el navegador el siguiente fichero HTML: `./utils/bootstrap_frontend/index.html`

xt

9.4. Ingresar a la página

Para ejecución local, la dirección URL donde se encuentra la aplicación es: *DIRECTORIO RAIZ/utils/bootstrap_frontend/index.html*

Para ejecución web, dependerá particularmente del tipo de despliegue implementado.

9.5. Cargar una imagen

Una vez en la página principal, dirigirse al botón "Seleccionar archivo" que se encuentra en la mitad de la misma y hacer clic en él para seleccionar una imagen o desde la computadora del usuario, los formatos soportados son .jpg, .png .

9.5.1. Error en el formato de la imagen

Si el formato de la imagen cargada esta dañado o es invalido, se mostrara el siguiente mensaje, como se muestra en la figura 75:

9.6. Cargar varias imágenes

Para la opción del análisis de un conjunto de imágenes, previamente se debe crear un archivo de compresión con todas las imágenes en alguno de los siguientes formatos: RPM (.rpm), RAR (.rar), RZIP (.rz), TAR (.tar), XZ (.xz), ZIP (.zip, .jar), GZIP (.gz), 7z (.7z).

Luego, hacer clic en el botón "Seleccionar archivo" para subir el archivo con todas las imágenes a ser analizadas.

Nota: No debe salir ningún mensaje de error para continuar con el análisis, ver imagen 75

9.7. Inicio de la detección

Una vez cargada la imagen o las imágenes en formatos validos, hacer clic en el botón **"Submit"**, para dar inicio a la detección.

El botón cambiara de nombre y aparece como "Loading", indicando que la detección esta en proceso. Como se puede apreciar en la Figura 76.

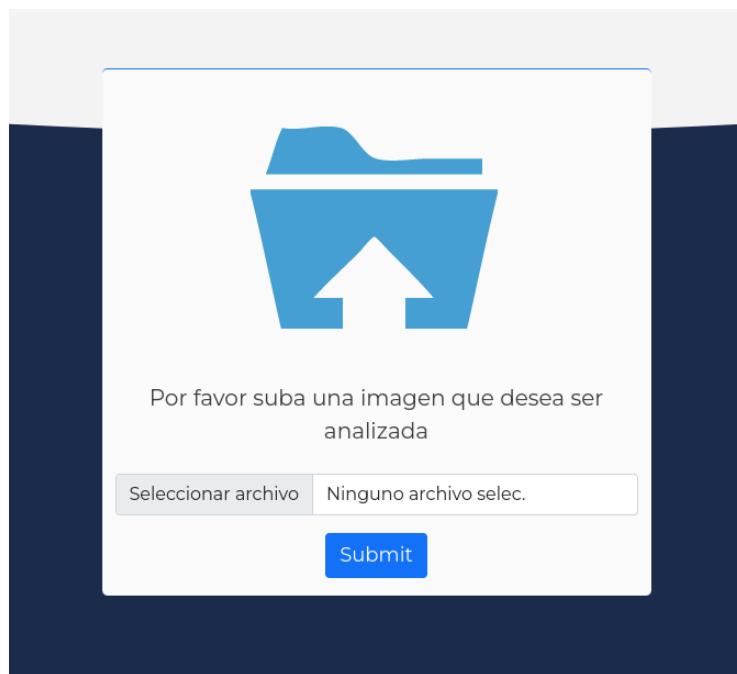


Figura 74: Subir una imagen para analizar

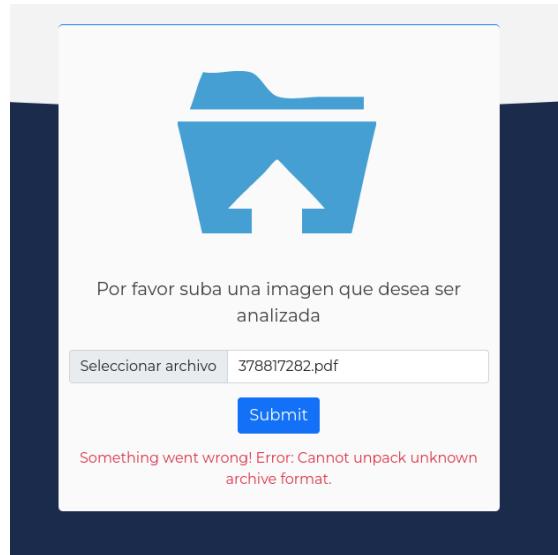


Figura 75: Error al subir una archivo de formato invalido

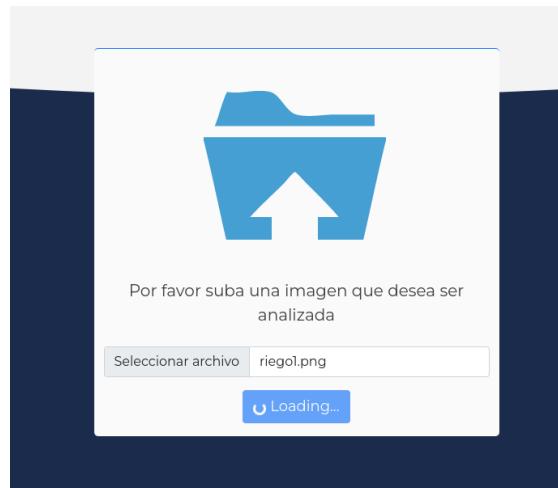
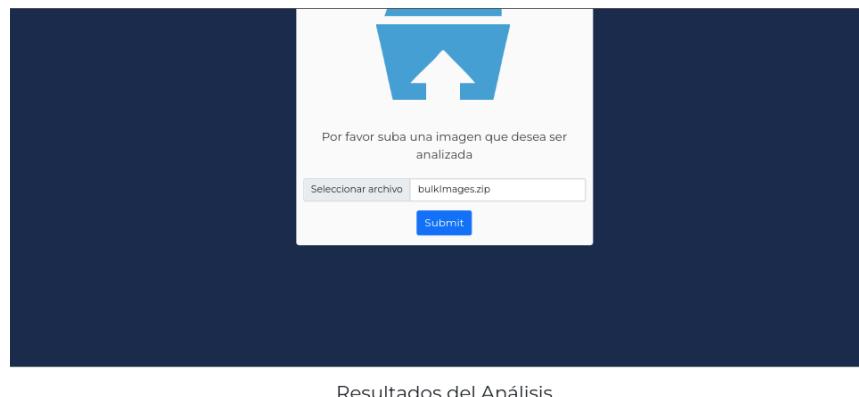


Figura 76: Inicio de la detección - Loading

9.8. Descarga de imágenes

Una vez realizada la detección de los objetos, se va a mostrar como resultado las imágenes analizadas indicando la presencia de los objetos Silo bolsas o Riegos por Pívot junto con una tabla donde se detalla la información de la detección.

Luego, se habilitara un botón ”**Download**” que al hacer clic en él, se procede a la descarga de la/s imágenes, como se ve en la Figura 77 y se debe seleccionar el archivo del destino como se muestra en la Figura 78



Resultados del Análisis

Información del Modelo de Machine Learning

Modelo	Tamaño de Imagen de Entrada	Umbral de Confianza	Umbral de IoU
YOLOv5 Small	640x640 px	0.65	0.45

Download

Información del Análisis

Nombre	Tiempo Total	Tiempo de Inferencia	Tamaño de la Imagen	Pivotes Detectados	Silobolas Detectados
bolsa1.png	54.2ms	0.053s	320x640 px	0	0
pivot_silo1.png	54.2ms	0.053s	352x640 px	1	0
pivot_silo2.png	54.2ms	0.054s	384x640 px	0	0
pivot_silo3.png	54.2ms	0.052s	352x640 px	0	0
riego1.png	54.2ms	0.054s	416x640 px	5	0
riego2.png	54.2ms	0.060s	512x640 px	3	0

Figura 77: Detección de un archivo con imágenes comprimidas

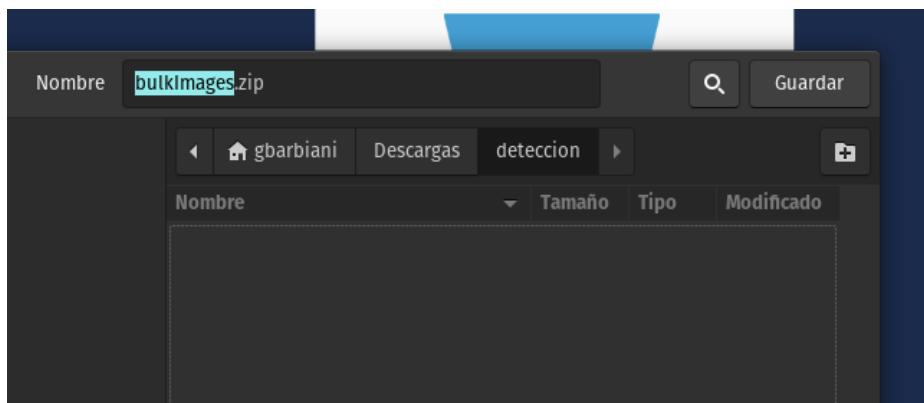


Figura 78: Descarga de archivo con imágenes detectadas - Resultados

Bibliografía

- [1] Ejemplo de foto satelital de pivotes de riego. <https://agrotendencia.tv/agricultura-en-el-desierto-como-es-posible/>, .
- [2] Ejemplo de silobolsa en el campo. <https://news.agrofy.com.ar/noticia/147548/nuevo-control-campo>, .
- [3] Aprendizaje supervisado: Introducción a la clasificación y principales algoritmos. <https://medium.com/datos-y-ciencia/aprendizaje-supervisado-introducci%C3%B3n-a-la-clasificaci%C3%B3n-y-principales-algoritmos-dadee99c9407>.
- [4] Neural networks as black box. <https://learnopencv.com/neural-networks-a-30000-feet-view-for-beginners/>.
- [5] Mnist train and test datasets. <https://github.com/mbornet-hl/MNIST/tree/master/IMAGES/GROUPS>, .
- [6] Understanding and visualizing convolutional neural networks. <https://medium.com/@SeoJaeDuk/archived-post-understanding-and-visualizing-convolutional-neural-networks-d6da3e2851cf>.
- [7] Convolutional neural networks - andrew ng - coursera course. <https://www.coursera.org/learn/convolutional-neural-networks>.
- [8] Intersection over union (iou) for object detection. <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>.
- [9] Voc2012: Pascal visual object classes. <http://host.robots.ox.ac.uk/pascal/VOC/>.
- [10] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015. URL <https://arxiv.org/abs/1506.02640>.
- [11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. URL <https://arxiv.org/abs/1804.02767>.
- [12] Narayana Darapaneni, M. S. Narayanan, Shakeeb Ansar, Ganesh Ravindran, Kumar Sanjeev, Abhijeet Kharade, and Anwesh Reddy Paduri. Traffic density determination using advanced computer vision. In V. Sivakumar Reddy, V. Kamakshi Prasad, D. N. Mallikarjuna Rao, and Suresh Chandra Satapathy, editors, *Intelligent Systems and Sustainable Computing*, pages 375–386, Singapore, 2022. Springer Nature Singapore. ISBN 978-981-19-0011-2.
- [13] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. URL <https://arxiv.org/abs/2004.10934>.
- [14] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for object detection, 2019. URL <https://arxiv.org/abs/1906.11172>.
- [15] Yolov4 — version 2: Bag of specials. <https://medium.com/visionwizard/yolov4-version-2-bag-of-specials-fab1032b7fa0>, .
- [16] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018. URL <https://arxiv.org/abs/1807.06521>.
- [17] Yolov5 using the pytorch framework. <https://github.com/ultralytics/yolov5>, .
- [18] Satellogic: Creating a searchable earth. <https://www.satellogic.com/>.

- [19] Kaggle: Your machine learning and data science community. <https://www.kaggle.com/>.
- [20] Google earth. <https://earth.google.com/web/@0.00000533,-39.4578002,-902.80278497a,22252656.11615181d,35y,0h,0t,0r>.
- [21] 12.000 piscines illégales détectées en provence grâce à l'intelligence artificielle. <https://www.francebleu.fr/infos/societe/dans-le-var-deja-4000-piscines-non-declarees-detectees-grace-a-l-intelligence-artificielle-1642074759>.
- [22] Kermap. <https://kermap.com/en/>.
- [23] Docker: A complete container solution. <https://www.docker.com/>, .
- [24] Redhat: ¿qué es una api de rest? <https://www.redhat.com/es/topics/api/what-is-a-rest-api>.
- [25] Flask: Web development, one drop at a time. <https://flask.palletsprojects.com/en/2.1.x/>.
- [26] Bootstrap: The most popular html, css, and js library in the world. <https://getbootstrap.com/>.
- [27] Dana Harry Ballard and Christopher M. Brown. Computer vision, 1982.
- [28] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis and machine vision* (3. ed.). 01 2008. ISBN 978-0-495-24438-7.
- [29] Thomas S. Huang. Computer vision: Evolution and promise. 1996.
- [30] Una introducción suave a los conceptos de aprendizaje automático. <https://medium.com/machine-learning-in-practice/a-gentle-introduction-to-machine-learning-concepts-cfe710910eb>.
- [31] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.
- [32] R G Morris. D.O. hebb: The organization of behavior, wiley: New york; 1949. *Brain Res Bull*, 50(5-6):437, November 1999.
- [33] Yann Lecun. A theoretical framework for back-propagation. 08 2001.
- [34] Martin A. Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993.
- [35] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL <https://proceedings.mlr.press/v28/pascanu13.html>.
- [36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. [https://doi.org/https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/https://doi.org/10.1016/0893-6080(89)90020-8). URL <https://www.sciencedirect.com/science/article/pii/0893608089900208>.
- [37] Coco and pascal voc data format for object detection - understanding annotation data formats for computer vision. <https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5/>.
- [38] Qgis: Un sistema de información geográfica libre y de código abierto. <https://www.qgis.org/es/site/>.
- [39] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016. URL <https://arxiv.org/abs/1612.08242>.
- [40] John A. Hartigan. *Clustering Algorithms*. John Wiley amp; Sons, Inc., USA, 99th edition, 1975. ISBN 047135645X.

- [41] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016. URL <https://arxiv.org/abs/1612.03144>.
- [42] Yolo v4: Optimal speed accuracy for object detection. <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>, .
- [43] Tesla: Inteligencia artificial y piloto automático. https://www.tesla.com/es_ES/AI.
- [44] Sysml open source project - what is sysml? who created sysml? <https://sysml.org/>.
- [45] Ian Sommerville. *Software Engineering*. Addison-Wesley, 2011. ISBN 978-0-13-703515-1.
- [46] Stackoverflow foro. <https://stackoverflow.com/questions>.
- [47] Nabucodonosor: Computadora nabucodonosor (ml). <https://ccad.unc.edu.ar/equipamiento/computadora-nabucodonosor/>.
- [48] Ssh tunnel: Definition. <https://www.ssh.com/academy/ssh/tunneling>.
- [49] Dockerhub: Container image library. <https://hub.docker.com/>, .
- [50] Nvidia cuda: A parallel computing platform and programming model. <https://developer.nvidia.com/cuda-zone>.
- [51] Virtualenv: A tool to create isolated python environments. <https://virtualenv.pypa.io/en/latest/>.
- [52] Yolototfrecords - use tensorflow detection api for training and evaluation. <https://github.com/mwindowshz/YoloToTfRecords>, .
- [53] Convolutional neural networks dicted by dr. andrew ng. <https://www.coursera.org/lecture/convolutional-neural-networks/object-localization-nEeJM>.