

# UNIVERSIDAD NACIONAL DE CÓRDOBA



---

## PROYECTO INTEGRADOR INGENIERÍA EN COMPUTACIÓN

---

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

---

*Autores:* Gianfranco  
BARBIANI, Loreley  
BUSTAMANTE

*Director:*  
Mg. Ing. Miguel SOLINAS  
*Codirector:*  
Dr. Ing. Miguel SOLINAS  
Jr.

5 de julio 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 futuros trabajos.

# Índice general

<b>1. Introducción</b>	1
1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	2
1.2.1. Objetivos Principales	2
1.2.2. Objetivos Particulares	4
1.3. Marco de investigación	4
<b>2. Marco Teórico</b>	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	9
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	13
2.5.1.4. Intersection Over Union	14
2.6. Dataset	18
2.6.1. Pre-entrenamiento	18
2.6.2. Formato de las Etiquetas	19
2.6.3. Dataset Personalizado	20
2.6.3.1. Distribución de Etiquetas	21
2.6.3.2. Correlograma de Etiquetas	22
<b>3. Estado del Arte</b>	24
3. Introducción	24
3.1. Detección de Objetos	24
3.2. YOLO: You Only Look Once	24
3.2.1. Funcionamiento de YOLO	26
3.2.1.1. Workflow	26
3.2.2. YOLOv3	27
3.2.2.1. Mejoras	27
3.2.2.2. Comparación con otras estructuras	30
3.2.3. YOLOv4	30
3.2.3.1. Mejoras	31

3.2.4. YOLOv5 .....	35
3.2.4.1. Mejoras .....	37
3.2.4.2. Memoria y FLOPS .....	37
3.2.4.3. Comparación de modelos de YOLOv5 .....	37
<b>4. Análisis del Diseño del Sistema .....</b>	<b>39</b>
4. Introducción .....	39
4.1. Entrevistas .....	39
4.2. Motivación e Importancia del proyecto .....	40
4.3. Casos de uso de Machine Learning - Detección de objetos .....	40
4.4. Funcionalidades .....	41
4.5. Definición de los Componentes .....	41
4.6. Definición de los Requerimientos .....	43
4.6.1. Diagrama de casos de uso .....	43
4.7. Definición del Comportamiento .....	46
4.8. Riesgos .....	47
4.8.1. Criterios .....	47
4.8.2. Identificación de Riesgos .....	48
4.8.3. Análisis de Riesgos .....	49
4.8.4. Planificación de Riesgos .....	50
4.9. Control de Versiones .....	51
<b>5. Implementación .....</b>	<b>52</b>
5. Introducción .....	52
5.1. Entornos de trabajo .....	52
5.1.1. Configuración Nabucodonosor .....	53
5.1.1.1. Machine Learning & Backend Docker .....	54
5.1.1.2. Frontend Docker .....	55
5.2. Dataset .....	55
5.2.1. Data Augmentation .....	55
5.2.2. Formatos .....	56
5.3. Entrenamiento .....	56
5.4. Pruebas .....	57
5.4.1. Precisión .....	57
5.4.2. Recall o Sensibilidad .....	58
5.4.3. Precision/Recall .....	59
5.4.4. F1 Score .....	60
5.4.5. Loss Function y Precisión Media Promedio (mAP, Mean Average Precision) .....	61
5.4.6. Matriz de Confusión .....	64
5.5. Despliegue .....	66
<b>6. Validación .....</b>	<b>67</b>
6. Introducción .....	67
6.1. Validación de los requerimientos de usuario .....	67
6.1.1. Matriz de trazabilidad .....	74

<b>7. Conclusión .....</b>	75
7. Introducción .....	75
7.1. Problemas y Limitaciones .....	75
7.2. Trabajos futuros .....	78
7.3. Aporte personal .....	78
<b>8. Apéndice .....</b>	79
8. Imágenes Dataset .....	79
9. Manual de Usuario .....	88
9.1. Introducción .....	88
9.2. Pre-requisitos .....	88
9.3. Instalación .....	88
9.4. Ingresar a la pagina .....	88
9.5. Cargar una imagen .....	88
9.5.1. Error en el formato de la imagen .....	89
9.6. Cargar varias imágenes .....	89
9.7. Inicio de la detección .....	89
9.8. Descarga de imágenes .....	90
9.9. Agradecimientos .....	91



## Índice de figuras

1. Riegos por pívot (circular): sistema de riego móviles mediante pivotes que permiten regar grandes superficies. ....	3
2. Silo bolsas: 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. ....	3
3. Workflow .....	4
4. Clasificación .....	7
5. Regresión .....	7
6. Ejemplo Linealmente Separable .....	8
7. Ejemplo Machine Learning .....	8
8. Abstracción de una Red Neuronal [12] .....	11
9. Tareas de Visión por computadora [22] .....	12
10. Bounding Box in YOLO .....	13
11. Non Max Supression in YOLO .....	14
12. Anchor Boxes en YOLO .....	15
13. Ground Thruth and Predicted Bounding Box .....	16
14. Intersetion over Union formula.....	16
15. Intersetion over Union examples .....	17
16. Las 20 clases de VOC2012 .....	18
17. Distribución de Etiquetas .....	21
18. Correlograma de Etiquetas .....	22
19. El sistema de detección de YOLO: Procesar imágenes con YOLO es simple y directo. Nuestro sistema (1) cambia el tamaño de la imagen de entrada a $448 \times 448$ , (2) ejecuta una sola red convolucional en la imagen y (3) establece un umbral para las detecciones resultantes según la confianza del modelo. [37] .....	25
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 [37]. .....	26
21. Predicciones de las cajas delimitadoras .....	28
22. Cajas delimitadoras con dimensiones previas y predicciones. Se predice el ancho y la altura de la caja como offsets desde el centro de la caja.Fuente [39] .....	29
23. Redes Piramidales de Características .....	30
24. Darknet-53 .....	31
25. Comparación 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[39] .....	32

26. Principales componentes de un Detector de Objetos moderno de una etapa .....	33
27. Ejemplo de Aumento de datos [43] .....	33
28. CutMix .....	34
29. Mosaic representa un nuevo metodo de data augmentation [28].....	34
30. Suavizado de Etiquetas .....	35
31. Módulo de Atención Espacial [? ] .....	36
32. Módulo de Atención Espacial Modificado [28].....	36
33. Arquitecturas: Memoria y FLOPS .....	37
34. YOLOv5 - Rendimiento .....	38
35. Workflow: Alcance .....	39
36. Arquitectura del Software Implementado .....	42
37. Diagrama de Caso de usos .....	43
38. Diagrama de Componentes .....	45
39. Diagrama de Secuencia .....	46
40. Workflow: Implementación .....	52
41. Workflow: Etapa de Datos .....	55
42. Workflow: Etapa de Modelado .....	56
43. Precisión .....	58
44. Recall .....	59
45. Precision/Recall .....	60
46. Curva F1 Score .....	60
47. Loss Function y mAP. 1eras 100 epochs. ....	62
48. Loss Function y mAP. Epochs 101 a 700. ....	63
49. Matriz de Confusión. 1eras 100 epochs. ....	64
50. Matriz de Confusión. Epochs 101 a 700. ....	65
51. Workflow: Etapa de Despliegue .....	66
52. Arquitectura del Software Implementado .....	66
53. Pagina principal .....	68
54. Subir una imagen para analizar .....	68
55. Seleccionar una imagen desde la PC-Usuario .....	69
56. Inicio de la detección - Loading .....	69
57. Error al subir una archivo de formato invalido .....	70
58. Fin de la detección - Resultado .....	71
59. Tabla de Información de detecciones - Resultado .....	72
60. Detección de un archivo con imágenes comprimidas .....	73
61. Descarga de archivo con imágenes detectadas - Resultados .....	74
62. Pivot etiquetado manualmente .....	79
63. Pivot etiquetado manualmente .....	80
64. Silobolsa etiquetado manualmente .....	80
65. Silobolsa etiquetado manualmente .....	81
66. Silobolsa etiquetado manualmente .....	81

67. Silobolsa etiquetado manualmente .....	82
68. Pivot generado con data augmentation.....	83
69. Pivot generado con data augmentation.....	84
70. Pivot generado con data augmentation.....	85
71. Silobolsa generado con data augmentation .....	86
72. Silobolsa generado con data augmentation .....	87
73. Subir una imagen para analizar .....	89
74. Error al subir una archivo de formato invalido .....	90
75. Inicio de la detección - Loading .....	90
76. Detección de un archivo con imágenes comprimidas .....	91
77. Descarga de archivo con imágenes detectadas - Resultados .....	92

## Índice de cuadros

1. Requerimientos de Usuario .....	44
2. Requerimientos Funcionales de la API .....	44
3. Requerimientos Funcionales del Frontend .....	44
4. Requerimientos No Funcionales del Sistema .....	45
5. Probabilidad de Riesgos Vs Efecto .....	47
6. Riesgos identificados para el proyecto .....	48
7. Riesgos según su probabilidad y efecto .....	49
8. Estrategias de manejo de riesgos - Parte 1 .....	50
9. Estrategias de manejo de riesgos - Parte 2 .....	51
10. Librerías Backend .....	54
11. Caso de Prueba del Requerimiento de Usuario R-01 .....	67
12. Caso de Prueba del Requerimiento de Usuario R-02 .....	69
13. Caso de Prueba del Requerimiento de Usuario R-03 .....	70
14. Caso de Prueba del Requerimiento de Usuario R-04 .....	71
15. Caso de Prueba del Requerimiento de Usuario R-05 .....	72
16. Caso de Prueba del Requerimiento de Usuario R-06 .....	73
17. Matriz de Trazabilidad .....	74



# 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 [16] y Kaggle [9].

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 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: [1] y también explotadas en el ámbito privado como lo hace la siguiente compañía: [10]

### 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 adquirir conocimientos sobre Deep Learning e Inteligencia Artificial y aplicar dichos conocimientos a desarrollar un prototipo para la detección de imágenes satelitales en un periodo de tiempo corto.
- Ayudar a combatir irregularidades detectándolas fácilmente. En particular detectando pivotes de riego 1 y silo bolsas 2. Ambos utilizados ampliamente en el rubro agrícola.



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



Figura 2: Silo bolsas: 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.

### 1.2.2. Objetivos Particulares

- Investigar sobre Machine Learning, su funcionamiento y sus casos de uso.
- Investigar y estudiar diferentes implementaciones de los modelos mas usados en lenguajes existentes.
- Evaluar en base a los requerimientos, cuál es la implementación o framework más adecuado para realizar este prototipo.
- Configurar un modelo en una supercomputadora con recursos necesarios, en un ambiente de desarrollo e implementarlo en un contenedor Docker [5] para que sea portable para ser instalado en cualquier computadora.
- Complementar el prototipo con una API [15] y una interfaz gráfica para mejorar su usabilidad.

### 1.3. Marco de investigación

El presente trabajo esta estructurado según el siguiente Workflow:

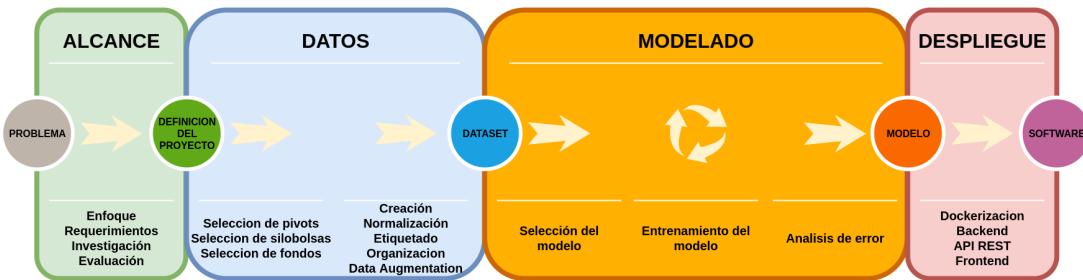


Figura 3: Workflow

1. Primero se definió el alcance de nuestro proyecto, para quién estaba enfocado y todo lo necesario para llevarlo a cabo, por lo que se decidió comenzar por una Investigación y posterior evaluación de los algoritmos mas utilizados para la detección de objetos en tiempo real, con el afán de lograr tiempos de detección reducidos.
2. Datos: Se continuó con la elección de los datos a utilizar, en este caso imágenes de riegos por pívot y silo bolsas con vista satelital. Se siguió con la creación, normalización y etiquetado de un conjunto de datos.
3. Modelo: Luego del análisis entre los modelos mas usados se optó por una primera elección de YOLOv3 (3), 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 al la versión 3 resultó ser mas efectivo para nuestro problema y con mejor rendimiento.

4. Despliegue: Después 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:
  - a) La primera parte consistió en la configuración del Algoritmo YOLOv5 en un entorno virtual contenido en un Docker con los requerimientos necesarios. 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 con el frontend (página web).
  - b) La segunda parte consistió en la programación de la API con la ayuda del framework Flask [7] 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 para que el usuario pueda interactuar con el sistema a través de una interfaz gráfica. Con los lenguajes de programación JavaScript, HTML y CSS y el framework Boostrap [2], se le dio la estructura y estilo a la pagina, dando una retroalimentación visual al usuario para indicar los resultados 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

Uno de las ramas de investigación más populares y utilizada de la Inteligencia Artificial es la Visión por Computadora o también llamada Visión Artificial [27] [42] [31]. 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. Sin embargo, tiempo atrás la capacidad de la visión por computador 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, [21] es esencialmente un conjunto de algoritmos diseñado por un ser humano. Sin embargo, los algoritmos aprenden de los datos, 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 qué hacer con ellos. 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.

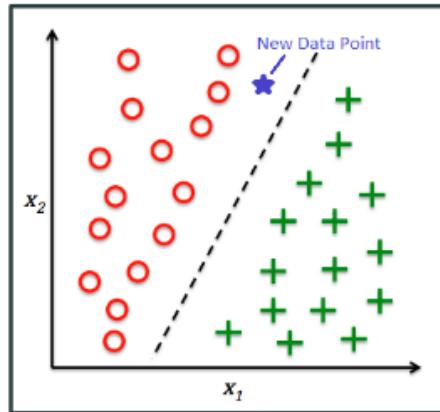


Figura 4: Clasificación

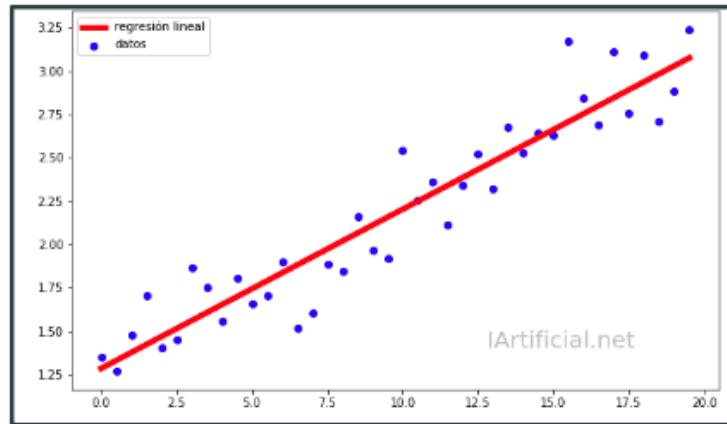


Figura 5: Regresión

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 [?]

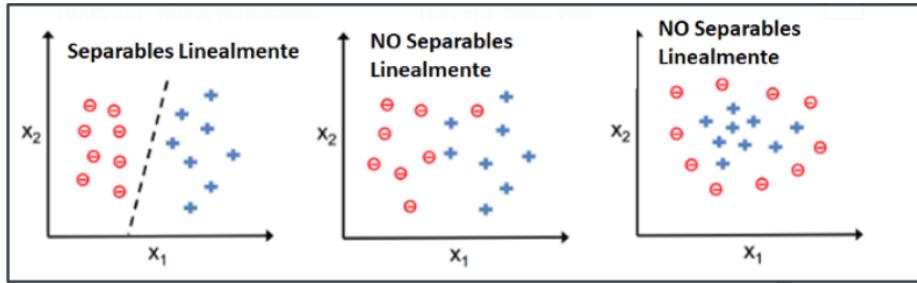


Figura 6: Ejemplo Linealmente Separable

#### 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 objetos correctamente.

Para ello se lo entrena a partir de imágenes ya correctamente (ground truth) etiquetadas (bounding box) 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.

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”.

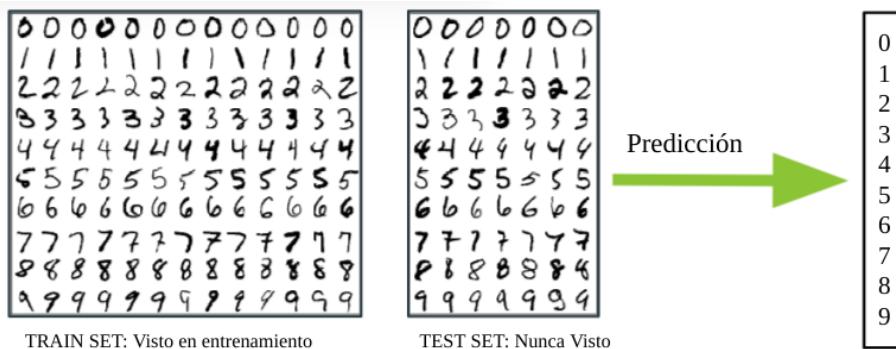


Figura 7: Ejemplo Machine Learning

### 2.3. Redes Neuronales

Las Redes Neuronales Artificiales (ANNs) son modelos matemáticos concebidos en un principio para estudiar (de cierta manera), el comportamiento del cerebro humano. Estos modelos matemáticos son llamados Redes Neuronales por que fueron inspirados por neuronas biológicas tal y como formalizaron, McCulloch y Pitts (1943) [34]; Hebb y Hebb (1949) [35]. las Redes Neuronales Artificiales son formadas por circuitos neuronales y las interconexiones entre neuronas, llamados 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) [32]. Backpropagation que continua como el “Hoyo en uno” de la investigación conexionista contemporánea, permite a una amplia gama de modelos ANNs a 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 ANN están limitadas en algún aspecto importante o no (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 lidar con muchos problemas.

En este capítulo, se introduce brevemente conceptos básicos relacionados al presente trabajo. Primero se introduce 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 (Computer Vision) 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 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 mapear correctamente las entradas  $x_i$  con las salidas  $y_i$ , mediante la adaptación de sus parámetros. 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 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 multiplicado por sus computaciones intermedias hacia la salida. En la mayoría de los casos, el mapeo  $f(x) : x \rightarrow y$  en la mayoría de los casos 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  $l$  que es utilizado para ponderar la entrada antes de ser transformadas por la función de activación. En cada 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:  $g_l(x) = (l, x)$  donde  $l$  es el parámetro de la capa y  $g$  es la función de activación de la capa. Una red neuronal comprende de una capa de entrada  $g_1$ , que es la primer capa de la red, una capa de salida  $g_3$  y varias capas intermedias. La ultima capa, la capa de salida, es donde se espera que este la predicción. El largo total de esta cadena representa la profundidad del modelo y es de donde proviene el termino aprendizaje profundo (Deep Learning).

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

El segundo paso, es llamado entrenamiento y consiste en apagar hacia atrás (backpropagation) una señal de error sobre los parámetros del modelo Riedmiller and Braun (1993) [40]. Una función de perdida (Loss function) es utilizada para computar el error cometido por el modelo en sus predicciones; esto es, la perdida es alta cuando el modelo esta haciendo un trabajo pobre y la perdida es baja cuando está performando 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 perdida 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 sampleados del set de entrenamiento. Cuando una red neuronal encuentra un set de parámetros “óptimo”, esta listo para ser deployado y comenzar 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 valido entre las entradas y las salidas. Por ejemplo, para evadir limitaciones lineales, funciones no lineales son utilizadas en capaz ocultas en vez de transformaciones lineales ya que provee 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. Mas específicamente, las funciones no lineales permiten al espacio de entrada ser doblado por lo que este espacio puede ser dividido en regiones lineales mas pequeñas Pascanu et al.(2013) [36]. La transformación no lineal, la correcta función de perdida y un apropiado numero de parámetros en las capas ocultas permiten a la red neuronal aproximar cualquier función de mapeo, este es el origen del termino aproximado universal Hornik et al. (1989). Para una introducción detallada de Redes Neuronales por favor referirse a Hornik et al. (1989) [30].

La función de mapeo aprendida  $f(, x)$  es continuamente 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  $F$ .

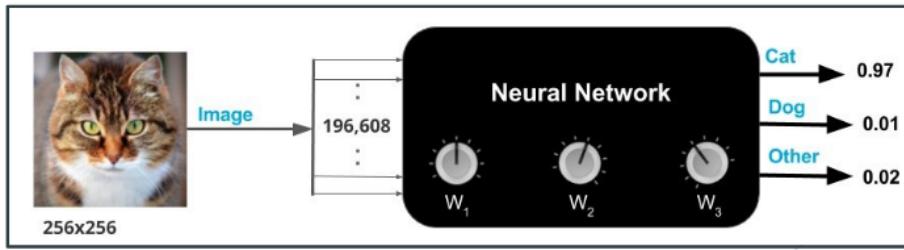


Figura 8: Abstracción de una Red Neuronal [12]

## 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 píxel de la imagen analizada.

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

Este proyecto optó para la *Detección de objetos (Object Detection)* como herramienta principal para resolver la problemática de la identificación y localización de pivotes de riego y silo bolsas en imágenes satélites dada la naturaleza

## Computer Vision Tasks

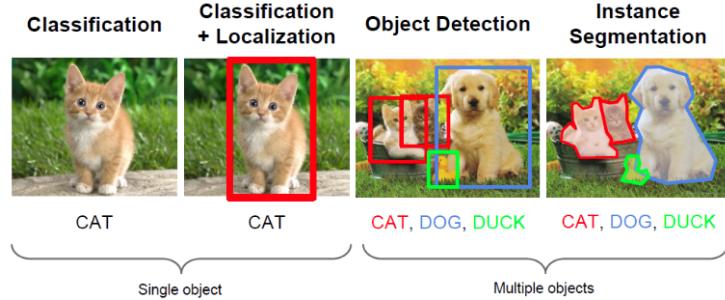


Figura 9: Tareas de Visión por computadora [22]

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 esta aplicación y que ameritan ser explicados. Sin embargo, dado que el modelo seleccionado en este trabajo está basado en YOLO (You Only Look Once) [37], 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 [3].

YOLO divide la imagen en celdas, donde cada una de ellas se “encarga” de detectar los objetos cuyo centro de Bounding Box 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.

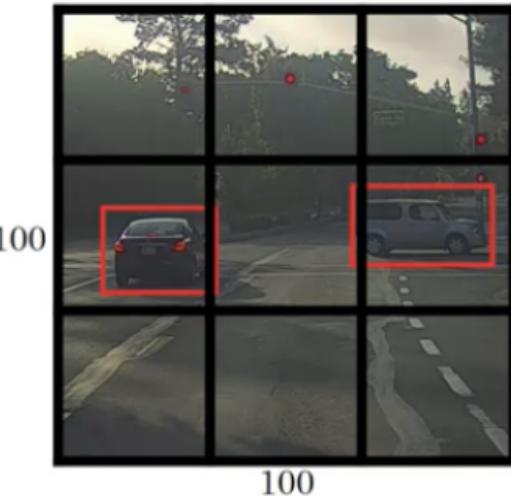


Figura 10: Bounding Box in YOLO

#### 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 y se selecciona la mejor Figura 11.

Procedimiento Non-max suppression [3] :

1. Examina las probabilidades asociadas con cada una de las detecciones.
2. Seleccionamos el BB con el  $P_c$  mas alto y lo elegimos como predicción parcial.  
Se descarta todos los BB con una  $P_c \leq 0,6$ .  
Si hay BB restantes:
3. Descartamos 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 [3].

Figura 12

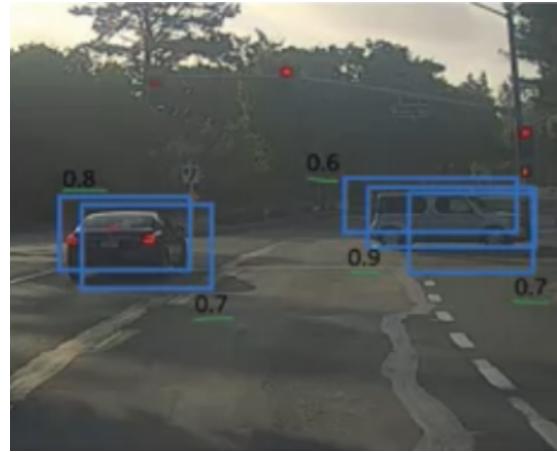


Figura 11: Non Max Supression in YOLO

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 mas avanzada, es decir automatizando la decisión utilizando un algoritmo de machine learning llamado K-Means.

#### 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 y evaluar un detector de objetos (arbitrario) necesitamos. Figura 13:

- 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.

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

- 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.

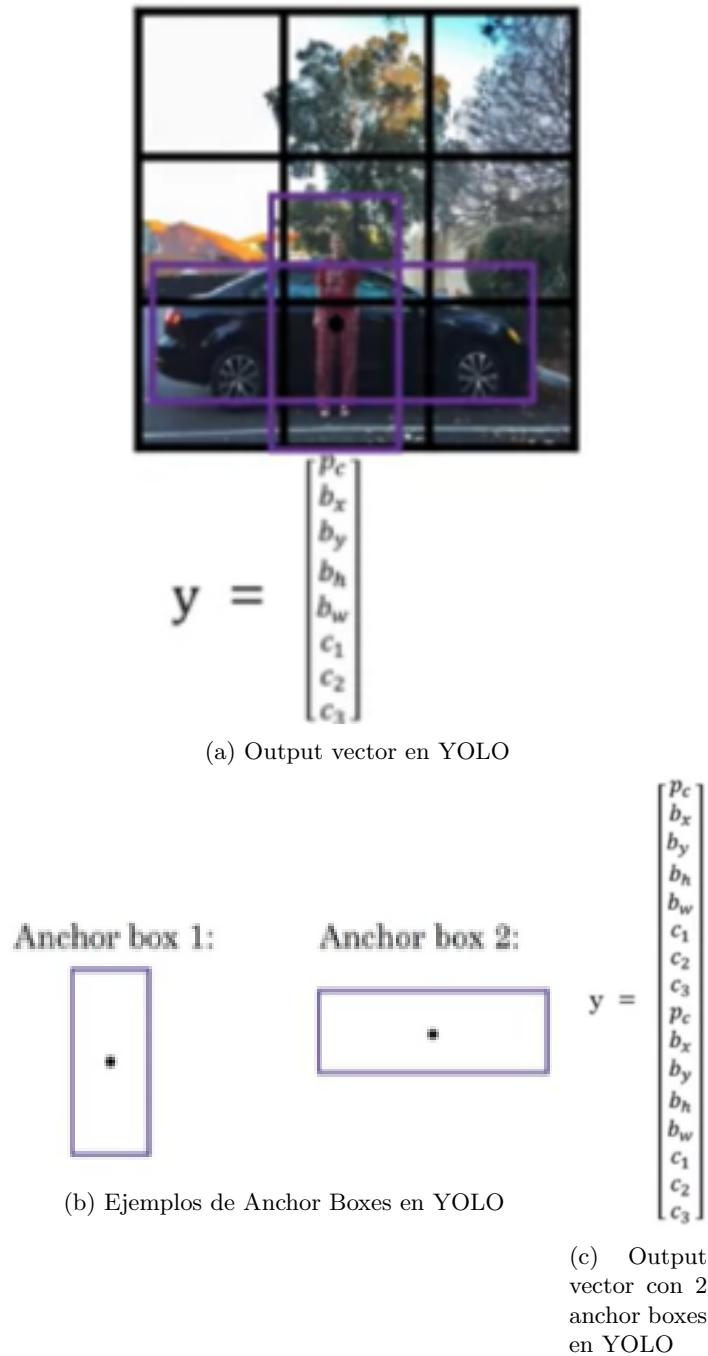


Figura 12: Anchor Boxes en YOLO

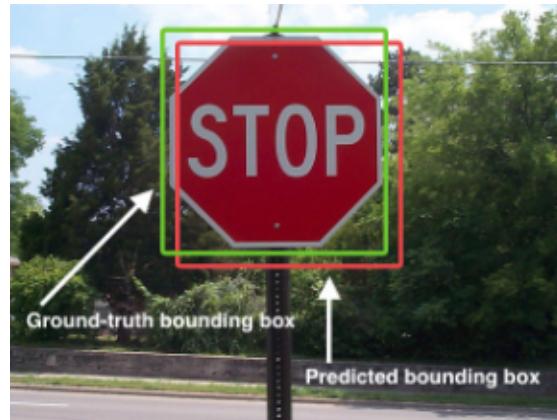


Figura 13: Ground Thruth and Predicted Bounding Box

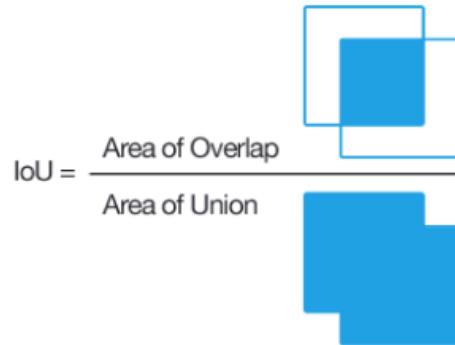


Figura 14: Intersetion over Union formula

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

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



Figura 15: Intersetion over Union examples

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.

## 2.6. Dataset

En esta sección se detallan las especificaciones del los datasets utilizados por el modelo YOLO.

### 2.6.1. Pre-entrenamiento

El modelo se entrenó desde cero (sus pesos establecidos aleatoriamente) usando el famoso dataset: VOC2012 Dataset [24]. 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).

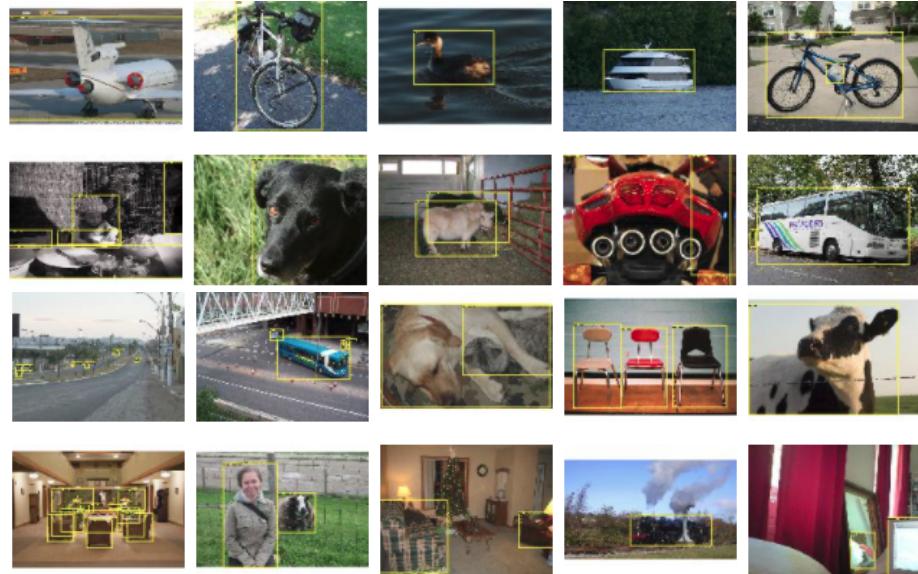


Figura 16: Las 20 clases de VOC2012

### 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:

---

```

<annotation>
  <folder>Kangaroo</folder>
  <filename>00001.jpg</filename>
  <path>./ Kangaroo/stock -12.jpg</path>
  <source>
    <database>Kangaroo</database>
  </source>
  <size>
    <width>450</width>
    <heighth>319</heighth>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>kangaroo</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <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 re-entrenó el modelo se basó en imágenes de pivotes y silobolsas extraídas de diferentes fuentes de acceso libre:

- Usando la herramienta QGIS [14] 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, relación de aspecto y resolución, esto es 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, relativamente la misma cantidad de objetos de cada clase.

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 numero tan grande de imágenes se optó por aplicar técnicas de Data Augmentation:

- Generación de Imágenes Sintéticas: Utilizando imágenes de “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 ??:

### 2.6.3.1 Distribución de Etiquetas

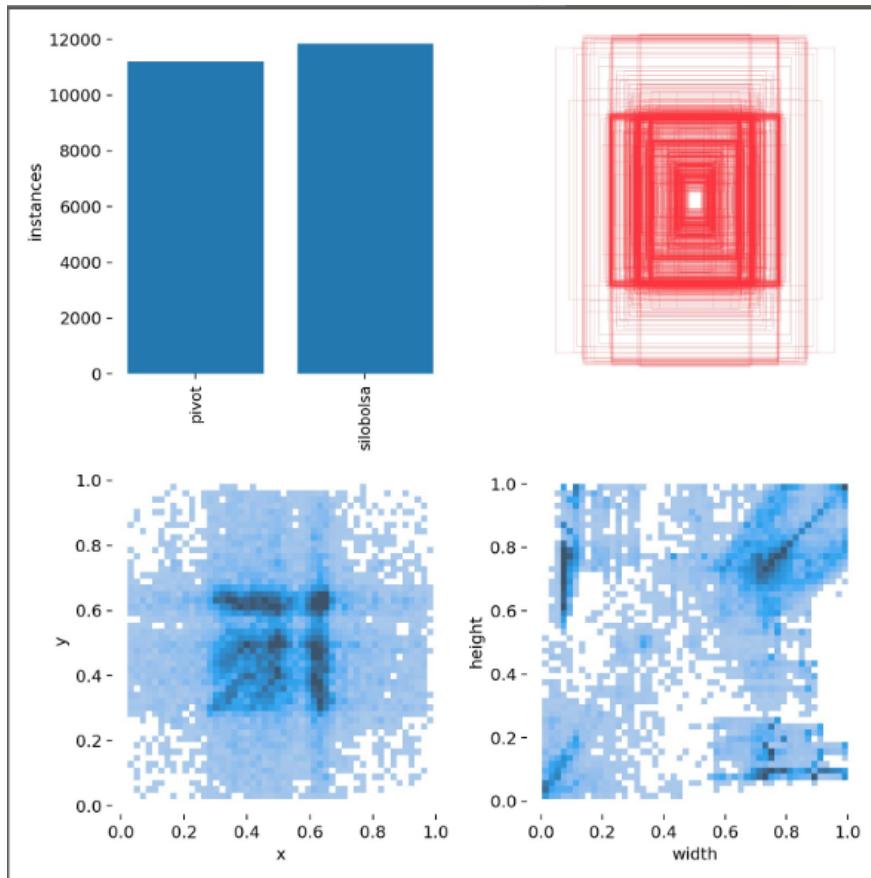


Figura 17: Distribución de Etiquetas

De la anterior Figura 17 se observan las siguientes conclusiones:

- En el gráfico ubicado en la esquina superior izquierda se puede ver la distribución de etiquetas por clase. Notar que están balanceadas: relativamente la misma cantidad de pivotes que de silobolsas.
- De las imágenes de la columna derecha se ve que los BB 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 de-

recha (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

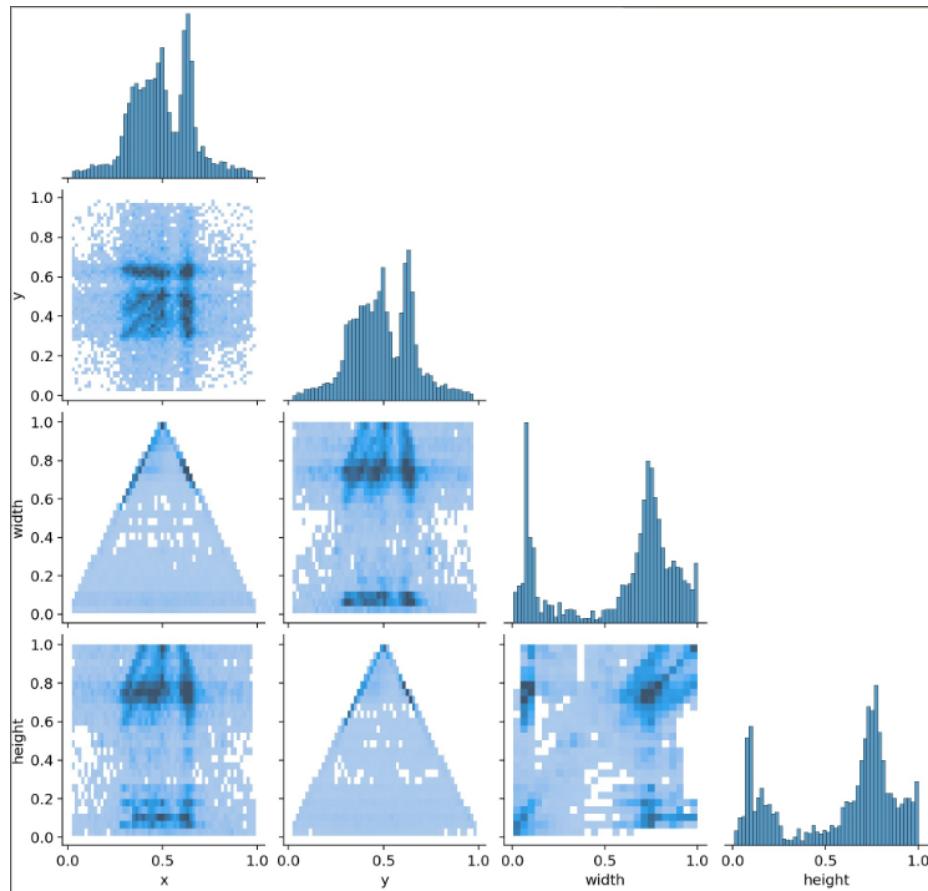


Figura 18: Correlograma de Etiquetas

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 entre 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 Detector de objetos muy eficaz. YOLO [37] 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. Lo cual para nosotros era un problema debido a que los objetos de nuestro interés aparecen en la mayoría de las imágenes como objetos pequeños.

YOLO [37] (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 [26] - [26] lanzada el 18 Mayo 2020, la cual es una versión modificada de YOLOv4, pasando por las versiones YOLOv2 [38] (25 Diciembre 2016), YOLOv3 [39] (8 Abril 2018) y YOLOv4 [28] (23 April 2020).

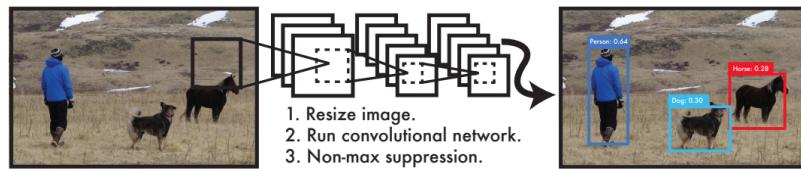


Figura 19: El sistema de detección de YOLO: Procesar imágenes con YOLO es simple y directo. Nuestro sistema (1) cambia el tamaño de la imagen de entrada a  $448 \times 448$ , (2) ejecuta una sola red convolucional en la imagen y (3) establece un umbral para las detecciones resultantes según la confianza del modelo. [37]

### 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, lo podemos ver en la siguiente Figura 20.

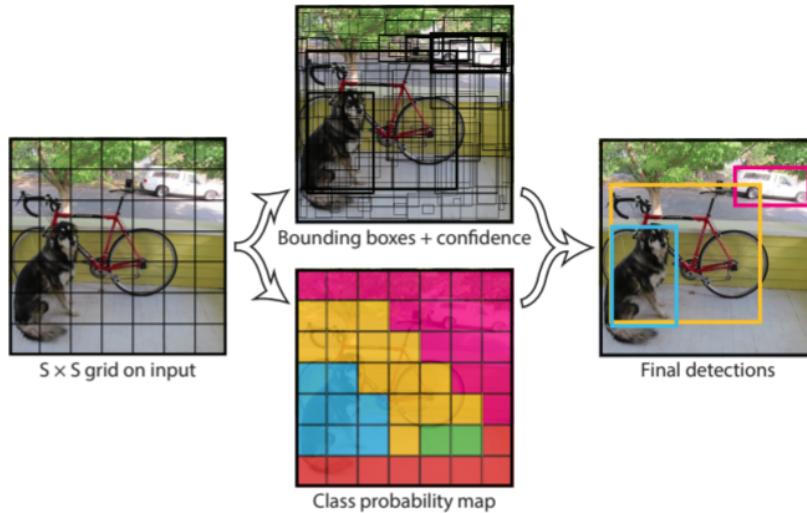


Figura 20: Funcionamiento de YOLO: primero se divide la imagen en cuadrículas más pequeñas, sobre éstas se calculan diferentes cajas delimitadoras o bounding boxes y las probabilidades de las distintas clases que se encuentre en ellas,y finalmente se obtienen las detecciones. Fuente [37].

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

- Intersection over Union
- 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 que 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  $(x, y)$  representan las coordenadas del centro de la caja detectada y  $(w, h)$  la anchura y altura respectivamente.
4. Además, cada celda de la cuadrícula también predice C probabilidades condicionales  $P(\text{Clase } i - \text{Objeto})$ , 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 y nos indica la probabilidad de que cada clase aparezca en las cajas detectadas y que tan bien se ajusta dicha caja.

### 3.2.2. YOLOv3

A continuación vamos a hablar de YOLOv3 [39], 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 mencionaron brevemente las mejoras que se implementaron en YOLOv3 con respecto a la primera versión YOLO, cuales 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 podemos definir dichas cajas de anclaje para que se adecuen mejor a la forma de los objetos de nuestro dataset.

- Clusters dimensionales: Como hemos mencionado, en muchos problemas de detección los objetos tienen una forma determinada. Para calcular las K cajas a priori que mejor cubren esta gama de formas en un dataset, se utiliza el algoritmo KMeans[29].

*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 [38]. 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

Es decir, la red predice offsets con respecto a cajas de anclaje (anchor boxes). Podemos 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, podemos definir dichas cajas de anclaje para que se adecuen mejor a la forma de los objetos de nuestro 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.

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

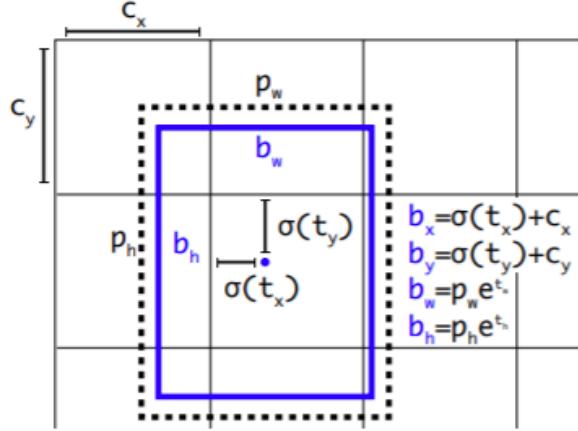


Figura 22: Cajas delimitadoras con dimensiones previas y predicciones. Se predice el ancho y la altura de la caja como offsets desde el centro de la caja. Fuente [39]

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 nos ayuda a comparar la precisión de las predicciones de nuestro 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 e utilizando un concepto similar a las redes piramidales de características o feature pyramid networks [33] (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 encargada de predecir un tensor 3d codificando las predicciones de cajas delimitadoras, la probabilidad de contener un objeto y predicciones de clase.

Así, el tensor final es de la forma  $N \times N \times (\text{Nboxes} (4 + 1 + \text{Nclasses})$  donde Nboxes es el número de cajas predichas en cada escala, 4 se corresponde a los offsets de la caja delimitadora, 1 a la predicción del objeto o probabilidad de contener un objeto y Nclasses es el número de clases del dataset.

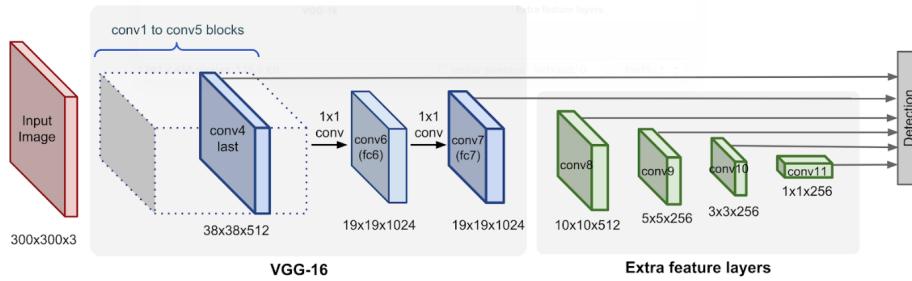


Figura 23: Redes Piramidales de Características

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 usando concatenación. Este método nos 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) [38] y capas residuales. Utiliza capas convolucionales sucesivas de  $3 \times 3$  y  $1 \times 1$ , con algunas interconexiones o atajos. Tiene 53 capas convolucionales en total y recibe el nombre de Darknet-53. Podemos ver un resumen de su estructura en la Figura 24.

#### 3.2.2.2 Comparación con otras estructuras

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

#### 3.2.3. YOLOv4

En esta sección hablaremos de YOLOv4 [28], 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 podemos observar en la siguiente Figura 26.

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

Figura 24: Darknet-53

Podemos además encontrar una descripción muy detallada junto a los parámetros que utiliza la red en este enlace. Sin embargo, aunque es una propuesta muy interesante para la detección de objetos en tiempo real por su gran velocidad en cuanto a FPS 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 vamos a mencionar 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

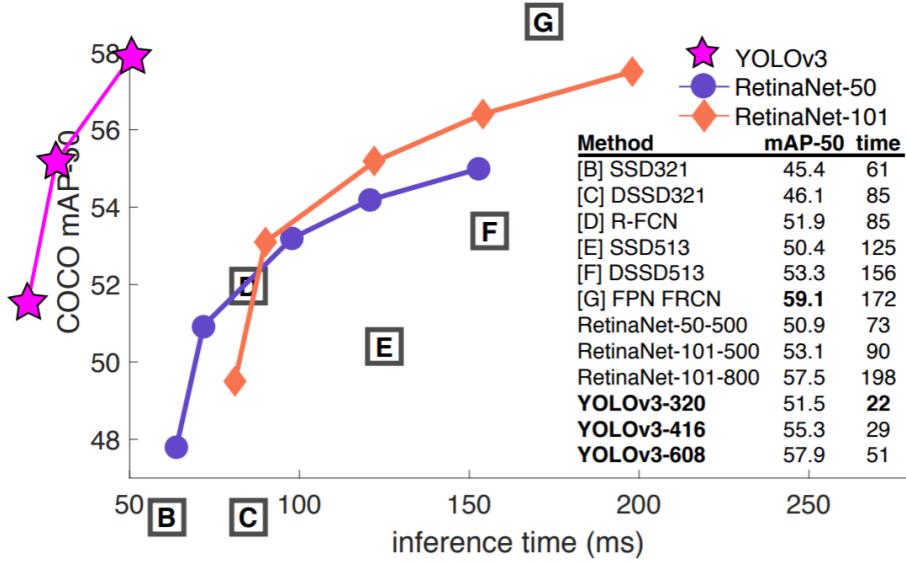


Figura 25: Comparación distintas estructuras de redes neuronales convolucionales: se observa el balance entre velocidad y precisión para 0.5 IOU, destacando YOLOv3 por su gran velocidad y manteniendo una buena precisión. Fuente[39]

Esta nueva arquitectura trae nuevos beneficios con respecto a la versión anterior de YOLO, entre ella podemos 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 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)

Son métodos que solo cambian la estrategia de entrenamiento o incrementan el costo de entrenamiento. 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.

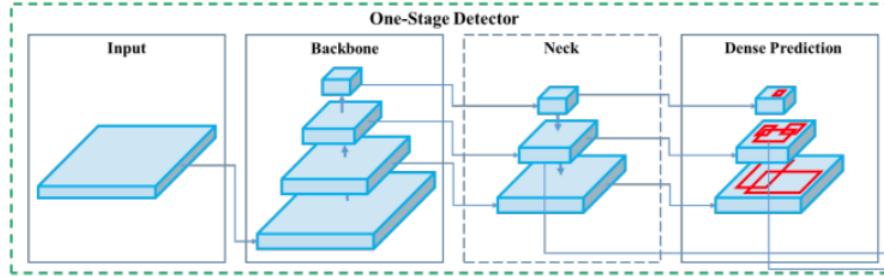


Figura 26: Principales componentes de un Detector de Objetos moderno de una etapa.

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

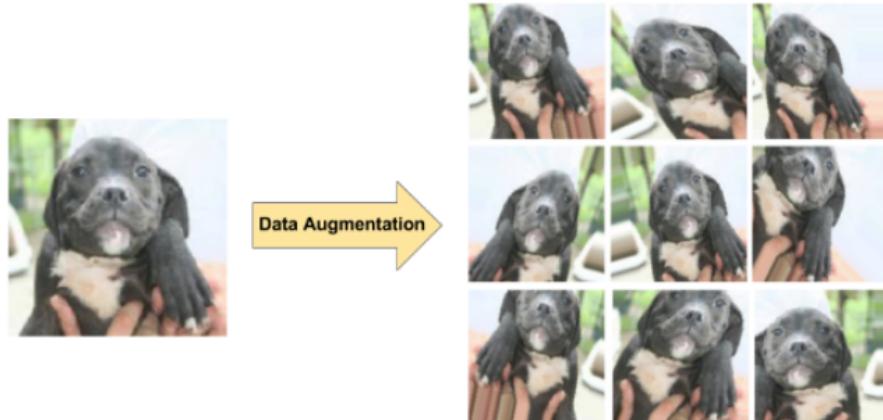


Figura 27: Ejemplo de Aumento de datos [43]

1. CutMix: En lugar de simplemente eliminar píxeles como en Cutout Figura 28, reemplazamos 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 Figura 29, en ciertas proporciones (en lugar de solo dos en CutMix). Permite que el modelo



Figura 28: CutMix

aprenda a identificar objetos a una escala menor de lo normal y mejorar la precisión de sus modelos hasta en un 10 %. Combina clases que pueden no verse juntas en su conjunto de entrenamiento.



Figura 29: Mosaic representa un nuevo metodo de data augmentation [28]

3. Class label smoothing: Es una técnica de regularización que aborda ambos problemas: overfitting 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 thruth por un factor  $\alpha$ :

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

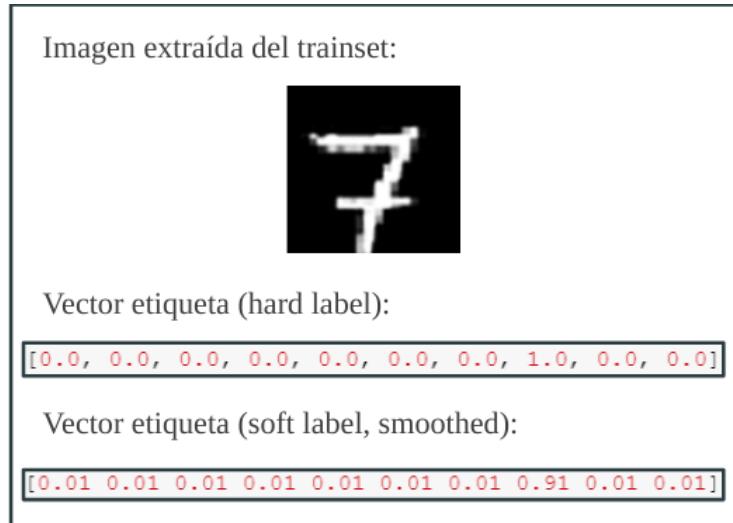


Figura 30: Suavizado de Etiquetas

### Bag of Specials (BoS)

Son módulos y métodos de post procesamiento que aumentan un poco el costo de inferencia pero pueden mejorar significativamente la precisión de la detección de objetos. A continuación los mencionamos:

#### 1. Spatial Attention Model (SAM)

Se aplica una máscara de atención espacial a las features de entrada, para obtener un feature map refinado: Se destacan así las características más relevantes creadas por las capas convolucionales y se remueven las menos importantes. Producido una mejora la clasificación y la detección, con un costo computacional bajo. En YOLOv4, una versión modificada de SAM [28] 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 utilizando el framework de Pytorch [26], el cual reduce sustancialmente el costo computacional de entrenamiento.

No existe documentación oficial sobre YOLOv5 a la fecha pero se sabe que cuenta los las siguientes mejoras:

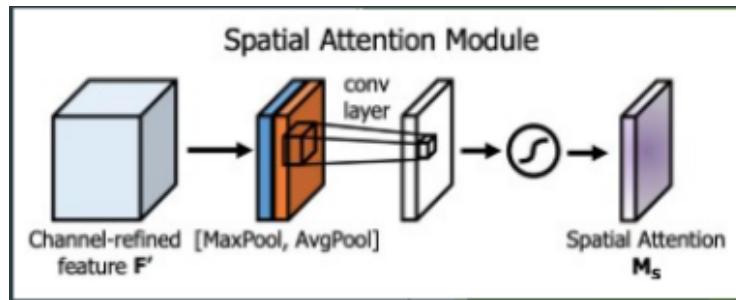


Figura 31: Módulo de Atención Espacial [? ]

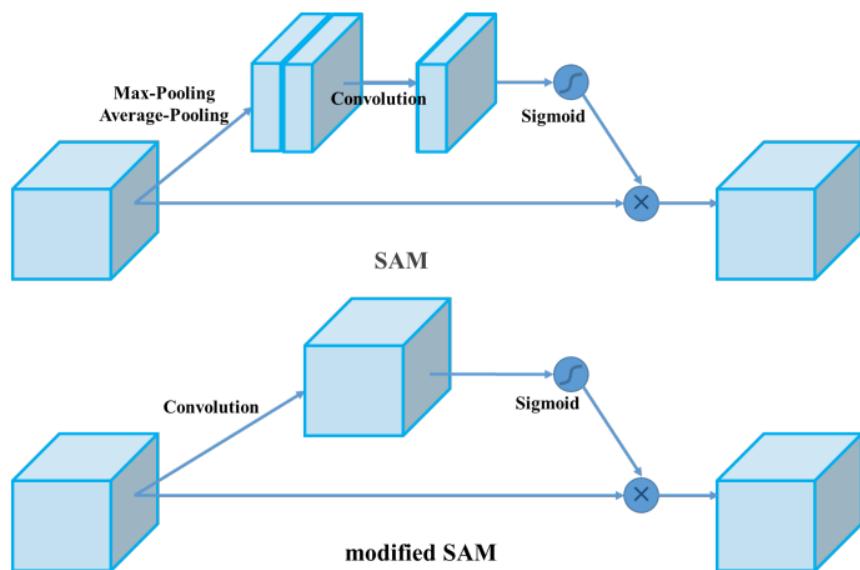


Figura 32: Módulo de Atención Espacial Modificado [28]

### 3.2.4.1 Mejoras

1. Es una implementación de YOLO v4 utilizando un framework en Python llamado Pytorch.
2. Posee muchas optimizaciones algorítmicas y aritméticas.
3. Se reduce sustancialmente el costo computacional de entrenamiento.

### 3.2.4.2 Memoria y FLOPS

Cuando se habla de un modelo que es “liviano” hablamos de su uso de la *memoria RAM* necesaria para entrenar. La cantidad necesaria viene dada por la cantidad de parámetros que el modelo seleccionado tiene para optimizar. Si mantenemos 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 vemos claramente cómo el cambio de arquitectura en YOLO v5l (large), con una cantidad de parámetros incluso menor que YOLO v3, arroja resultados más performantes (mayor FPS) y precisos (mayor AP - precisión promedio-) que este último.

Model	AP <sup>val</sup>	AP <sup>test</sup>	AP <sub>50</sub>	Speed <sub>GPU</sub>	FPS <sub>GPU</sub>	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 33: Arquitecturas: Memoria y FLOPS

### 3.2.4.3 Comparación de modelos de YOLOv5

En la siguiente Figura (34), se puede observar como YOLOv5 hace un excelente rendimiento en la detección de objetos, especialmente el YOLO V5S, con mas 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 este fin en comparación con otras técnicas de vanguardia.

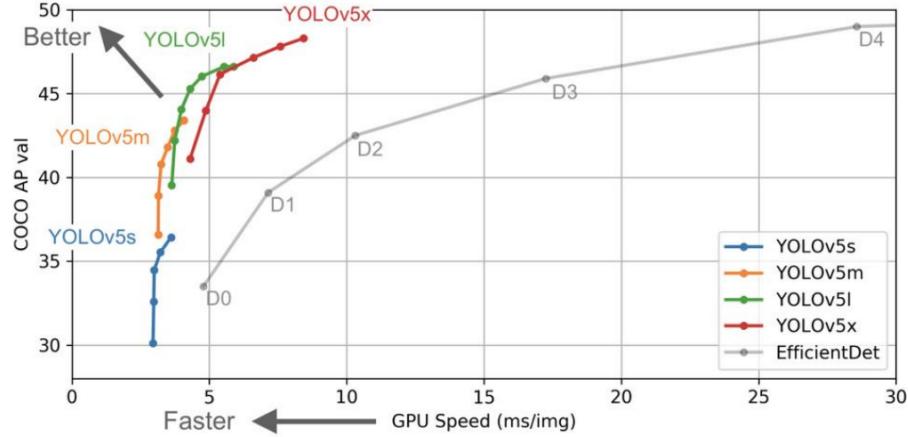


Figura 34: YOLOv5 - Rendimiento

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 los stakeholders interesados en hacer uso de la aplicación resultante de este trabajo final donde nos 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.

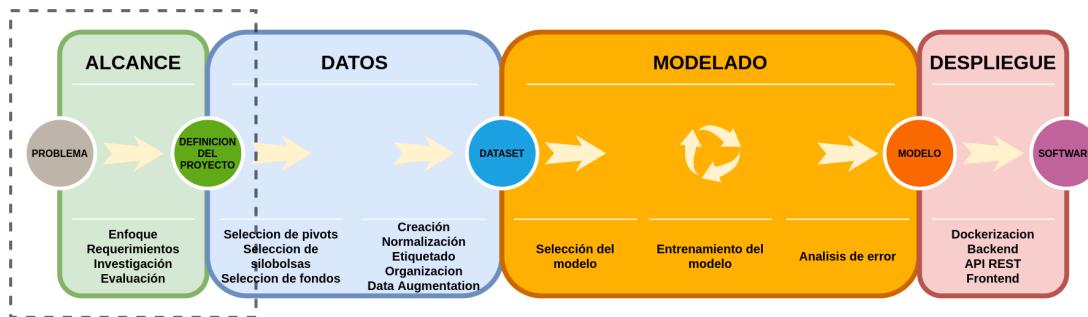


Figura 35: Workflow: Alcance

#### 4.1. Entrevistas

Las partes interesadas, como conocedores del área de las imágenes satelitales y el sector agropecuario, fueron entrevistados con el objetivo de entender las primeras especificaciones para el prototipo planteado en la sección 5 y para lograr entender mejor de los problemas del sistema de riego que se usa en el campo, como así también el almacenamiento de los cereales, legumbres, entre otros, y el uso de este prototipo y el beneficio del mismo para un futuro.

De la charla con los stakeholders se plantearon los siguientes problemas siguientes: El primer problema es un problema de digitalización:

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

Las soluciones propuestas que surgieron de la entrevista fueron:

- Debido a la fuerte influencia de algoritmos de Machine Learning de detección, se propuso hacer uso de esta tecnología para desarrollar una aplicación que no provee lo siguiente:
  - Detectar múltiples objetos, es este caso los objetos en cuestión son las silo-bolsas y los riegos por pivotes.
  - Dar la posición X e Y del objeto en la imagen (o su centro) y dibujar un rectángulo a su alrededor.
  - Otra alternativa es Detectar “a tiempo”. Esta es una característica que debemos tener en cuenta si por ejemplo queremos hacer detección en tiempo real sobre vídeo.
  - Otra de las propiedades de la detención de objetos que nos provee es su carácter de velocidad y precisión para la detección.

#### **4.2. Motivación e Importancia del proyecto**

Ante las preocupaciones anteriormente planteadas buscamos proponer una solución para satisfacer esas necesidades que no estaban cubiertas, por lo cual enumeramos las motivaciones que tuvimos 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, desde reconocimiento facial en tiempo real (celulares), sugerencias de compras, autos autónomos, entre otras, hoy en día están todas a nuestro alcance y es una tecnología que está madura y adoptada en nuestra 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 que ofrece respuestas adecuadas, tales como son las traducciones automáticas, las recomendaciones de vídeos, 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 [20] vienen de serie con un hardware avanzado capaz de ofrecer las funciones de Piloto automático y capacidades de conducción autónoma total basado en una IA avanzada 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 [15], 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 API 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. **Elección del Modelo :** El modelo elegido luego de realizar varias pruebas y con el cual se obtuvieron los mejores resultados fue: Yolo v5, con un excelente rendimiento en la detección de objetos, en comparación a sus versiones anteriores.
2. **Elección del lenguaje de programación:** El framework empleado para el entrenamiento y en el cual esta basado YOLOv5 [26] en Pytorch y Python.
3. **Desarrollo de una API:** El código escrito en JavaScript es el código cliente que invoca al servidor. Para que sea accesible a través de Internet es necesario el desarrollo de una API REST, que permita llamar las funciones con métodos HTML. La herramienta elegida para esa tarea es Node.js, es un entorno en tiempo de ejecución multiplataforma para la capa del servidor basado en JavaScript. Node.js es un entorno controlado por eventos diseñado para

crear aplicaciones escalables, permitiéndote establecer y gestionar múltiples conexiones al mismo tiempo y flexible para aplicaciones web y APIs.

4. **Desarrollo de una GUI:** La API públicamente expuesta a Internet necesita un frontend, o una interfaz web, que acceda 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 usuario y enviarlos para su procesamiento a la API. Para lograr eso, se decidió utilizar Bootstrap [2], donde las páginas que adaptan su contenido de forma dinámica a medida que reciben entradas del usuario, en vez de descargar páginas nuevas de un servidor. La herramienta 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 muy sencillo.

#### 5. Diagrama general de la arquitectura

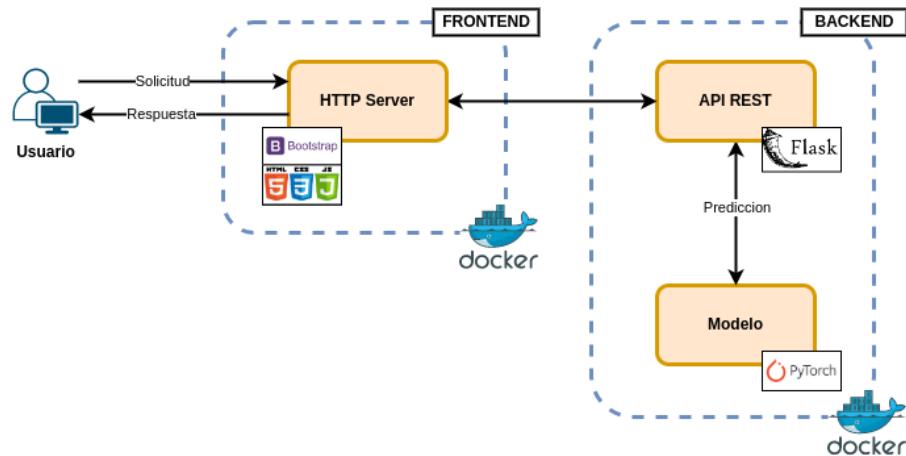


Figura 36: Arquitectura del Software Implementado

## 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 [19] no se consideró.

Antes de plantear los requerimientos, primero es necesario definir un diagrama de casos de uso, como se ve en la Figura 37. 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

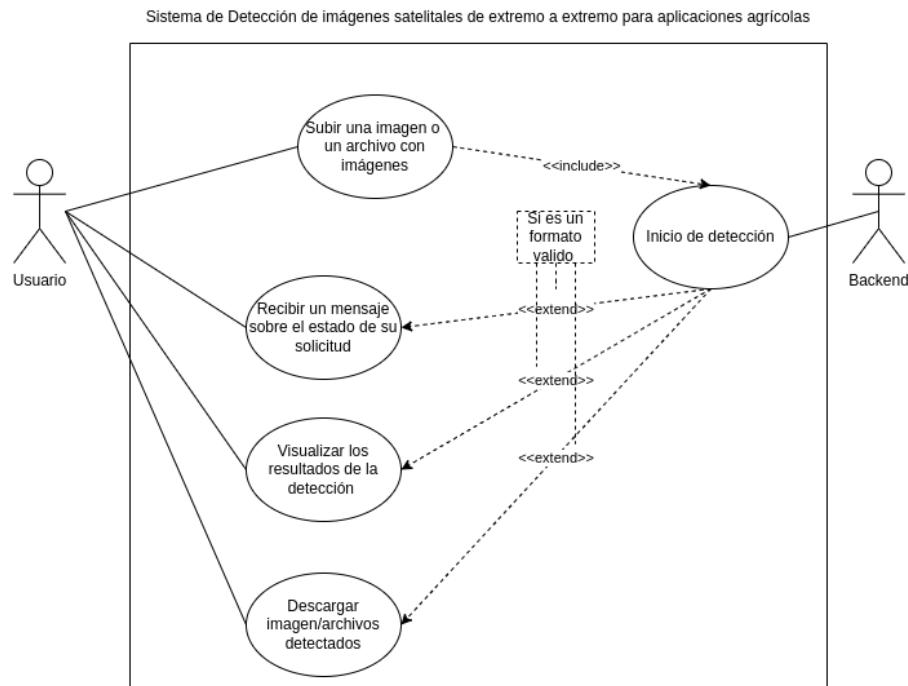


Figura 37: Diagrama de Caso de usos

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. 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 silo bolsas y pivotes de riego sobre los archivos recibidos.
RF-API-04	Retornar un archivos con la/s imágenes 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 empleados en el prototipo, se ofrece un diagrama de componentes, realizado en la Figura 38. Se distinguen 3 componentes principales:

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

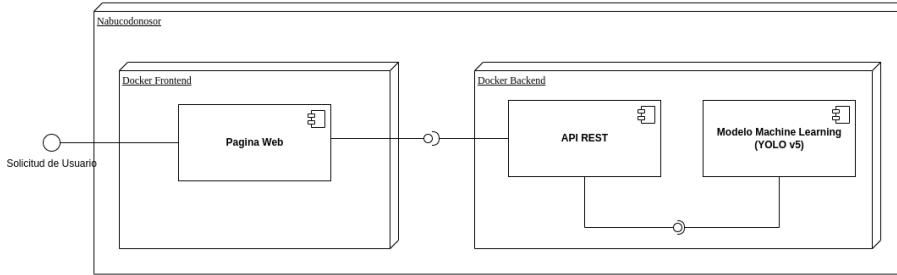


Figura 38: Diagrama de Componentes

#### 4.7. Definición del Comportamiento

Para entender cómo interactúan los componentes diagramados, se emplea un diagrama de secuencia en la Figura 39, 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 la detección correspondientes.

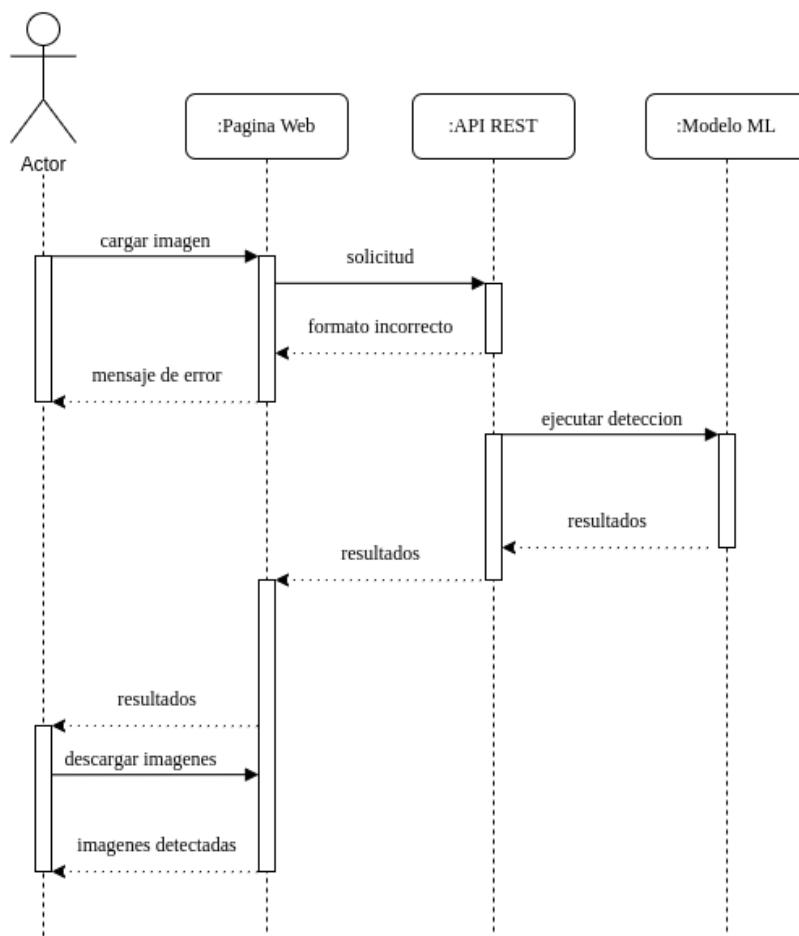


Figura 39: Diagrama de Secuencia

## 4.8. Riesgos

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

En presente sección se establecen diferentes criterios para identificar posibles riesgos, asignar prioridades a los mismos, evaluar su probabilidad y encontrar estrategias que permiten 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 siguientes probabilidades y gravedades que se tuvieron en cuenta según Ian Sommerville en su libro Software Engineering [41]:

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

Cuadro 8: Estrategias de manejo de riesgos - Parte 1

Código	Riesgo	Probabilidad
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"> <li>■ Lectura de tutoriales y realización de cursos cortos para lograr una introducción en temas cuyo desconocimiento impide el avance del proyecto.</li> <li>■ Empleo y modificación/adaptación de ejemplos previstos para el aprendizaje.</li> <li>■ Búsqueda de explicaciones adicionales en foros de usuarios como StackOverflow [18].</li> <li>■ Consultas a programadores o profesionales experimentados en los campos en cuestión.</li> </ul>
Riesgo-06	Rechazo de la implementación por parte del usuario final.	<ul style="list-style-type: none"> <li>■ Explicación de la tecnología empleada y proporcionar manual de usuario para su uso.</li> <li>■ Prueba de piloto para identificar mejoras en cuanto a la usabilidad y posibilidad de mejoras en un futuro.</li> </ul>

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](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 los datos obtenidos, para luego entrenar el modelo con la correcta monitorización de los resultados y, finalmente, desplegar el software final.

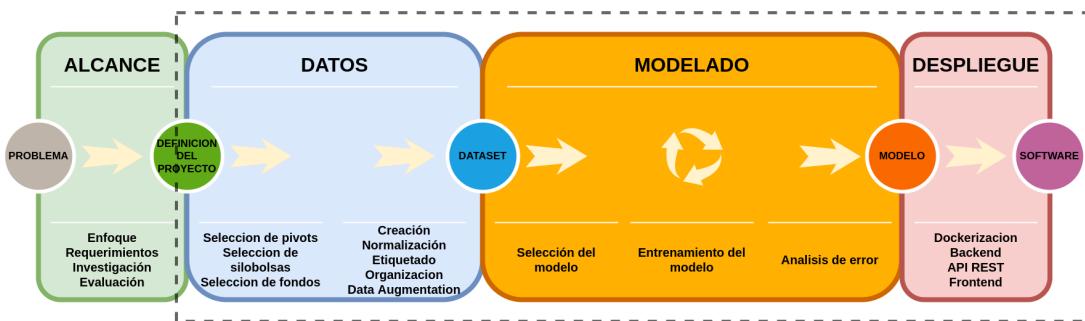


Figura 40: Workflow: Implementación

### 5.1. Entornos de trabajo

Para el desarrollo de este proyecto, se hizo uso de 2 computadoras con el sistema operativo Linux instalado (distintas distribuciones de Ubuntu), con conexión a internet y de la super computadora de la UNC Nabucodonosor (*Nabu*) [11]. 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 [17], 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 modificaría 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 *vainilla* 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* [6], 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* [13] 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"> <li>◦ matplotlib</li> <li>◦ numpy</li> <li>◦ opencv</li> <li>◦ Pillow</li> <li>◦ PyYAML</li> <li>◦ requests</li> <li>◦ scipy</li> <li>◦ torch</li> <li>◦ torchvision</li> <li>◦ tqdm</li> </ul>
Para Logueo	<ul style="list-style-type: none"> <li>◦ tensorboard</li> </ul>
Para Gráficas	<ul style="list-style-type: none"> <li>◦ seaborn</li> <li>◦ pandas</li> </ul>
Para Backend	<ul style="list-style-type: none"> <li>◦ flask</li> <li>◦ pyunpack</li> <li>◦ flask_cors</li> </ul>
Para Monitoreo de Recursos	<ul style="list-style-type: none"> <li>◦ htop</li> </ul>

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* [23]. 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 [43] y se realiza aplicando diferentes transformaciones a las imágenes como invertirlas, zoom, rotaciones entre otras dando una mayor generalización y una reducción del overfitting.

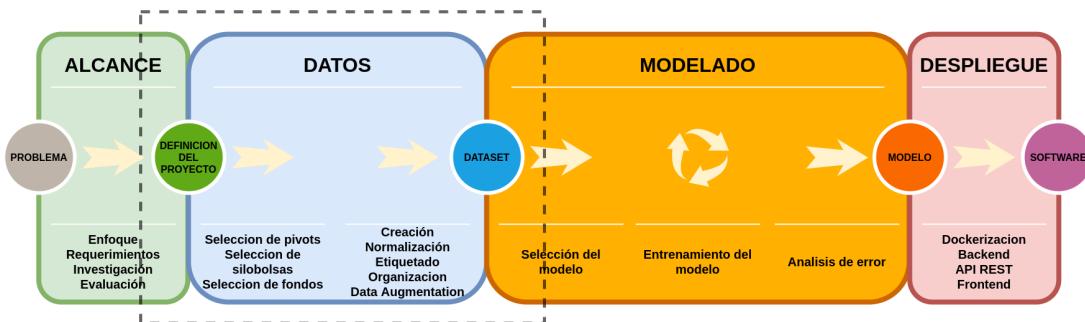


Figura 41: Workflow: Etapa de Datos

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 combinarián 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* [25]. 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

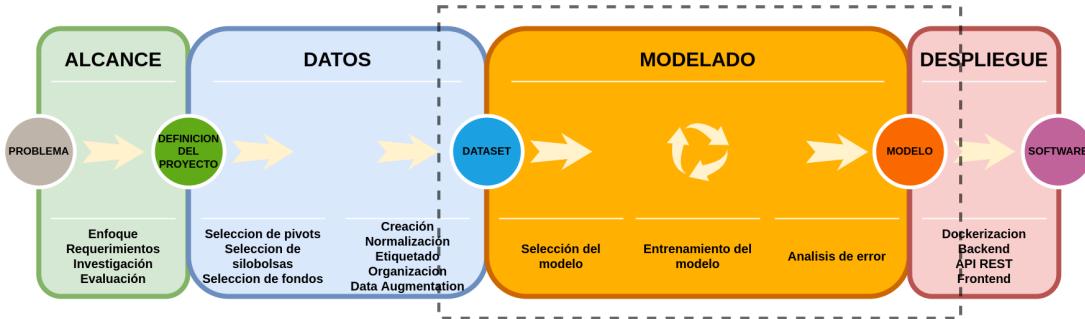


Figura 42: Workflow: Etapa de Modelado

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 ML. 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

En la Figura 43, podemos observar 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.

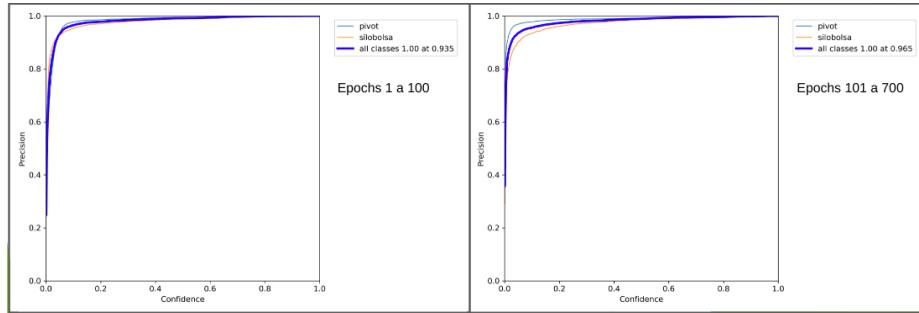


Figura 43: Precisión

### 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 que hemos 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

En la Figura 44 podemos observar 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.

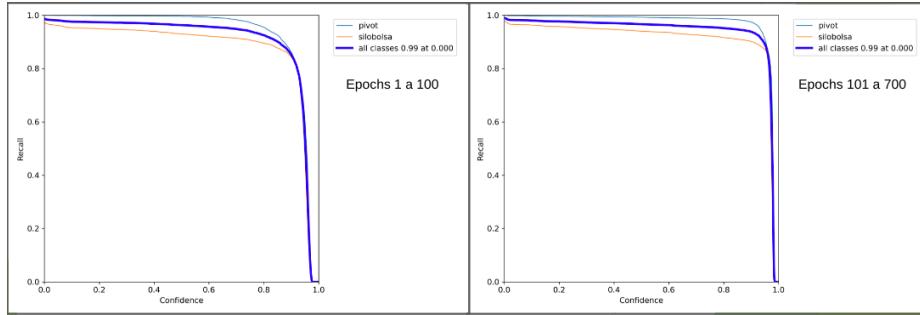


Figura 44: Recall

### 5.4.3. Precision/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 45 podemos observar la curva de Precision/Recall obtenida para la detección de Silobolsas y Pivot desde la Epoch 1 al 100 y de la Epoch 101 a la 700.

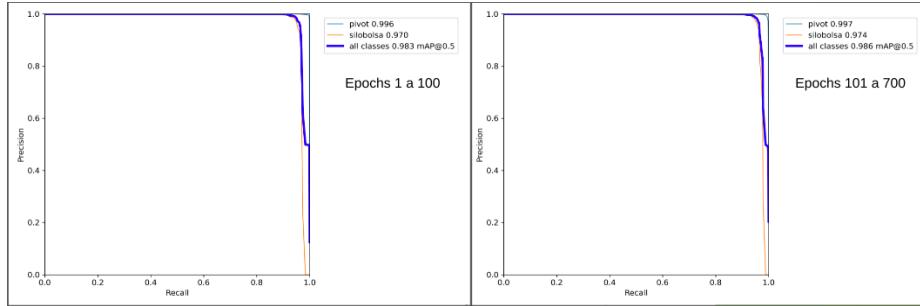


Figura 45: Precision/Recall

#### 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

En la Figura 46 podemos observar 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.

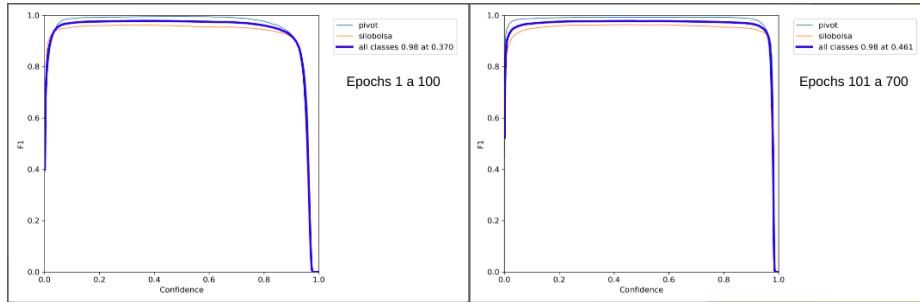


Figura 46: Curva F1 Score

#### 5.4.5. Loss Function y Precisión Media Promedio (mAP, Mean Average Precision)

En las siguientes imágenes se exponen los valores que toman las 3 loss functions del modelo (bounding box, detección de objeto, clasificación) para los conjunto de validación y testeo; así como también las curvas de precision, recall y mAP generales (ver figuras 47 y 48).

Luego de las 1ras 100 epochs de entrenamiento, se observan los siguientes resultados:

- En cuanto a métricas, los resultados ya eran muy buenos al finalizar la epoch 100.
- Las 3 losses de validación continuaban bajando al finalizar el entrenamiento, lo cual indicaba que aún había lugar para continuar mejorando los resultados, sin overfitting.
- 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 losses 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 ya cumplían con creces las expectativas.

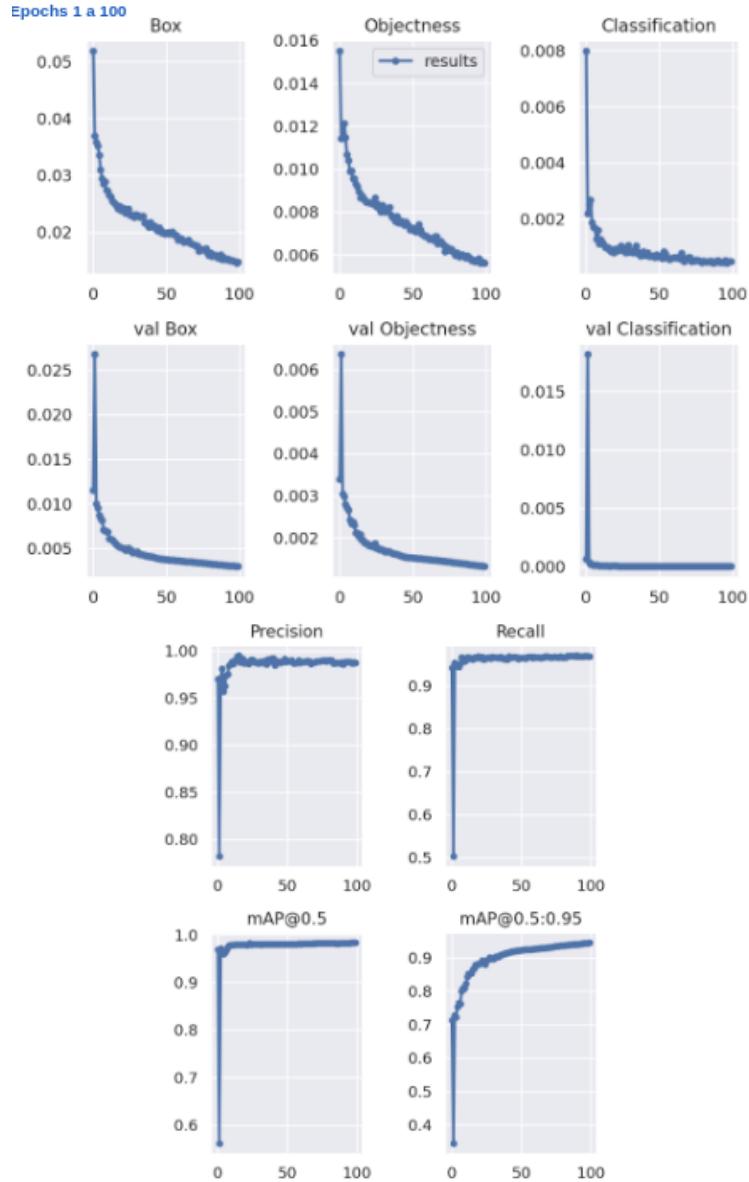


Figura 47: Loss Function y mAP. 1eras 100 epochs.

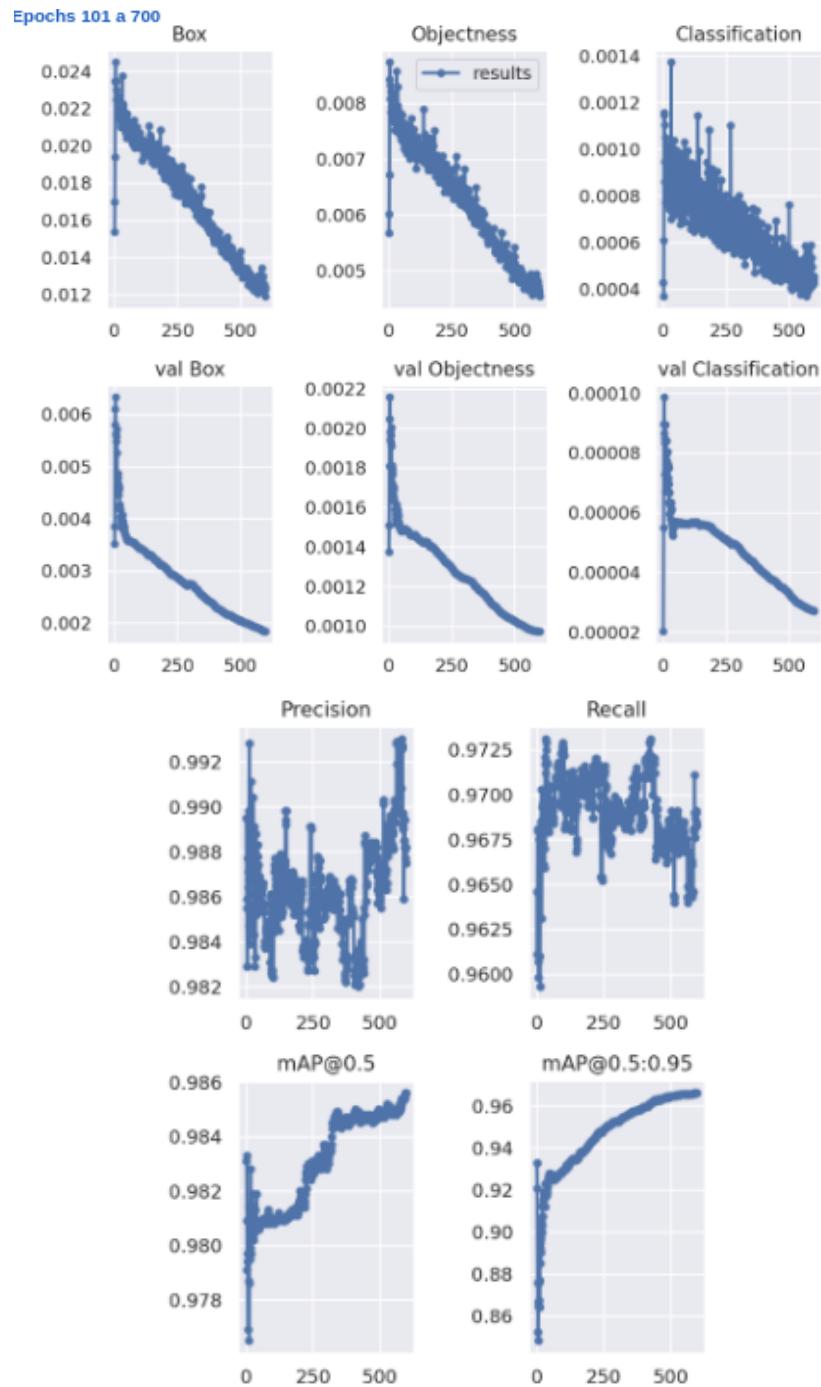


Figura 48: Loss Function y mAP. Epochs 101 a 700.

#### 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.

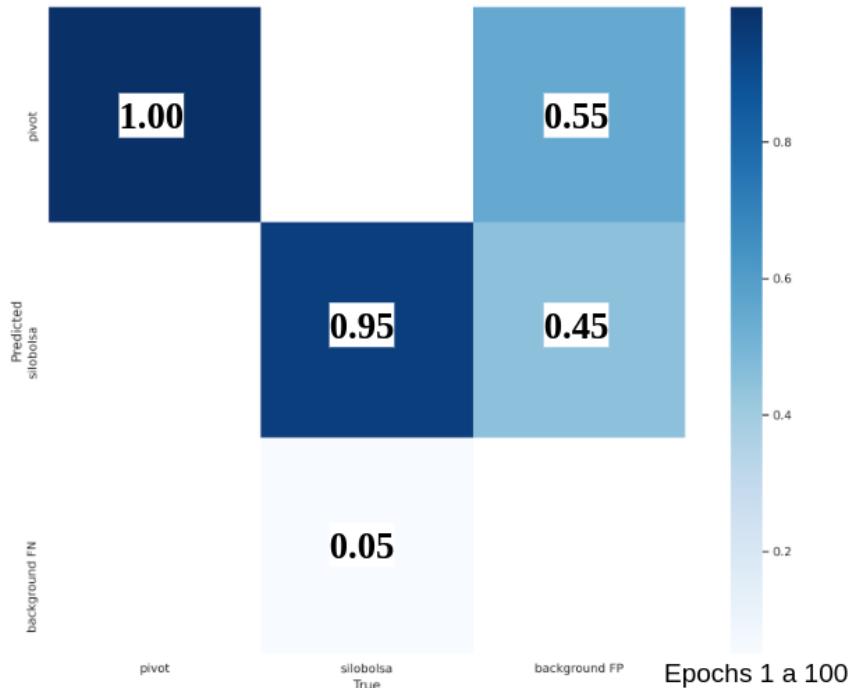


Figura 49: Matriz de Confusión. 1eras 100 epochs.

Al finalizar las 1ras 100 epochs, se ve en el gráfico de la izquierda que las detecciones de Pívot son excelentes, es decir, cada pívot fue reconocido, por eso el resultado 1.00. Sin embargo, de las falsas predicciones (FP), más de la mitad de las mismas eran de Pívot (55 %). Por otro lado, en el caso de las silobolas, se detectaron correctamente el 95 % de las mismas, con una proporción de FP de 45 %.

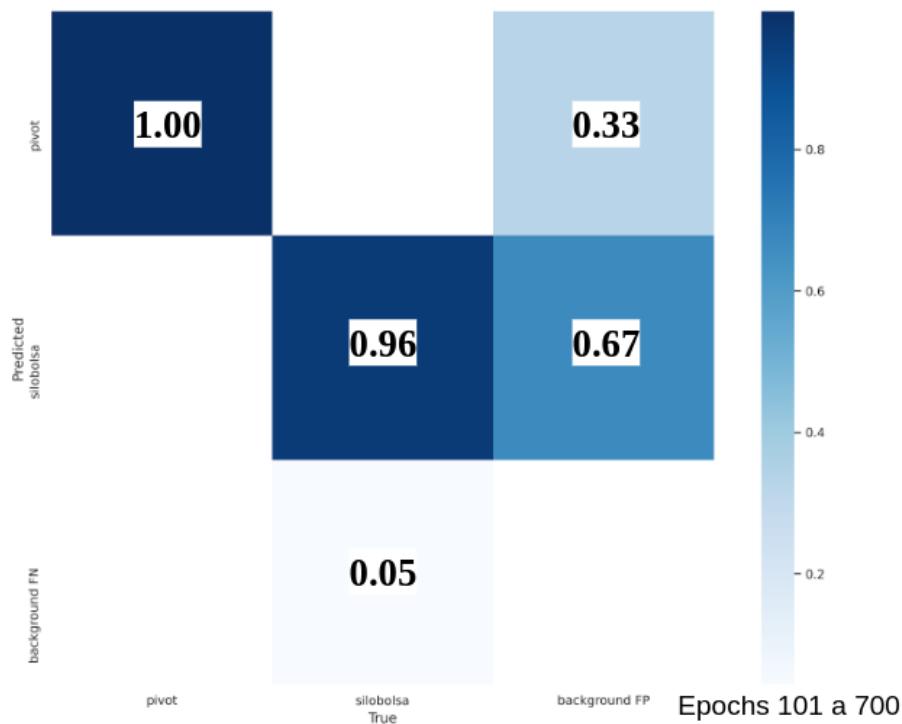


Figura 50: Matriz de Confusión. Epochs 101 a 700.

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

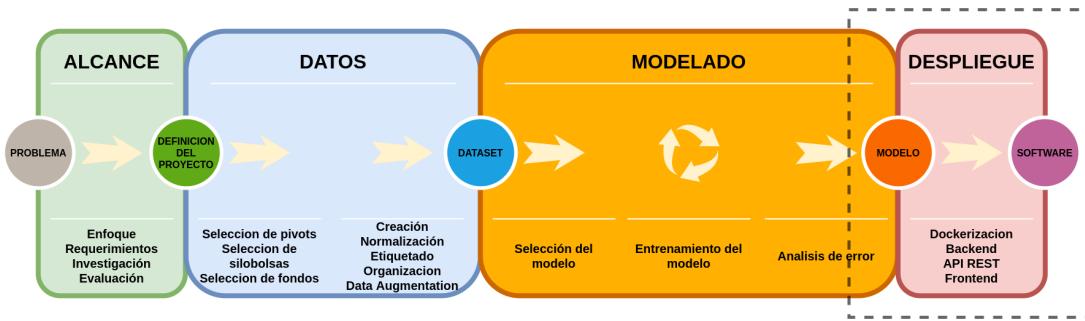


Figura 51: Workflow: Etapa de Despliegue

Para la etapa de despliegue se optó por hacer uso de los 2 contenedores Docker descritos linea arriba: uno corriendo el Backend y otro el Frontend. Este enfoque permite que el software en su totalidad sea fácilmente portado a diferentes sistemas.

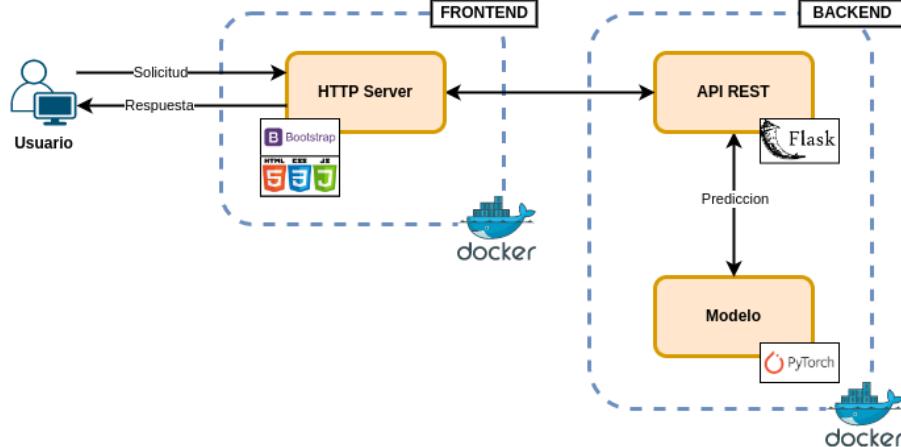


Figura 52: Arquitectura del Software Implementado

## 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 detallaran 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 imagen nueva al sistema, desde la pagina principal.
Objetivo	Lograr que el usuario pueda cargar una imagen de diferentes formatos de forma exitosa.
Procedimiento	<ol style="list-style-type: none"><li>El usuario debe dirigirse hacia la página principal y hacer clic en el cuadro con el título “Seleccionar archivo”.</li><li>Luego debe seleccionar la imagen o un archivo de imágenes a ser analizadas desde su computadora.</li><li>Después se indicará si la imagen fue subida exitosamente o no.</li></ol>
Resultados esperados	Éxito de la operación al subir una imagen.- Figuras 53 73 55
Estado	APROBADO

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



Figura 53: Pagina principal

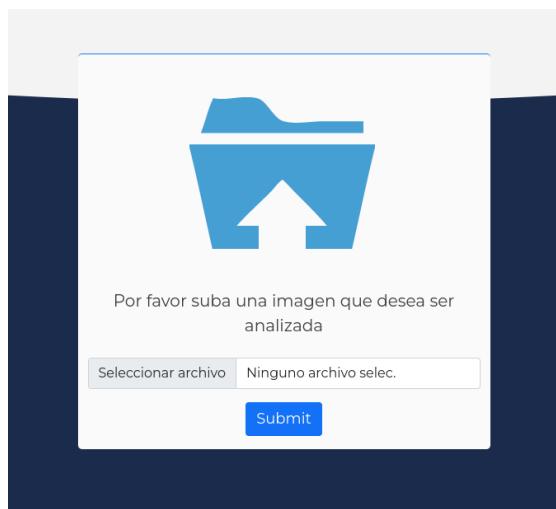


Figura 54: Subir una imagen para analizar

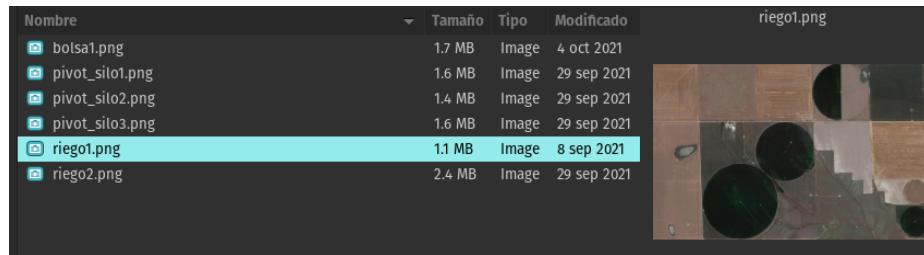


Figura 55: Seleccionar una imagen desde la PC-Usuario

TC-02	Validad 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"> <li>El usuario debe dirigirse a la pagina y luego de subir una imagen debe observar si el formato es aceptado.</li> <li>Luego debe ver habilitado el botón “Submit” para dar inicio a la Detección.</li> </ol>
Resultados esperados	Éxito en la presencia de la imagen subida.- Figura 73, 55 y 75
Estado	APROBADO

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

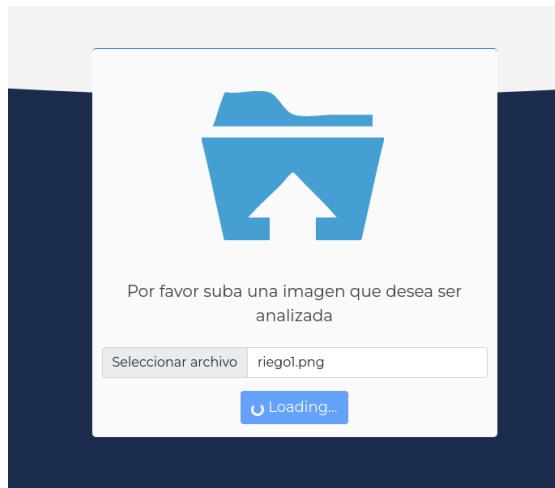


Figura 56: Inicio de la detección - Loading

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"> <li>El usuario debe dirigirse a la pagina y luego de subir una imagen errónea debe observar un mensaje de imagen invalida.</li> <li>Luego debe ver habilitado el botón para volver a cargar una imagen nueva.</li> </ol>
Resultados esperados	Éxito en la presencia de un mensaje de formato invalido.- Figura <a href="#">74</a> muestra el mensaje de error.
Estado	APROBADO

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

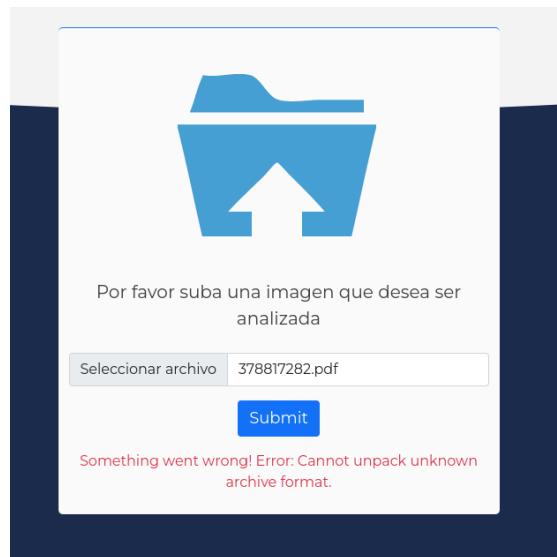


Figura 57: Error al subir una archivo de formato invalido

TC-04	Validar que debe ser posible visualizar en la pagina 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"> <li>1. El usuario debe dirigirse a la pagina, luego de subir/selecccionar una imagen y hacer clic en el botón de 'Submit', iniciará la detección automáticamente,</li> <li>2. Luego, como resultado debe ver la imagen resultante con las detecciones encontradas.</li> <li>3. Además debe observar una tabla con el historial de todas las detecciones y los detalles de las mismas.</li> </ol>
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 58 y 59.
Estado	APROBADO

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

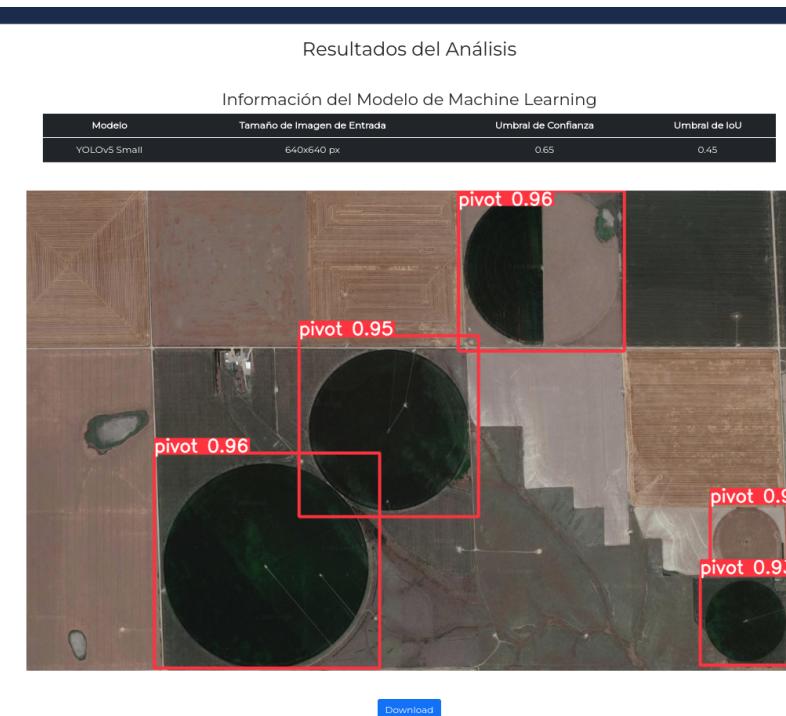


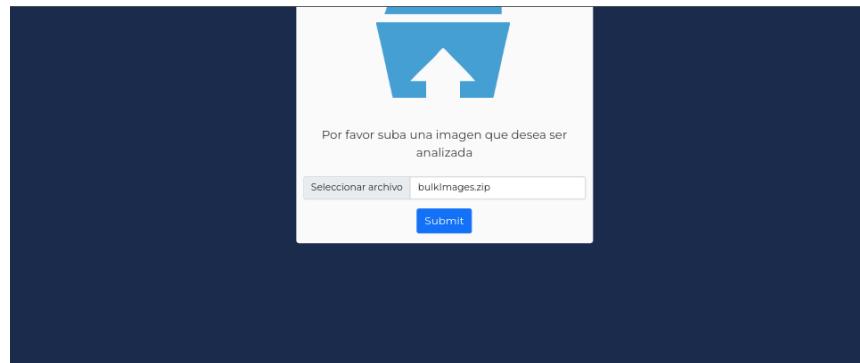
Figura 58: Fin de la detección - Resultado



Figura 59: 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 que tipo corresponde: silobolsa o riego por pivote.
Procedimiento	<ol style="list-style-type: none"> <li>1. El usuario luego de solicitar que se analice una imagen, debe visualizar el resultado.</li> <li>2. La imagen resultante debe contener cuadros delimitadores encerrando a los objetos detectados e indicando su tipo.</li> </ol>
Resultados esperados	Éxito en la presencia del resultado de la detección con información adicional .- Figura 76 detalla el historial y los resultados de la detección. Figura 59 muestra el resultado en caso de haber subido una sola imagen.
Estado	APROBADO

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



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 60: 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"> <li>El usuario una vez que pueda visualizar la imagen analizada, debe tener la opción de descargarla a su máquina local.</li> <li>La imagen analizada y descargada debe encontrarse en la máquina local del usuario.</li> </ol>
Resultados esperados	Éxito en la presencia de la imagen o un archivo con varias imágenes analizadas en la máquina local del usuario.- Figura 77.
Estado	APROBADO

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

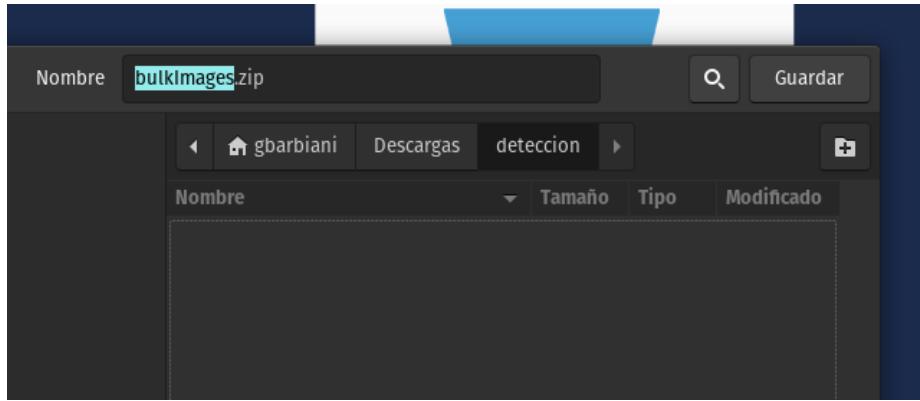


Figura 61: Descarga de archivo con imágenes detectadas - Resultados

### 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 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-AP1-01	RF-AP1-02	RF-AP1-03	RF-AP1-04	RF-AP1-05	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 profundo 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 que es muy útil y puede ser aplicado a diferentes problemas, como es en este 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 éstos 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 publico.

### 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 [4], 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 la métrica 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 mas importante, 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 pagina 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 materia tal en la carrera, la problemática que se planteo 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 planteados.

## Chapter 8

### Apéndice

#### 8. Imágenes Dataset

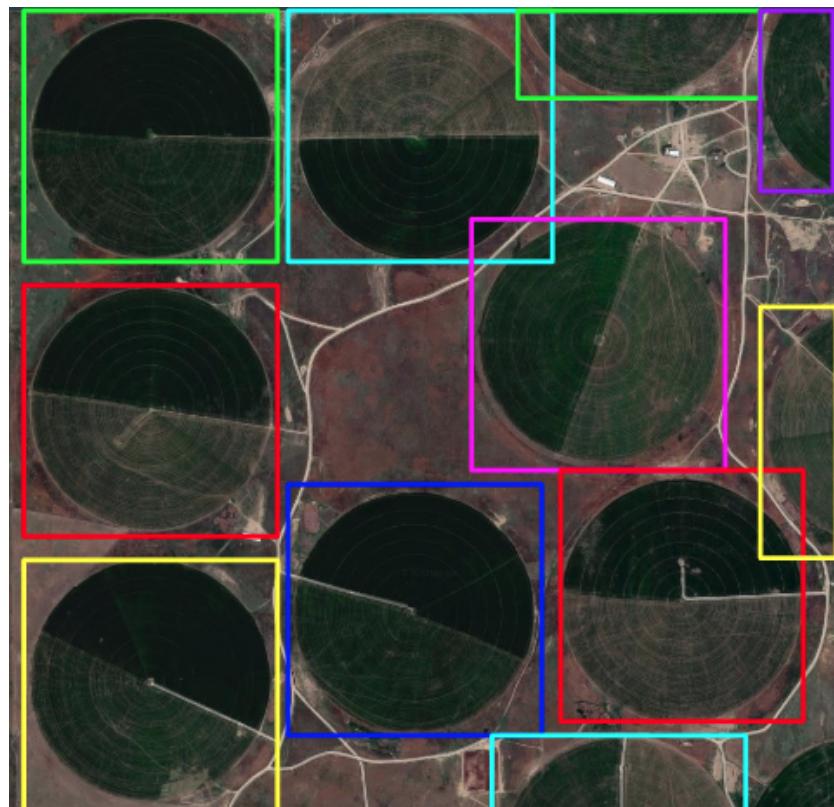


Figura 62: Pivot etiquetado manualmente

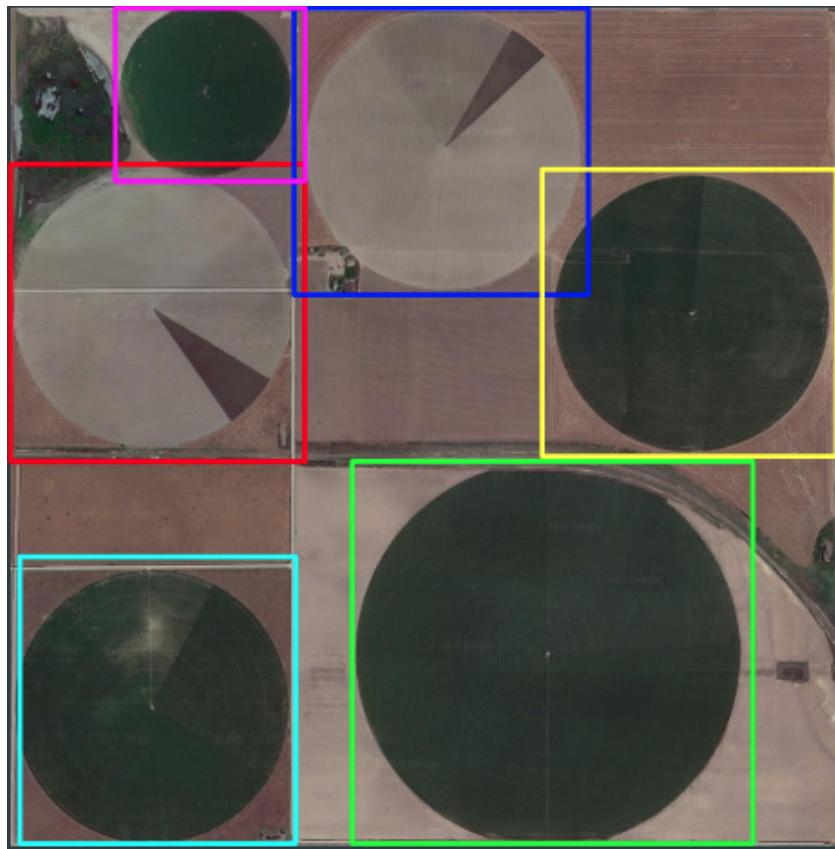


Figura 63: Pivot etiquetado manualmente



Figura 64: Silobolsa etiquetado manualmente



Figura 65: Silobolsa etiquetado manualmente



Figura 66: Silobolsa etiquetado manualmente



Figura 67: Silobolsa etiquetado manualmente



Figura 68: Pivot generado con data augmentation



Figura 69: Pivot generado con data augmentation

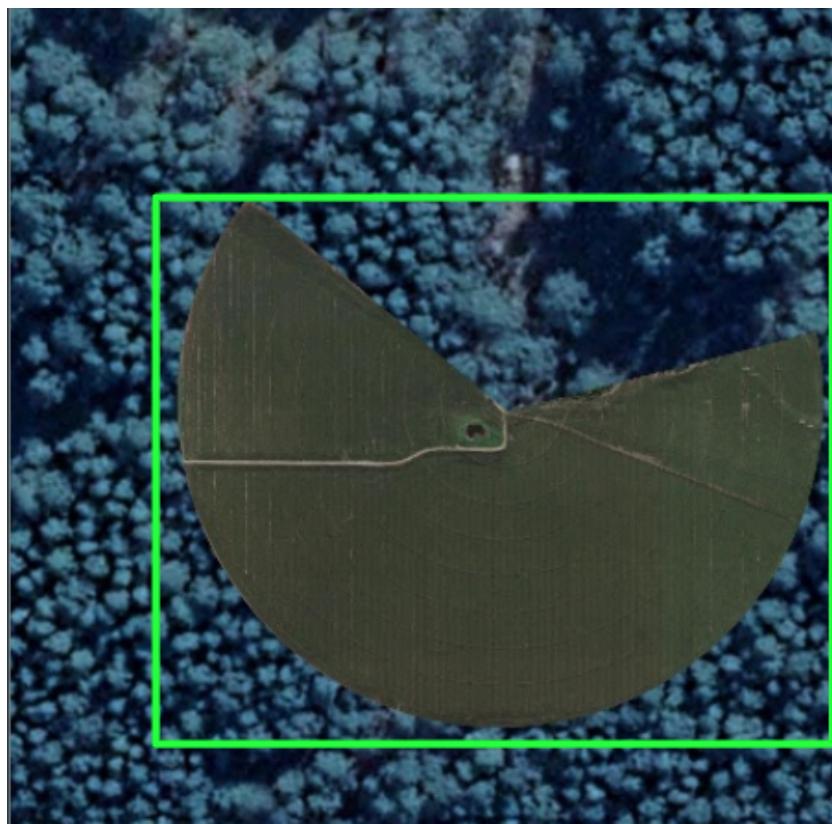


Figura 70: Pivot generado con data augmentation



Figura 71: Silobolsa generado con data augmentation



Figura 72: 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 Silobolsas 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](https://github.com/gianfrancob/detector_pivotes_silobolsas)
2. Inicializar el contenedor Docker corriendo: `docker run ./utils/docker/Dockerfile`
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 pagina abriendo en el navegador el siguiente fichero HTML: `./utils/bootstrap_frontend/index.html`

### 9.4. Ingresar a la pagina

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 pagina 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 74:

## 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 74

## 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" como indicio de q la detección esta en proceso. Como se puede apreciar en la figura 75.

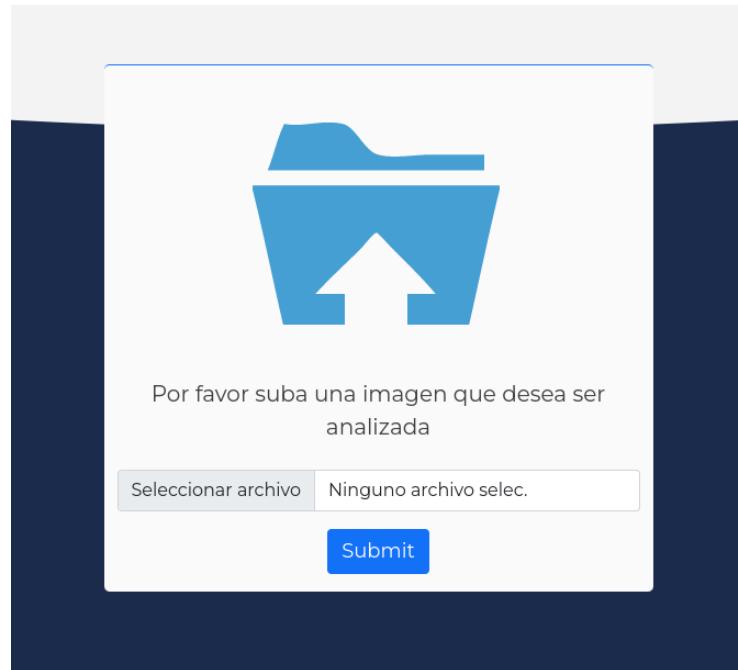


Figura 73: Subir una imagen para analizar

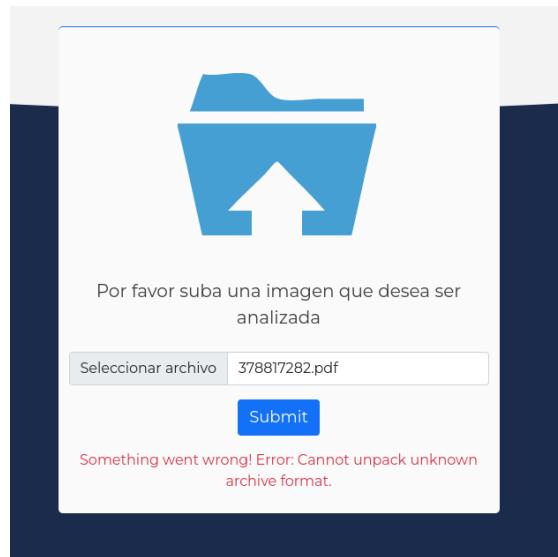


Figura 74: Error al subir una archivo de formato invalido

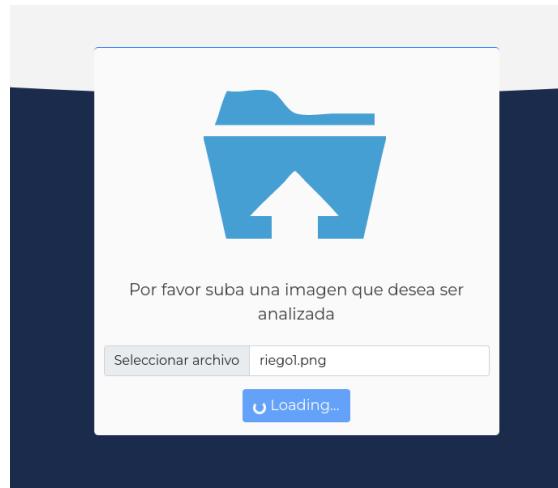


Figura 75: Inicio de la detección - Loading

## 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 76 y se debe seleccionar el archivo del destino como se muestra en la Figura 77

Nombre	Tiempo Total	Tiempo de Inferencia	Tamaño de la Imagen	Pivotes Detectados	Silobalsas Detectadas
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
riegol.png	54.2ms	0.054s	416x640 px	5	0
riegol2.png	54.2ms	0.060s	512x640 px	5	0

Figura 76: Detección de un archivo con imágenes comprimidas

## 9.9. Agradecimientos

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>.

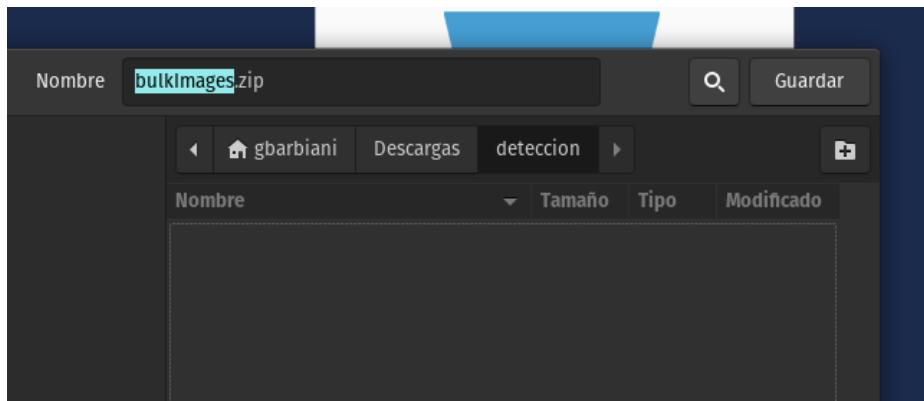


Figura 77: Descarga de archivo con imágenes detectadas - Resultados

## Bibliografía

- [1] 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>, accessed: 2022-01-16
- [2] Bootstrap: The most popular html, css, and js library in the world. <https://getbootstrap.com/>, accessed: 2022-03-25
- [3] Convolutional neural networks - andrew ng - coursera course. <https://www.coursera.org/learn/convolutional-neural-networks>, accessed: 2021-03-25
- [4] Convolutional neural networks dictated by dr. andrew ng. <https://www.coursera.org/lecture/convolutional-neural-networks/object-localization-nEeJM>, accessed: 2022-03-25
- [5] Docker: A complete container solution. <https://www.docker.com/>, accessed: 2022-03-25
- [6] Dockerhub: Container image library. <https://hub.docker.com/>, accessed: 2022-03-25
- [7] Flask: Web development, one drop at a time. <https://flask.palletsprojects.com/en/2.1.x/>, accessed: 2022-03-25
- [8] Intersection over union (iou) for object detection. <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>, accessed: 2021-03-25
- [9] Kaggle: Your machine learning and data science community. <https://www.kaggle.com/>, accessed: 2022-03-25
- [10] Kermap. <https://kermap.com/en/>, accessed: 2022-01-16
- [11] Nabucodonosor: Computadora nabucodonosor (ml). <https://ccad.unc.edu.ar/equipamiento/computadora-nabucodonosor/>, accessed: 2022-03-25
- [12] Neural networks as black box. <https://learnonopencv.com/neural-networks-a-30000-feet-view-for-beginners/>, accessed: 2021-03-25
- [13] Nvidia cuda: A parallel computing platform and programming model. <https://developer.nvidia.com/cuda-zone>, accessed: 2022-03-25
- [14] Qgis: Un sistema de información geográfica libre y de código abierto. <https://www.qgis.org/es/site/>, accessed: 2022-03-25
- [15] Redhat: ¿qué es una api de rest? <https://www.redhat.com/es/topics/api/what-is-a-rest-api>, accessed: 2022-03-25
- [16] Satellogic: Creating a searchable earth. <https://www.satellogic.com/>, accessed: 2022-03-25
- [17] Ssh tunnel: Definition. <https://www.ssh.com/academy/ssh/tunneling>, accessed: 2022-03-25

- [18] Stackoverflow foro. <https://stackoverflow.com/questions>, accessed: 2021-10-09
- [19] Sysml open source project - what is sysml? who created sysml? <https://sysml.org/>, accessed: 2021-12-11
- [20] Tesla: Inteligencia artificial y piloto automático. [https://www.tesla.com/es\\_ES/AI](https://www.tesla.com/es_ES/AI), accessed: 2022-03-25
- [21] Una introducción suave a los conceptos de aprendizaje automático. <https://medium.com/machine-learning-in-practice/a-gentle-introduction-to-machine-learning-concepts-cfe710910eb>, accessed: 2021-03-25
- [22] Understanding and visualizing convolutional neural networks. <https://medium.com/@SeoJaeDuk/archived-post-understanding-and-visualizing-convolutional-neural-networks-d6da3e2851cf>, accessed: 2021-03-25
- [23] Virtualenv: A tool to create isolated python environments. <https://virtualenv.pypa.io/en/latest/>, accessed: 2022-03-25
- [24] Voc2012: Pascal visual object classes. <http://host.robots.ox.ac.uk/pascal/VOC/>, accessed: 2022-03-25
- [25] Yolototrecords - use tensorflow detection api for training and evaluation. <https://github.com/mwindowshz/YoloToTfRecords>, accessed: 2022-03-25
- [26] Yolov5 using the pytorch framework. <https://github.com/ultralytics/yolov5>
- [27] Ballard, D.H., Brown, C.M.: Computer vision (1982)
- [28] Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection (2020). <https://doi.org/10.48550/ARXIV.2004.10934>, <https://arxiv.org/abs/2004.10934>
- [29] Hartigan, J.A.: Clustering Algorithms. John Wiley amp; Sons, Inc., USA, 99th edn. (1975)
- [30] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989). [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), <https://www.sciencedirect.com/science/article/pii/0893608089900208>
- [31] Huang, T.S.: Computer vision: Evolution and promise (1996)
- [32] Lecun, Y.: A theoretical framework for back-propagation (08 2001)
- [33] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection (2016). <https://doi.org/10.48550/ARXIV.1612.03144>, <https://arxiv.org/abs/1612.03144>
- [34] McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (Dec 1943)
- [35] Morris, R.G.: D.O. hebb: The organization of behavior, wiley: New york; 1949. Brain Res Bull **50**(5-6), 437 (Nov 1999)
- [36] Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Dasgupta, S., McAllester, D. (eds.) Proceedings

- of the 30th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 28, pp. 1310–1318. PMLR, Atlanta, Georgia, USA (17–19 Jun 2013), <https://proceedings.mlr.press/v28/pascanu13.html>
- [37] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection (2015). <https://doi.org/10.48550/ARXIV.1506.02640>, <https://arxiv.org/abs/1506.02640>
  - [38] Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger (2016). <https://doi.org/10.48550/ARXIV.1612.08242>, <https://arxiv.org/abs/1612.08242>
  - [39] Redmon, J., Farhadi, A.: Yolov3: An incremental improvement (2018). <https://doi.org/10.48550/ARXIV.1804.02767>, <https://arxiv.org/abs/1804.02767>
  - [40] Riedmiller, M.A., Braun, H.: A direct adaptive method for faster backpropagation learning: the rprop algorithm. IEEE International Conference on Neural Networks pp. 586–591 vol.1 (1993)
  - [41] Sommerville, I.: Software Engineering. Addison-Wesley (2011)
  - [42] Sonka, M., Hlavac, V., Boyle, R.: Image processing, analysis and machine vision (3. ed.). (01 2008)
  - [43] Zoph, B., Cubuk, E.D., Ghiasi, G., Lin, T.Y., Shlens, J., Le, Q.V.: Learning data augmentation strategies for object detection (2019). <https://doi.org/10.48550/ARXIV.1906.11172>, <https://arxiv.org/abs/1906.11172>