UNIVERSIDAD PRIVADA ANTENOR ORREGO

ESCUELA DE POSGRADO

TESIS PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA DE SISTEMAS CON MENCIÓN EN SISTEMAS DE INFORMACIÓN

"Desarrollo de una estrategia inteligente mediante redes neuronales convolucionales profundas para la medición de la calidad del crecimiento de plantines producidos en viveros industriales de la región La Libertad"

Área de investigación:

Sistemas inteligentes

Autor:

Ing. Erick M. Fiestas Sorogastua

Jurado Evaluador:

Presidente: Ms. Calderón Sedano, José Antonio

Secretario: Ms. De La Cruz Rodríguez, Oscar Miguel

Vocal: Ms. Cerna Sánchez, Eduardo Elmer

Asesor:

Dr. Ing. Prado Gardini, Sixto Ricardo

Código Orcid: https://orcid.org/0000-0002-9135-2663

TRUJILLO – PERÚ

2022

Fecha de sustentación: 25/03/2022

Dedicado:

A mi madre.

AGRADECIMIENTOS

A Dios y mi familia por su acompañamiento y el gran soporte que me dan en cada proyecto que inicio. A mi asesor Dr. Ing. Sixto Ricardo Prado Gardini, primero, por confiar en mi y darme la oportunidad de ser parte de este trabajo. Y segundo, por estar presente de una forma profesional en cada etapa del desarrollo de esta investigación.

A los ingenieros Jorge Alva y Paulo Linares con quienes hicimos equipo para llevar adelante el grupo de visión por computadora del LABINM. ¡Gracias por su apoyo, chicos!

Mi agradecimiento al CONCYTEC por el financiamiento otorgado con fondos del FONDECYT y el Banco Mundial al proyecto "Desarrollo e implementación de un sistema robotizado para un control de calidad eficiente y continuo del crecimiento de los plantines en viveros industriales de la región La Libertad-Perú" con número de contrato 074-2018-FONDECYT-BM-IADT-AV, del cual esta tesis forma parte.

RESUMEN

"Desarrollo de una Estrategia Inteligente mediante Redes Neuronales Convolucionales Profundas para la Medición de la Calidad del Crecimiento de Plantines Producidos en Viveros Industriales de la Región La Libertad"

La agricultura en el Perú es una actividad económica fundamental que en los últimos años ha tenido un crecimiento acelerado, lo que ha permitido su industrialización. Un rol fundamental para lograr este crecimiento elevado y sostenido son los viveros industriales que proveen de plantines a los fundos agroindustriales, los que previamente pasan por un control de calidad de crecimiento a fin de asegurar un desarrollo como plantas productoras de frutos con calidad de exportación. El control de calidad lo realizan operarios especializados que verifican, en cada plantin, la altura de los tallos, el número de hojas de la plántula, el área foliar de cada hoja, el desarrollo de las raíces, lo que implica acciones de observación visual, cogida, traslado y dejada del plantin desde el punto de avaluación hacia otro diferente según calidad de crecimiento. Sin embargo, si el recurso humano especializado no está disponible en la cantidad que demanda la producción entonces se generan cuellos de botella y por lo tanto una merma en la productividad. Un caso critico es la alcachofa cuya demanda en el mercado nacional es aproximadamente de 400 millones de plantines al año y es el plantin que por sus características fenológicas exige el mayor esfuerzo por parte de los operarios para dar una correcta valorización de la calidad de su crecimiento. En este trabajo se desarrolla una estrategia inteligente mediante redes neuronales convolucionales profundas para la medición de la calidad del crecimiento de plantines de alcachofa producidos en viveros industriales de la región La Libertad-Perú. Primero, se elabora el dataset para el aprendizaje de la red neuronal convolucional constituido por imágenes de plantines de alacachofa reales e imágenes de plantines virtuales. Segundo, se diseñan y comparan modelos de Machine Learning (PCA, kmeans-verificar) y Deep Learning (VGG16 y Yolov3) para determinar la mejor estrategia de detección y clasificación de plantines en imágenes RGB. Cuarto, se obtiene la correlación de Pierson para verificar la relación entre la predicción de la estrategia previamente determinada y el criterio de clasificación de un vivero industrial de la región, lográndose una correlación del 85%. Finalmente, se desarrolla un sistema de monitoreo, supervisión y gestión en tiempo real que integra dashboard, IA, componentes eléctricos industriales, computación en la nube y robótica. Todo ello ejecutándose en un entorno de red local y en la nube (Google Cloud Platform).

Palabras clave: *Visión por computadora, Deep Learning, Agricultura, IIoT*

ABSTRACT

Agriculture in Peru is growing at an accelerated rate; this has allowed its industrialization. Industrial nurseries must provide seedlings to farmers, a year they must produce more than 400 million seedlings only artichoke. However, the increase in production is becoming more difficult due to the scarcity of specialized labor for quality control of products, within which there is a specific visual analysis procedure to classify seedlings according to their quality. The present thesis work presents the development of an intelligent strategy using deep convolutional neural networks for measuring the quality of the growth of seedlings produced in industrial nurseries in the region of La Libertad. First, the necessary dataset based on images of artichoke seedlings is prepared, including images of real and synthetic seedlings. Second, a set of Machine and Deep Learning models (PCA, kmeans, VGG16 and Yolov3) are proposed and designed in order to be able to perform a comparison of the performances of each of these for the task of detecting and classifying RGB images of the seedlings. Third, the training, testing and validation of the models is carried out. Fourth, the Pierson correlation is obtained to verify the relationship between the prediction of previously trained models and the classification criteria of an industrial nursery in the region. Finally, the model is displayed with a respective monitoring panel (dashboard) that runs on the local network and in the cloud (Google Cloud Platform). The conclusions of the work are shown as a Computer Vision System (SVC) based on Industrial Internet of Thing (IIoT) that integrates AI, industrial components, cloud computing and robotics, achieving a seedling classification capacity that correlates an 85 % with what has been done in industrial nurseries.

Palabras clave: *Visión por computadora, Deep Learning, Agricultura, IIoT*

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO I

INTRODUCCION

"La educación es el arma más poderosa que puedes usar para cambiar el mundo"

Nelson Mandela.

CAPÍTULO I: INTRODUCCION

El presente trabajo se desarrolla en el marco del proyecto N°74-2018-FONDECYT-BM-IADT-AV, titulado "Desarrollo e Implementación de un Sistema Robotizado para un Control de Calidad Eficiente y Continuo del Crecimiento de los Plantines en Viveros Industriales de La Región La Libertad-Perú", que tiene como propósito principal automatizar el control de calidad del crecimiento de plantines de producción industrial y con ello resolver el problema de falta de operarios especializados para la actividad de control previamente mencionado, en especial en periodos de alta demanda del mercado agroindustrial.

Acciones propias del control de calidad del crecimiento de un plantin, son verificar la altura y grosor de los tallos, el número de hojas de la plántula, el desarrollo de las raíces, lo que implica por parte del operario realizar observación visual y trasplante (coger, trasladar y dejar, Fig. 1) del platin desde y hacia diferentes bandejas multi celdas (recipientes estandarizados que contienen a los plantines) según calidad previamente

Fig. 1. Plantín de alcachofa en evaluación manual de su crecimiento. Fuente: Elaboración propia.

calculada. Esta acción de coger, trasladar y dejar, también es conocido como el repique del plantin.

En los viveros industriales de la región de La Libertad, el control de calidad del plantin de alcachofa tiene tres posibles resultados (Fig. 2); buena calidad (listo para entrega a cliente final), calidad media (plantin recuperable) y mala calidad (plantin para consumo no industrial). Y cada plantin se deposita en bandejas que contienen solo plantines de la misma calidad, teniendo como resultante bandejas homogéneas en calidad. Por otro lado, los operarios o jornaleros no hacen u so de instrumentos de medición debido a la exigencia de velocidad del repique, que en promedio es de 3 segundos/plantín para un jornalero con experiencia es por ello que utilizar personal no calificado no solo incide en un deficiente control de calidad (lo que genera en una devolución por parte de las agroindustrias o clientes de aproximadamente del 15% al 18% del producto total) si no que genera cuello de botella en el proceso al ejecutar su trabajo de forma mucho más lenta.

Para el caso de los plantines de alcachofa el control de calidad se basada en el proceso de repique. El repique es un procedimiento en el que se clasifican los plantines (como bueno, regular o malo) según su condición para supervivir en campo, para lo cual se consideran las características morfológicas de la planta (hojas y tallo) como factores principales. El problema radica en que este proceso se realiza por jornadas, por lo cual se requiere de operadores (jornaleros) con habilidad para manipular los plantines, que por su estructura frágil es vulnerable a múltiples factores que afectan su calidad. La falta de jornaleros no solo puede afectar la producción, sino también incurre en gastos elevados de capacitación.

Fig. 2. Clases de plantín según su calidad. Fuente: Elaboración propia.

En Agrolalibertad (2020), se indica que la región La Libertad-Perú es el mayor exportador de alcachofas a nivel nacional, con el 60% de la producción total, lo que ha representado un ingreso de 57 millones de dólares en el año 2019. Un ejemplo claro de esta elevada producción es la empresa Agrogenesis que produce aproximadamente 400 millones de plantines de alcachofa al año y todos ellos pasan por un control de calidad de su crecimiento. En general y de acuerdo a datos de producción de la misma empresa, en una bandeja multi celda de 72 plantines se observa que aproximadamente el 70% de plantines se clasifican como de buena calidad, 20% de regular calidad y 10% de mala calidad (ver Anexo I). Es decir, dado una producción de 400 millones de plantines, entonces aproximadamente 280 millones son de buena calidad, 80 millones son de regular calidad y 40 millones de mala calidad.

El control de calidad no solo determina la calidad del crecimiento del plantin si no que permite aplicar oportunamente estrategias de recuperación para los plantines de regular y hasta de mala calidad minimizando con ello las perdidas de producción. El tiempo promedio que demora un jornalero experimentado para determinar la calidad de un plantin es de 3 segundos/plantín, si se considera una jornada laboral de 40 horas semanales entonces su producción promedio en un mes es aproximadamente de 192 mil plantines. Por lo que si la producción promedio anual en un vivero industrial del nivel de Agrogenesis es de 400 millones de plantines de alcachofas entonces se requiere evaluar 33 millones de plantines/mes. Lo que implica tener aproximadamente 174 jornaleros altamente experimentados durante el proceso de control de calidad del crecimiento del plantin (que aproximadamente se da a los 20 días de sembrado la semilla del mismo).

El inconveniente se presenta cuando no se dispone de la cantidad de jornaleros experimentados que se requiere para satisfacer la demanda del mercado. El aumento de las horas de trabajo por jornalero no es una solución efectiva por el poco impacto que este representa en la producción general y además incrementa la fatiga del trabajador en desmedro de la calidad de su trabajo.

Una alternativa de solución es complementar el trabajo de los jornaleros especializados mediante la automatización del control de calidad del crecimiento de los plantines el que

debe realizar automáticamente las siguientes acciones: 1) extraer las características morfológicas de crecimiento del plantin (visión); 2) calcular la calidad del crecimiento (procesamiento de imagen digital); coger establemente el plantin desde su correspondiente agujero de la bandeja que la contiene (mecanismo automatizado); 3) trasladar establemente el plantin (mecanismo automatizado); 4) dejar establemente el plantin en el agujero de la nueva bandeja, según criterio de calidad (mecanismo automatizado).

En este trabajo se desarrolla una estrategia para implementar automáticamente las dos primeras acciones descritas previamente, mediante técnicas de Inteligencia Artificial (IA) basadas en Redes Neuronales Convolucionales (CNN). Las que han demostrado una alta estabilidad ante ruidos naturales como el cambio de luminosidad del medio ambiente (Shatnawi et. al., 2018, An et. al., 2019) y se ejecutan a alta velocidad pudiendo con ello cumplir con los requerimientos de productividad. Cabe mencionar que en los últimos años se han desarrollado con gran interés diversos algoritmos de inteligencia artificial aplicables a la agricultura, obteniéndose resultados prometedores por lo que se mantiene una activa investigación en esa línea (Mochida et. al., 2018).

Por lo tanto, de acuerdo a lo descrito previamente aquí se plantea la siguiente pregunta de investigación:
¿Cómo una estrategia inteligente basada en redes neuronales convolucionales profundas mide la calidad del crecimiento de plantines de alcachofa en viveros industriales de la región La Libertad?

De acuerdo a lo descrito en la realidad problemática y la pregunta de investigación, se plantea el siguiente objetivo principal:

- Desarrollar una estrategia inteligente mediante redes neuronales convolucionales para la medición de la calidad del crecimiento de plantines producidos en viveros industriales de la Región La Libertad

Y a fin de alcanzar el objetivo general se plantean los siguientes objetivos específicos:

- Implementar el *dataset* de imágenes de plantines de alcachofa de diferentes calidades para entrenar y validar modelos clasificadores de plantines basado en redes neuronales.
- Diseñar diferentes modelos clasificadores de plantines basados en redes neuronales convolucionales para determinar aquel que mejor aprenda desde el *dataset* previamente desarrollado.
- Implementar un sistema de visión por computadora basado en el modelo clasificador de plantines previamente determinado para integrarse a un sistema robotizado manipulador de plantines (coge, transporta y deja).

La hipótesis planteada es:
El desarrollo de una estrategia inteligente mediante redes neuronales convolucionales profundas permitirá medir la calidad del crecimiento de plantines producidos en viveros industriales de la región la libertad.

La justificación de este trabajo se basa en un:

- Aporte de procesos: Se mejora el control de calidad para clasificar plantines según su crecimiento en agroindustrias de la región La Libertad.
- Aporte académico: Se integran diferentes conocimientos de Inteligencia Artificial mediante el método científico para obtener un sistema de visión por computadora con un alto potencial de escalamiento al sector productivo agroindustrial.
- Aporte cultural: Se fomenta la capcitación y formación continua por parte del recurso humano al integrar en los procesos productivos convencionale tecnología de punta.

CAPÍTULO II

MARCO TEÓRICO

"Comienza haciendo lo que es necesario, después lo que es posible
y de repente estarás haciendo lo imposible"
San Agustín.

CAPÍTULO II: MARCO TEÓRICO

2.1. Antecedentes

An, Li, Cui y Yue (2019) en su investigación "*Identification and Classification of Maize Drought Stress Using Deep Convolutional Neural Network*", se identificaron y clasificaron los niveles del estrés por sequedad de plantines de maíz haciendo uso de una red neuronal convolucional profunda (DCNN), la que aplicaron a 1710 imágenes de plantines de maíz. La investigación alcanzó los siguientes resultados: la exactitud y precisión para la identificación y clasificación de estrés por sequedad fue de 98.14% y 95.95%, según el tipo de DCNN que se usó, por lo que se concluyó que en esta labor de identificación y clasificación el método de DCNN es más preciso que los métodos tradicionales de machine learning, ya que a parte de su precisión también permite ahorrar tiempo por tener un tiempo de respuesta más corto.

Bréda (2003) en su investigación "*Ground-based measurements of leaf área index: a review of methods, instruments and current controversies*", revisó todos los métodos publicados hasta ese momento, que miden en una hoja: el índice del área, la evolución temporal y sus características morfológicas. La investigación concluye que existen métodos que se pueden complementar a fin de determinar la relación entre la luminosidad y el índice de área de la hoja. Adicionalmente, indican que el área de la hoja (entre otras características morfológicas de la misma) puede medirse usando fotografías y la transmitancia de la radiación. El principal aporte de este trabajo es haber facilitado el conocimiento de los principales métodos indirectos y directos de medición del índice de área de la hoja.

Lin, Si, Chen y Wu (2016) en su investigación "*A new approach for Measuring Leaf Projected Area for Potted Plant base on Computer Vision*", desarrollaron una técnica mediante la transformación de Hough y un objeto de referencia para medir de forma exacta y rápida el área de una hoja. Aquí usaron como objetos a detectar (en una imagen) plantines de lechuga en macetero. La metodología empleada convierte los valores RGB a escala de grises, para luego binarizarla y obtener una segmentación de los objetos (plantines, macetero) encontrados en la imagen. La investigación logró realizar la separación entre los plantines y el macetero que los contenía haciendo uso de la

transformación de Hough a fin de poder detectar el área de las hojas de los plantines. El principal aporte del trabajo de investigación es la tarea de segmentación. Sin embargo, el problema del traslape entre las hojas de los plantines no fue abordado en este trabajo.

Xiao, Tan, Liu, Yang (2019) en su investigación "*Classification Method of plug seedlings base on transfer learning*", estudiaron la clasificación de plantines de pimentón usando un algoritmo de *Deep learning* convolucional a fin de identificar las características de crecimiento del plantín en un entorno estructurado. Los plantines están ubicados en bandejas multicelda las que a su vez son desplazadas mediante una faja transportadora, hasta la zona de visión artificial (cámara) que consta de una cavidad aislante de luminosidades indeseadas y en la que se realizan tomas fotográficas homogéneas en luminosidad.

El método de clasificación de los plantines (en bandeja) tiene dos fases: primero se identifica a la bandeja que contiene a los plantines, y posteriormente se identifican los plantines contenidos en ella. Un algoritmo de *Deep Learning* convolucional realiza un pre-entrenamiento sobre las fotos de los plantines (que previamente fueron transformados de RGB a escala de grises). Aquí usan la técnica de *Transfer Learning* a fin de evitar la necesidad de una base de datos de numerosa data, y por ende un tiempo de cálculo excesivo. Como resultado relevante presenta una clasificación con 95.50% de precisión y un tiempo de entrenamiento de 6mins y 8s.

2.2. Marco Teórico

El marco teórico abarca tres principales partes. La primera expone lo concerniente a los plantines y su forma de medición de la calidad; la segunda, engloba los principales algoritmos de procesamiento de imágenes digitales basados en inteligencia artificial (IA) que aportan a la investigación; y finalmente, se expone acerca de las tecnologías de la información que hacen posible la integración de los algoritmos de IA con relación al hardware y software.

2.2.1. Medición de la calidad del plantín

El plantín es la segunda fase de crecimiento de una planta (Fig. 3) y fue estudiado desde inicios del siglo XX por los silvicultores (personas que se dedican a reforestar bosques y montañas), quienes observarón que plantínes con determinadas características morfológicas y fisiológicas alcanzaban un desarrollo maduro de buena calidad que permitían reforestar exitosamente los bosques (Grossnickle & MacDonald, 2018). La silvicultura marcó el inicio de la producción de plantines a gran escala (viveros) mediante métodos y conocimientos replicables que permiten una continua mejora del plantin y asegura un correcto desarrollo como planta o arbusto.

2.2.2. La calidad del plantín

En Pérez (2013) y Thompson (1985) se indica que calidad de un plantín se basa en dos características importantes: genética y cuidados en el vivero. Ambas características determinan que un plantín se desarrolle en fundo agrícola como un arbusto generador de frutos de calidad de exportación.

2.2.3. Técnicas de medición de la calidad del plantín

Según la revisión de la bibliografía, se puede establecer dos perspectivas para la medición de la calidad del plantín. Por un lado, en el sector industrial, la calidad está basada en el volumen de raíces desarrolladas en el sustrato, el tamaño de las hojas y, como agrega Thompson (1985), el diámetro del cuello de la raíz. Por otro lado, la cuantificación de la calidad con fines de investigación requiere de

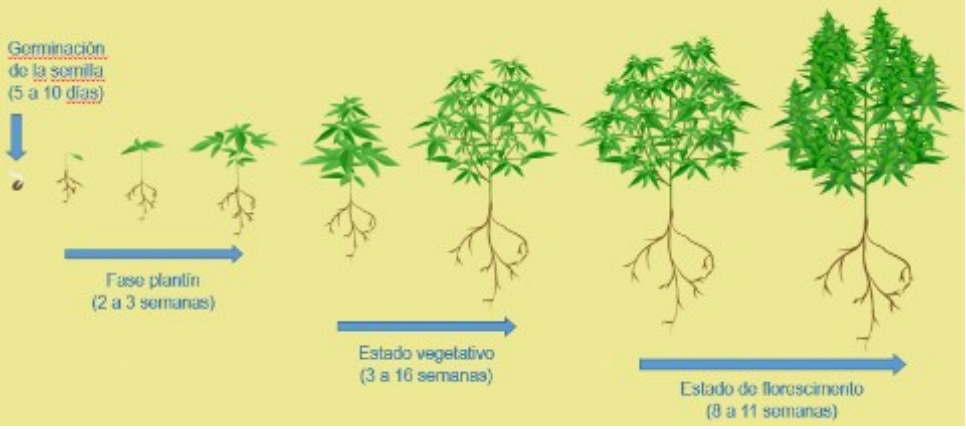


Fig. 3. Fases de crecimiento de la Cannabis Sativa. Fuente: Hennings (2017).

instrumentos de medición con un alto grado de precisión, resolución y repetibilidad como por ejemplo balanzas y verniers optimamente calibrados (Perez, 2013). Los que permiten aplicar eficientemente índices de calidad como el de Dickson (*ICD*):

$$ICD = \frac{Pesosecototal(g)}{\frac{Altura(cm)}{Diámetro(mm)} + \frac{Pesosecoparteaérea(g)}{Pesosecoraíz(g)}} \quad (1)$$

Las variables de la Ec. (1) son caracteriticas morfológicas del plantin, sin embargo estas pueden ser influenciadas por apreciaciones visuales (valores cualitativos) del operario con respecto a las hojas, tallos y raíces (Pérez, 2013). Por otro lado, las características fisiológicas son la concentración de macronutrientes (nitrógeno, fósforo, potasio, calcio, magnesio y azufre), micronutrientes (fierro, manganeso, zinc, cobre y boro), tolerancia a la sequedad, tolerancia al frío, etc. Sin embargo como se indica en Grossnickle & MacDonald, (2018) unas características de fisiológicas de calidad tienen una relación directa con las características morfológicas de calidad. Por lo que, no se puede esperar tener plantines con buena calidad morfológica sin haber recibido un buen tratamiento fisiológico. Esto ultimo ha permitido desarrollar medidas de calidad heurísticas o "practicas" por parte de los operarios especializados de los viveros industriales. Es decir, cuando el plantin de alcachofa tiene 20 dias de sembrado el proceso de control de calidad evalua solo las características morfológicas del mismo, asumiendo que han tenido un buen tratamiento fisiológico.

En esa línea, el trabajo que se describe en este documento desarrolla una estrategia inteligente para extraer solo las caracteriticas morfológicas de los plantines de alcachofa. Y apartir de ahora, todo lo descrito en lo que queda del documento se centrara en obtener lo mejor posible las características morfológicas de los plantines, al menos que se diga lo contrario.

En la Tabla 1 se muestras los valores de los parametros que definen un crecimiento del plantin de buena y regular calidad. Estos valores provienen de la empresa

agroindustrial que mas plantines de alcachofa produce a nivel nacional (Agrogenesis) y que fue cedida por la misma para el mejor desarrollo de este trabajo de investigación. A partir de los datos mostrados en la Tabla 1 se determinan valores promedio de la relación Tallo-hoja (o largo de hoja), la altura de las hojas de alcachofa (desde el inicio del tallo hasta el ápice de la hoja más alta), el ancho de la hoja y finalmente dos valores umbrales (determinados según las experiencias de los operarios especializados) que establecen los límites entre bueno, regular y mala calidad de crecimiento de un plantin según los siguientes intervalos:

- Bueno ≥ 32
- $22 \leq$ Regular < 32
- Malo < 22

Tabla 1. Medición de las dimensiones del plantín.

Descripción	**1era Cat. (Bueno) (mm)**		**2da Cat. (Regular) (mm)**	
(a) Tallo – hoja	49.5 ± 0.5		33.98 ± 0.5	
(b) Ancho de la hoja	23.41 ± 0.5		23.09 ± 0.5	
(c) Altura	53.47 ± 0.5		34.01 ± 0.5	
(d) Ancho de hoja superior	20.19 ± 0.5		13.96 ± 0.5	
Promedio entre a,c,d	41.04		27.32	
Incertidumbre	±9		±5	
Límites	32	50	22	32

2.2.4. Sensores

En Perez-Sanz, Navarro, & Egea-Cortines, (2017) se describen los sensores mas relevantes para medir las características morfológicas de plantines y plantas siendo estas los siguientes: visión mono-RGB, estéreo visión, imágenes multi e hiperespectrales, LIDAR, termografía, etc.

2.2.5. Algoritmos para medir la calidad de crecimiento del plantín

Las imágenes digitales de plantines de alcachofa son una de las alternativas mas utilizadas para, apartir de ellas, determinar las características morfológicas de un plantin. Para lograr aquello de forma eficiente, en los últimos años se han desarrollado diversos algoritmos de procesamiento digital de imágenes con diferentes perspectivas (Fig. 4) que detectan en una imagen regiones que representan un plantin y luego sobre estas regiones aplican estrategias para generar data que se asocia a las características morfológicas reales del plantin.

En Moscoso, Fiestas y Prado (2018) se implementó un sistema robotizado con visión artificial en el que se diseñó un algoritmo para el procesamiento de

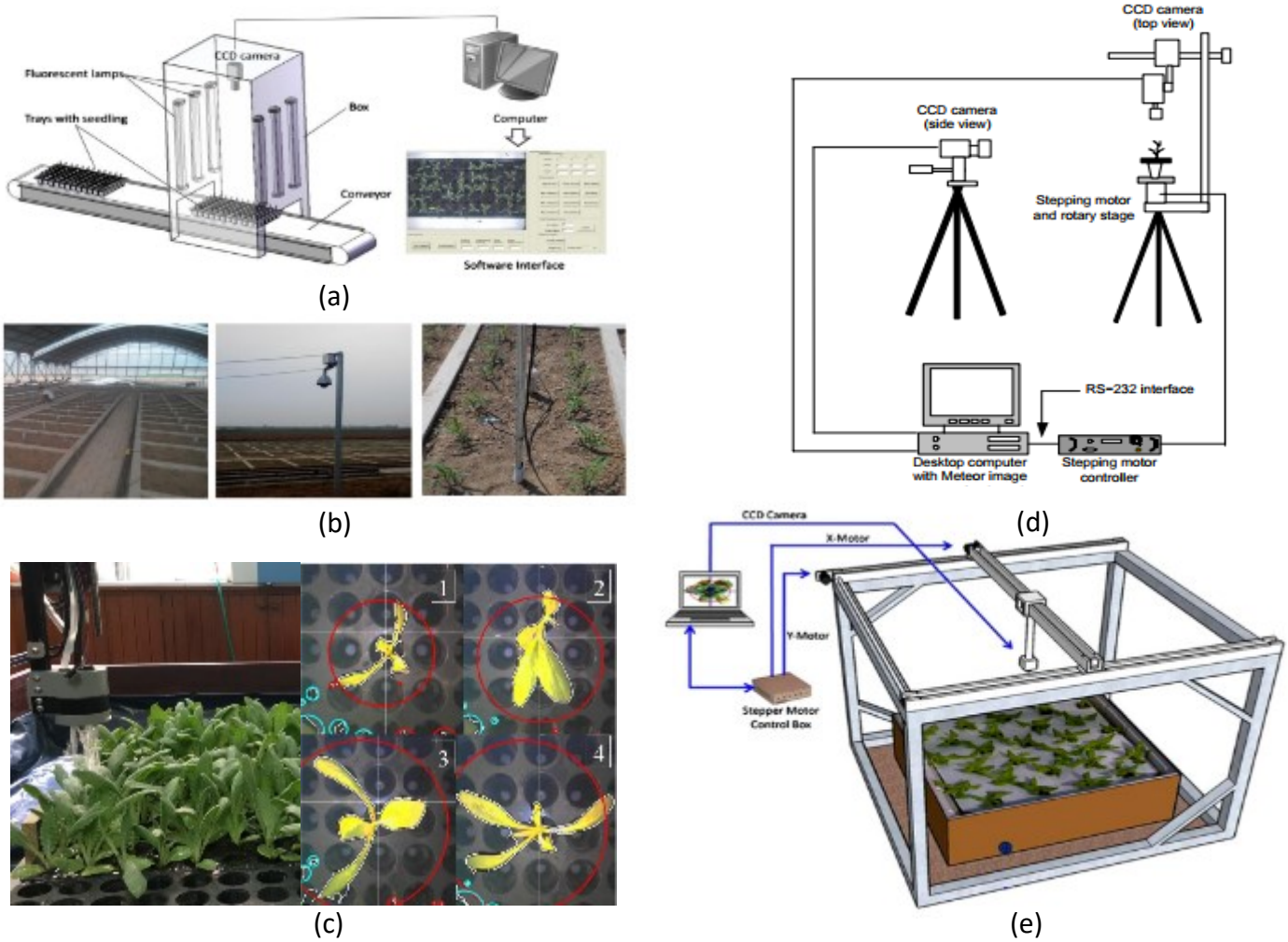


Fig. 4. Diversos sistemas de adquisición de imágen de plantines. (a) Tong, Li, & Jiang (2013), (b) An, Li, Li, Cui, & Yue, (2019), (c) Moscoso-Fiestas-Prado, (2018), (d) Lin, Si, Chen, & Wu, (2016), (e) Story, Kacira, Kubota, Akoglu, & An, (2010).

imágenes mediante la librería OpenCV. Aquí se identifico plantines grandes (óptimos) y pequeños (deficientes), sin embargo, el algoritmo era poco robusto a las variaciones de luminosidad del entorno (ruido luminoso) y no consideraba el traslape entre hojas de plantines vecinos.

Una alternativa robusta y altamente invariante a los ruidos ambientales son los algoritmos de inteligencia artificial basados en el aprendizaje (Goodfellow, Bengio, & Courville (2016). Los que se basan en las siguientes definiciones y conceptos:

2.2.5.1. Machine Learning

El aprendizaje de máquina (*Machine Learning*, ML) según Goodfellow et. al., (2016) quien cita a Mitchell (1997), es: "un programa de computadora que aprende a partir de la experiencia, E, con respecto a alguna clase de tarea, T, y un desempeño medido, P, y se cumple que su desempeño en T, medido por P, mejora con la experiencia E".

Experiencia (E), se basa en la forma de aprender del ML y puede ser principalmente de forma supervisada o no supervisada (Brownlee, 2019):

- Algoritmos de aprendizaje supervisado: requiere de una base de datos donde cada dato se etiqueta con una cualidad asociada a sus características en particular, entonces el algoritmo aprende a asociar la cualidad o etiqueta con las correspondientes caracterisicas. Es decir, aprende que estas etiquetas son la "verdad", y una vez aprendida esta "verdad" lo aplica a otros conjuntos de datos diferente, pero de la misma naturaleza siendo ahora el algoritmo el que asigna la etiqueta correspondiente según características que detecte en el dato en particular evaluado.
- Algoritmos de aprendizaje no supervisado: en este caso no existen conjuntos de datos (de entrenamiento) previamente etiquetados. Para tal experiencia el algoritmo de ML tiene sus funciones matemáticas para establecer sus "propias" etiquetas.

Tarea (T), en el contexto de ML una simple acción relizada por una persona (por ejemplo, el caminar) puede ser una tarea muy compleja de realizar

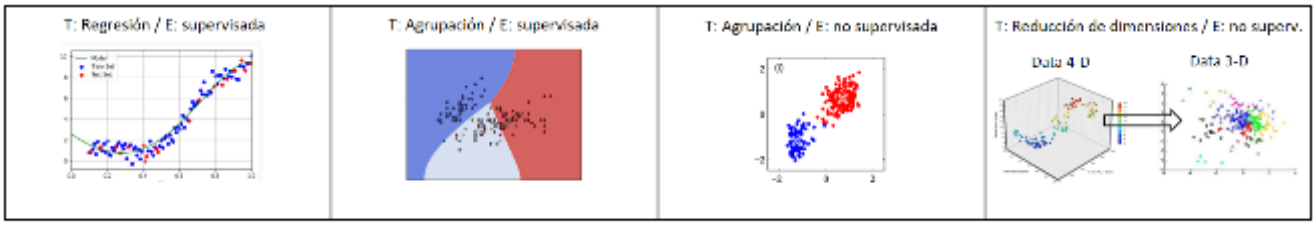


Fig. 5. Principales algoritmos de ML para las tareas (T) de regresión, agrupación y reducción de dimensiones. Cada algoritmo trabaja con una experiencia (E) específica. Gráficas en Python con *Matplotlib*. Fuente: elaboración propia.

mediante la programación. En esa línea, algunas de las tareas de ML pueden ser (Goodfellow et. al., 2016): clasificación (establecer la clase del objeto), regresión (predecir el valor numérico de una variable), detección (identificar la ubicación del objeto en una imagen) y agrupamiento (generar conjuntos de datos de forma no supervisada).

Desempeño (P), evalua la habilidad de un algoritmo de ML a través de una medida cuantitativa que se selecciona según la tarea T a realizar, como por ejemplo el *Mean Squared Error* (MSE), el que idealmente tiene que ser cero al final del entrenamiento (Goodfellow et. al., 2016).

En la Fig. 5 se muestran cuatro de los más relevantes algoritmos de ML (Goodfellow et. al., 2016). El primero, mediante una tarea de regresión predicen fenómenos que pueden ser representados por funciones polinomiales. El segundo y tercero se usan para clasificar grupo de datos y el último permite reducir dimensiones de datos; p. ej., una imagen de 400x400 píxeles a un vector bidimensional.

2.2.5.2. Deep Convolutional Neural Networks

De acuerdo a Goodfellow et. al., (2016), las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son un tipo especial de redes neuronales artificiales (RNA) que se usan para el procesamiento de datos estructurados de diferente forma, p. ej., en una topología tipo red (cuadrícula, como imágenes en escala de grises), o series de tiempo (vectores), o datos de

imágenes a colores (tensores 3D $ancho \times alto \times rgb$, siendo RGB los 3 canales de color: rojo, verde y azul).

Las RNA son funciones anidadas que operan matrices, cada valor de éstas es el producto de la entrada y sus pesos que sumados todos pueden superar el umbral de una función de activación; estas funciones se pueden interpretar como capas, y en el caso de la CNN estas capas son operaciones de convolución (Goodfellow et. al., 2016) que tienen tres características que las hacen eficientes: 1) conexiones dispersas (reducen el número de pesos y de conexiones por entrada), 2) parámetros compartidos (los mismos pesos se aplican a cada una de las entradas), y 3) representaciones equivariantes (aplicación de un conjunto de pesos de forma iterativa en los datos de entrada).

La convolución es una operación matemática definida en la ec. (2) ec. (2) que en DL permite detectar patrones en datos del tipo estructura-dos en redes N-dimensionales (Goodfellow et. al., 2016).

$$s(t) = \int x(a)w\,(t-a)da \tag{2}$$

En la ec. (3) se muestra una operación de convolución discreta con un kernel 2-dimensional que podría usarse, p.ej., en imágenes binarizadas (píxeles blancos y negros).

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3}$$

Un kernel es un conjunto de pesos que filtran los valores de entrada dispuestos en matrices o también llamados tensores (Goodfellow et. al., 2016). La ec. (4) es una convolución (kernel 6D) de una imagen RGB (rojo, verde y azul) representado como un tensor 3D (*ancho* × *alto* × *canal*).

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n,j\%t+1,k\%t+1} \tag{4}$$

En la Fig. 6 se muestra un modelo de CNN con capas de convolución que extraen patrones desde la información de entrada para realizar una clasificación (tarea, T). Aquí se extraen patrones de formas y colores; también cuenta con un maxpooling que asegura robustez ante la variabilidad de la información y a fin de linealizar los valores de entrada se usa una función de activación que puede ser del tipo ReLu, tanh, ELU, entre otros (Kundur et. al., 2019). Adicionalmente, la capa totalmente conectada reduce las matrices de entrada a un solo vector; finalmente, las salidas representan una distribución de probabilidad de 10 clases, la clase con mayor probabilidad representa la predicción final de la clasificación.

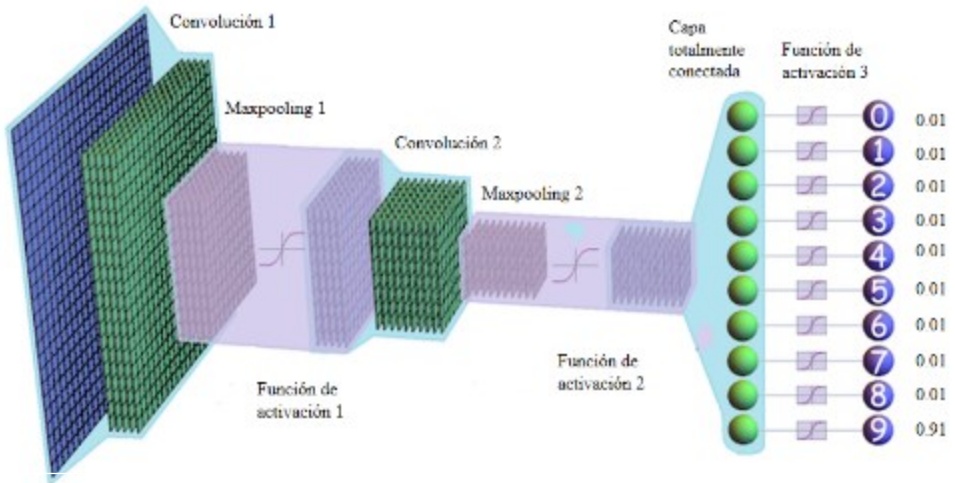


Fig. 6. Arquitectura de red neuronal convolucional en la que se muestran sus principales partes. Fuente: elaboración propia.

2.2.5.3. Modelo CNN basado en Yolov3

Como indica Redmon et. al. (2016) el modelo YOLO (Fig. 7) es una CNN que "razona" globalmente, permitiéndole con ello detectar diferentes objetos a través de un solo procesamiento (*You Only Look Once*). La ec. (5) establece un *score* de confianza de predicción del YOLO comparando sus resultados de clasificar un objeto con una metrica (IoU) de predicciones de detección de objeto (Redmon et. al., 2016).

$$confidence = \Pr(Object) \times IOU_{pred}^{truth} \tag{5}$$

En la Fig. 7 se observa como entrada una convolución con kernel de 7x7x64 que salta dos posiciones entre muestra y muestra, le sigue una serie de 24 capas de convolución, capas maxpooling y capas totalmente conectadas. El modelo

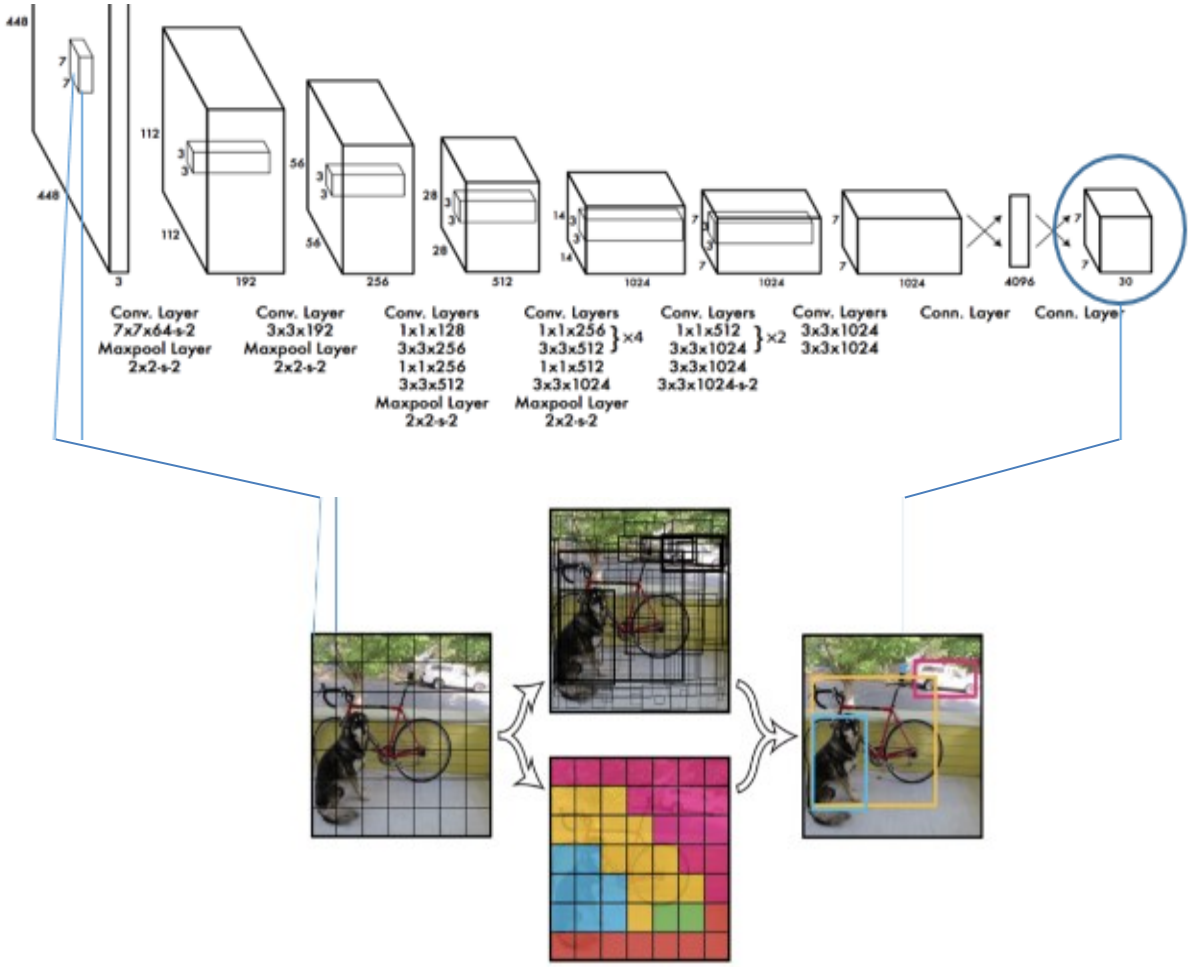


Fig. 7. Arquitectura básica del modelo CNN basado en YOLO. Fuente: Redmon et. al., 2016.

predice múltiples coordenadas (marcos negros) de los objetos detectados, la clase de objeto (con lo cual se genera un mapa coloreado de clases) y el score de confianza (que permite con un umbral discriminar y obtener solo 3 predicciones: perro, bicicleta y auto).

En Redmon-Farhadi (2018) se expone la versión 3 de este modelo, el que usa un mejor extractor de características: *Feature Pyramid Networks* (FPN). Este modelo sigue el paradigma de su antecesor YOLO9000, una versión que repotencia el reconocimiento de objetos a través de marcos ancla pre-establecidos que son los que posteriormente se van a ajustar al tamaño (ancho y alto) de los objetos detectados.

2.2.5.4. Métricas de desempeño en la tarea de clasificación

De acuerdo a Hossin-Sulaiman (2015), las métricas de evaluación juegan un rol fundamental discriminador a fin de poder determinar un clasificador óptimo en el proceso de entrenamiento. Una herramienta altamente eficaz para complementar una metrica de evaluación es la matriz de confusión (MC) que tienen como propósito separar las predicciones correctas de la clase positiva (verdadero positivo, tp) de las predicciones erróneas de la clase positiva (falso positivo, fp), las predicciones erróneas de la clase negativa (falso negativo, fn) y las predicciones correctas de la clase negativa (verdadero negativo, tn). Para ello usan las ec. (6)-(9). En la Tabla 2 se muestra una MC multi-clase (3 clases).

$$tp = tp1, \tag{6}$$

$$tn = tn1 + tn2 + tn3 + tn4, \tag{7}$$

$$fn = fp1 + fp2, \tag{8}$$

$$fp = fn1 + fn2 \tag{9}$$

Tabla 2. Matriz de confusión para clasificaciones multi-clase.

	Clase de predicción			
Clase verdadera		**Clase 1**	**Clase 2**	**Clase 3**
	Clase 1	tp1	fp1	fp2
	Clase 2	fn1	tn1	tn2
	Clase 3	fn2	tn3	tn4

En la Tabla 3 se muestran las métricas complementarias más usadas para las tareas de clasificación multi-clase considerando las ec. (6)–(9)

Tabla 3. Principales métricas para evaluaciones de clasificación (Hossin-Sulaiman, 2015)

Métrica	Fórmula	Foco de evaluación
Exactitud (acc)	$\frac{tp + tn}{tp + fp + tn + fn}$	Mide la razón de las razones correctas considerando el número total de instancias evaluadas.
Razón de error (err)	$\frac{tp + fn}{tp + fp + tn + fn}$	Mide la razón de las predicciones incorrectas considerando el número total de instancias evaluadas.
Recall o sensibilidad (r)	$\frac{tp}{tp + fn}$	Mide la fracción de clases positivas que son correctamente clasificadas.
Especificidad (sp)	$\frac{tn}{tn + fp}$	Mide la fracción de clases negativas que son correctamente clasificadas.
Precisión (p)	$\frac{tp}{tp + fp}$	Mide los patrones positivos que son correctamente predichos considerando el número total de

		predicciones de la clase positiva.
F1-score (f1s)	$\frac{2tp}{2tp + fp + fn}$	Es el promedio armónico de la precisión y el *recall*.

2.2.6. Tecnología de Información y Comunicación

Las tecnologías de la información y comunicación (TIC) son un conjunto de dispositivos que permiten la medición, el procesamiento, el almacenamiento y distribución de la información en sus diferentes versiones y en una gran diversidad de aplicaciones (Niebel, Kopp, & Beerfeltz, 2013). Las TIC mediante la correspondiente integración de hardware y software permiten una automatización mas eficiente de los procesos productivos impulsando con ello un desarrollo mas eficaz (Niebel et al., 2013). P. ej., en la agroindustria (Fig. 8) hay un completo ecosistema de tecnologías que generan, procesan e intercambian información, a fin de tomar las mejores decisiones.

Los algoritmos de inteligencia artificial (IA), como el ML, es desplegado tanto en servidores en la nube (*Cloud Computing*) como *in situ* (*Edge Computing*), solo es

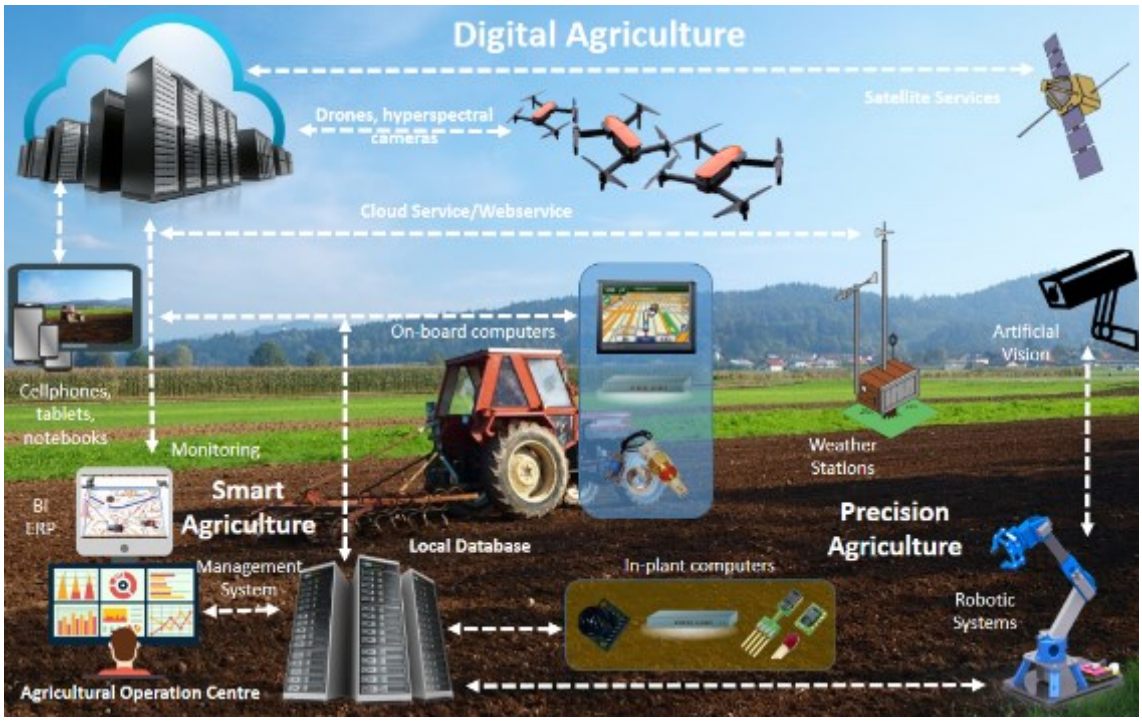


Fig. 8. Principales tecnologías de la información y comunicación de la agricultura digital. Fuente: elaboración propia.

necesario que tenga acceso a los datos, que hoy en día se ha facilitado debido al IoT (*Internet of Things*) y a la mejora del hardware siendo más compacto y potente (Atzori, Iera, & Morabito, 2017).

2.3. Marco Conceptual

- Vivero: es una empresa agrícola que cultiva una diversidad amplia de vegetales, hortalizas y plantas ornamentales para producir plantines.
- Plantín: segunda fase de crecimiento de la planta, precedida por la germinación de la semilla y sucedida por el estado vegetativo de la planta.
- Calidad del plantín: está basado en los valores de la Tabla 1 obtenidos en la visita oficial realizada al viviero Agrogénesis.
- Inteligente: algoritmo que mejora su desempeño en una tarea específica luego de una experiencia con la data de entrenamiento.
- Estrategia inteligente: procedimiento automático basado en un algoritmo inteligente con altas capacidades de soportar el ruido.
- Inteligencia Artificial: conjunto de algoritmos matemáticos que imitan las formas de razonar de criaturas biológicas como animales o seres humanos.
- Deep Learning: algoritmo en el campo de la inteligencia artificial que es capaz de poder autoorganizarse producto de la experiencia sin ser susceptibles al ruido y cambio impredecibles en entornos caóticos.
- Características morfológicas: en el contexto de los plantines es un conjunto de características fundamentadas en el área del conjunto de hojas, el diámetro del cuello de la raíz, el color, el largo del tallo, entre otros.
- Características fisiológicas: en el contexto de los plantines es un conjunto de características fundamentas en la deficiencia de nutrientes, cantidad de clorofila, resistencia al frío, a la sequedad, etc.
- Sensor: dispositivo que permite transformar una señal física específica (intensidad luminosa, presión, movimiento, etc.) a una eléctrica.
- RGB: formato de imagen que combina los colores Rojo, Verde y Azúl.
- Estéreo visión: sistema de visión basado en más de una cámara que combina tomas desde diferente perspectiva.

- Internet de las Cosas (IoT): tecnología masiva de comunicación entre dispositivos dotas de una diversa gama de sensores que envían información a la internet.
- Cloud Computing: paradigma de computación basada en correr los procesos de procesamiento en la nube.
- Programación: elaboración de un programa en el campo de la informática haciendo uso de procesos de codificación y decodificación de órdenes que dan sentido a su funcionamiento.

CAPÍTULO III

METODOLOGIA

"La verdadera sabiduría está en reconocer la propia ignorancia"

Sócrates.

CAPÍTULO III: METODOLOGIA

En este capítulo se describen las técnicas, métodos e instrumentos utilizados en el desarrollo de la presente investigación.

3.1. Población

Un total de 619 plantines de alcachofa obtenidos de 10 bandejas (no se consideraron las celdas de las bandejas sin plantines que no crecieron o que se destruyeron durante su manipulación).

3.2. Muestra

El tipo de muestreo es: Muestreo aleatorio simple. Se consideró la cantidad total de la población de plantines.

3.3. Unidad de Analisis

El plantín de alcachofa producido en los viveros industriales.

3.4. Operacionalización de variables

- Variable independiente 1 (VI1): la estrategia inteligente mediante redes neuronales convolucionales profundas.
- Variable dependiente 1 (VD1): medición la calidad del crecimiento de plantines de alcachofa en viveros industriales de la región La Libertad.

Tabla 4. Definición de la variable dependiente y sus indicadores.

Variable dependiente: medición la calidad del crecimiento de plantines de alcachofa en viveros industriales de la región La Libertad.			
Dimensiones	Indicadores	Unidad de medida	Instrumento de Investigación
Correlación entre la clasificación de plantines según su calidad basado en	Coeficiente de correlación de Pierson.	Valor porcentual	Gráficas de regresión lineal de la correlación

el estándar de los viveros industriales y el resultado de la predicción del modelo CNN			

Tabla 5. Definición de la variable independiente y sus indicadores.

Variable independiente: estrategia inteligente basada en neuronales convolucionales profundas con el propósito de procesar imágenes de plantines a fin de predecir su calidad			
Dimensiones	Indicadores	Unidad de medida	Instrumento de Investigación
Base de datos de imágenes de plantines de alcachofa	Imágenes de plantines de alcachofa reales	Número de imágenes de plantines	Cámara RGB, y programa de captura de imágenes de plantines reales y sintéticos
	Imágenes de plantines de alcachofa sintéticos		
Diseño de la arquitectura de la CNN para la predicción de la calidad del plantín	Capas de la red neuronal convolucional	Número de capas y parámetros	Tabla comparativa del número de capas y parámetros de los diseños propuestos
	Parámetros para el funcionamiento de la red neuronal		
Entrenamiento del modelo de predicción de calidad del plantín basado en la arquitectura CNN.	Intersección sobre la unión (IoU)	Valor porcentual	Programa desarrollado en Python para el cálculo de los indicadores y tabla de imágenes de plantines reales
	Exactitud		
	F1-score (promedio entre la		

	sensibilidad y la precisión)		clasficados según el criterio del vivero industrial Agrogénesis.
Despliegue del modelo en un servidor integrado con *dashboards* que corren en la red local y en la nube.	Funcionalidades de los *dashboards*	Número de funcionalidades	Explorador web para llamar a los *dashboards*.

3.5. Técnicas e Instrumentos de Recolección de datos

Tabla 6. Técnicas e instrumentos de recolección de datos de las variables.

Indicador	**Técnica**	**Forma de aplicación**	**Forma de obtención**
Coeficiente de correlación de Pierson.	Análisis de datos	Personal	Gráficas de correlación entre las diferentes arquitecturas de CNN haciendo uso de un programa hecho en python.
Imágenes de plantines de alcachofa reales	Observación	Personal	Y se obtiene también un registro de imágenes reales de plantines obtenidas en laboratorio.

Imágenes de plantines de alcachofa sintéticos	Observación	Personal	Se obtiene un registro basado en plantines sintéticos (buenos, regulares y malos) que se producen a través de la renderización de un código escrito en C+L el cual está basado en los Sistemas-L libres de contexto y el lenguaje de programación C
Capas de la red neuronal convolucional	Observación	Personal	Por medio de la bibliografía se conoce la cantidad de capas con las que cuenta cada modelo
Operaciones ariméticas	Observación	Personal	Por medio de la bibliografía se conoce la cantidad de operaciones ariméticas realizadas por cada modelo
Intersección sobre la unión (IoU)	Aplicación de fórmula	Personal	Intersección de las imágenes dividido por la unión de las imágenes
Exactitud	Aplicación de fórmula	Personal	$\frac{tp + tn}{tp + fp + tn + fn}$ Donde:

			tp: positivo verdadero, tn: negativo verdadero, fp: falso positivo, fn: falso negativo
F1-score (promedio entre la sensibilidad y la precisión)	Aplicación de fórmula	Personal	$\frac{2tp}{2tp + fp + fn}$ Donde: tp: positivo verdadero, tn: negativo verdadero, fp: falso positivo, fn: falso negativo

3.6. Procedimiento Metodológico

El desarrollo de la investigación tiene las siguientes fases: elaboración del dataset de imágenes, identificación de los modelos CNN para el reconocimiento de imágenes, entrenamiento de los modelos CNN previamente identificados, correlación de predicciones y estimaciones, y finalmente la puesta en marcha del modelo para la clasificación de plantines.

3.6.1. Elaboración del dataset de imágenes de plantines de alcachofa

Aquí el *dataset* se construye mediante dos formas de generar imágenes: el primero genera imágenes de plantines de alcachofa virtuales 3D desarrollados en un entorno de Sistemas L (Ubbens et. al., 2018) y el segundo, genera imágenes de plantines reales capturadas por el sistema de visión artificial del robot desarrollado en el laboratorio LABINM – Robótica y Automatización (UPAO). Luego cada imagen generada es clasificada y etiquetada a fin de tener el *dataset* listo para iniciar el entrenamiento de los modelos predictivos clasificadores previamente identificados (ML y DL).

3.6.1.1. Modelamiento y síntesis de plantines de alcachofa virtuales

En Ubbens et. al. (2018) se indica que existen muy pocos *datasets* de imágenes de plantas (plantines) disponibles en Internet con las caracteriticas necesarias (calidad y cantidad) para entrenar algoritmos de ML y DL. Ademas mencionan los altos costos que implica construir un *datasets* de imágenes de plantas con determinadas morfologías (todo el servicio agrícola y de tomas de imágenes, que incluye recurso humano especializado, materiales y equipos en ambos casos). Una alternativa ante esta realidad es sintetizar plantas artificiales en entornos virtuales 3D que en los últimos años ha tenido un interesante desarrollo (Turgut et. al., 2020). Una de las estrategias mas utilizadas para tal fin, es el Sistema L que según Prusinkiewicz (2004) es una reescritura de cadena de caracteres que permite emular el crecimiento topologico de las plantas (Aristid Lindenmayeres, 1968).

En este trabajo se diseña e implementa una técnica para generar imágenes de plantines virtuales (un plantin o varios en una sola imagen) en un entorno de Sistema L (específicamente el Sistema-DOL (S-DOL) que usa el lenguaje de programación L+C (integra sintaxis de los sistemas L y el lenguaje C). Esta técnica tiene las siguientes tres partes: modelamiento matemático del plantín de alcachofa, síntesis del plantin virtual y generación de imágenes de plantines agrupados.

En la ec. (9) se define formalmente la cadena del S-DOL (Prusinkiewicz, 2004) para el plantín de alcachofa.

$$G = \langle V, \omega, P \rangle, V^*, V^+ \tag{9}$$

donde V representa el alfabeto con el que se codifica el modelo del plantín; $\omega \in V^+$ es el axioma, una palabra no vacía de la cual parte la reescritura de las cadenas; P es el conjunto de producciones (entendidas como iteraciones) que permiten la reescritura de cadenas necesarias que forman el código final del plantín; V^* es el conjunto de todas las palabras en V; V^+ es el conjunto de todas las palabras no vacías en V.

Una vez definido G se establece el mecanismo que lo interpreta y apartir de ahí se ejecutan acciones. Para ello se usa la técnica denominada "Interpretación de Tortuga" (Prusinkiewicz, 2004), que grafica el rastro que deja un ente ficticio móvil (turtuga), según su posición y orientación relativa.

La orientación de la tortuga se establece por medio de tres vectores $\overrightarrow{H}, \vec{L}\ y\ \vec{U}$ que tienen origen común solidario a la tortuga y son ortogonales entre sí, tal que:

$$\vec{H} \times \vec{L} = \vec{U} \tag{10}$$

$$\left[\vec{H}'\ \vec{L}'\ \vec{U}'\right] = \left[\vec{H}\ \vec{L}\ \vec{U}\right]R \tag{11}$$

Donde la ec. (11) define el giro relativo de la tortuga siendo R el conjunto de rotaciones de la tortuga considerando post-multiplicación de matrices (Prusinkiewicz, 2004). Las ec. (12) – (14) determinn las rotaciones con respecto a cada vector $\overrightarrow{H}, \vec{L}\ y\ \vec{U}$:

$$R_U(\alpha) = \begin{bmatrix} cos\alpha & sen\alpha & 0 \\ -sen\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$R_L(\alpha) = \begin{bmatrix} cos\alpha & 0 & -sen\alpha \\ 0 & 1 & 0 \\ sen\alpha & 0 & cos\alpha \end{bmatrix} \tag{13}$$

$$R_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sen\alpha \\ 0 & sen\alpha & cos\alpha \end{bmatrix} \tag{14}$$

La traslación relativa se ejecuta una vez orientada la tortuga, por lo que no existen relaciones con las posiciones y orientaciones anteriores. Con las coordenadas relativas importa solo registrar la trayectoria con la cual la tortuga se mueve a través del espacio cartesiano 3D. La instrucción del movimiento de la tortuga se establece en la ec. (9).

Se definen los elementos de la estructura del S-DOL diseñado para la producción de plantines de alcachofa, el cual se presenta en el diagrama de flujo mostrado en la Fig. 9.

- **Definición de constantes:** son parámetros que controlan la forma de crecimiento de la planta.

Fig. 9. Diagrama de flujo general del sistema OL-S. Fuente: elaboración propia.

- **Número de producciones:** cada producción es una reescritura de cadena, lo cual se traduce como "el paso" del desarrollo de la planta (crecimiento discreto).
- **Axioma:** es la palabra inicial para la primera producción; figurativamente, en el contexto de esta tesis, se traduce como "la semilla".
- **Reglas de producción:** definen las reglas de la reescritura de cadena de caracteres a medida que se van dando las producciones.
- **Decomposición:** permite establecer la topología de la planta, estableciendo la dirección y ubicación de los objetos (hojas, frutos, flores, etc.) de la planta.
- **Homomorfismo:** establece la forma de los objetos según la topología de la planta.

El alfabeto V (ec. 9) son símbolos que tienen que ser "interpretados por la tortuga". En la Tabla 7 se muestran los principales símbolos de V.

Tabla 7. Comandos de posicionamiento y orientación para la tortuga.

Comando	Acción
#	Incrementa el ancho de la línea de la trayectoria que va describiendo la tortuga, en mm.
!	Disminuye el ancho de la línea de la trayectoria que va describiendo la tortuga, en mm.

[ ]	Inicia una nueva rama, lo que esté entre corchetes serán las instrucciones donde se incluirán otros elementos como hojas, otras ramas secundarias, o flores, frutas, etc.
( )	Lo que está entre paréntesis representa el valor a incrementar o disminuir con respecto a un comando en específico. P. ej.: #(20).
/	Rota alrededor del eje $\vec{H}$ un ángulo en grados positivo si gira en sentido antihorario, de ser antihorario el ángulo es negativo.
&	Rota alrededor del eje $\vec{L}$ un ángulo en grados positivo si gira en sentido antihorario, de ser antihorario el ángulo es negativo.
−	Rota en sentido antihorario alrededor del eje $\vec{U}$ un ángulo en grados.
;	Aumenta una cantidad específica en el índice del vector de colores del programa L-Studio si el número es positivo, de ser negativo disminuye el índice del vector.
F	Traslada a la tortuga hacia adelante dibujando una línea como rastro.
~	Permite establecer el ancho, largo y altura de una superficie como la de una hoja (llama a un archivo de superficie generado en el L-Studio, ver Fig. 10).
&&	Operador lógico AND.

La cadena de reescritura G del S-DOL, se codifica usando lenguaje L+C, a travez de la librería CPFG v. 4.0 (Mech, 2016) mediante el software LStudio para Windows 10 OS. Y en este trabajo el modelo del plantín de alcachofa en L-Studio requiere de los siguientes archivos (ver CD adjunto, archivo “Plantin–Sistema L”):

- **.func:** contiene las funciones de crecimiento variable del plantín en sus diferentes ciclos de vida.
- **.s:** guarda las superficies de crecimiento de las diferentes partes de la planta, aquellas que presentan un área mayor a un umbral, por ejemplo cotiledones y las hojas del plantín (Fig. 10).
- **.fset:** muestra el conjunto de todas las funciones de crecimiento del plantín, es decir, el contenido de cada archivo .func.
- **.pnl:** guarda la información de los botones creados en LStudio; botones usados para la modificación de parámetros de crecimiento del plantín en tiempo de ejecución. En este proyecto no se hace uso de esta funcionalidad.

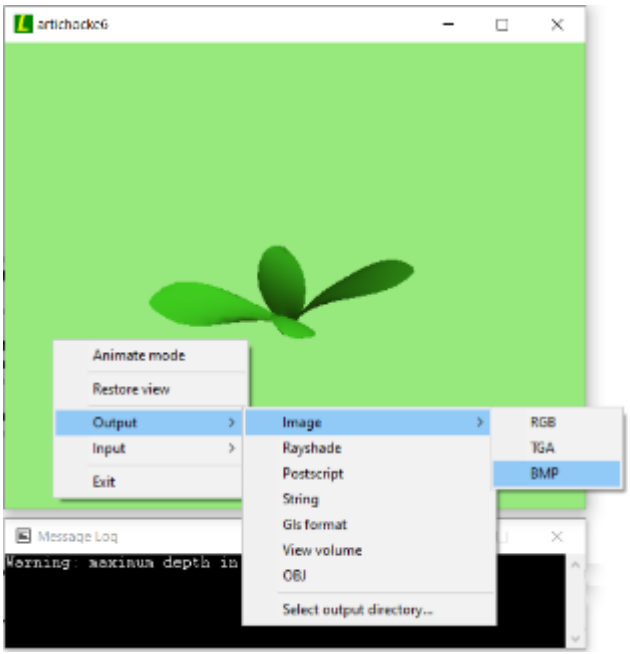


Fig. 10. Exportación de una imagen (.bmp) de la síntesis de un plantín virtual 3D de alcachofa en el software LStudio codificado en lenguaje L+C. Fuente: elaboración propia.

- **.mat:** contiene los datos de colores del modelo de plantín a lo largo de su crecimiento.
- **.l:** guarda el modelo codificado en lenguaje L+C del plantin virtual.
- **.v:** configura el entorno virtual 3D, puntos de vista de visualización, listado de superficies y funciones relacionados a los nombres de los archivos .s y .func, respectivamente.

Al ejecutar el modelo virtul 3D del plantin (en el entorno LStudio) se puede modificar la orientación, la perspectiva, realizar animaciones de crecimiento y exportar en diferentes formatos (en este trabajo se exporta en .bmp, ver Fig. 10).

Las tomas o capturas de imágenes de los plantines virtuales se hacen desde una perspectiva que emule a una cámara cuya dirección de enfoque es perpendicular a la superficie soporte del plantin de alcahofa, tal como sucede con la cámara digital acoplada al sistema robotico desarrollado en el LABINM (UPAO).

Para exportar las imágenes generadas virtualmente se pulsa clic derecho *output>Image>BMP*. El entorno LStudio tiene una ventana de comandos para exportar

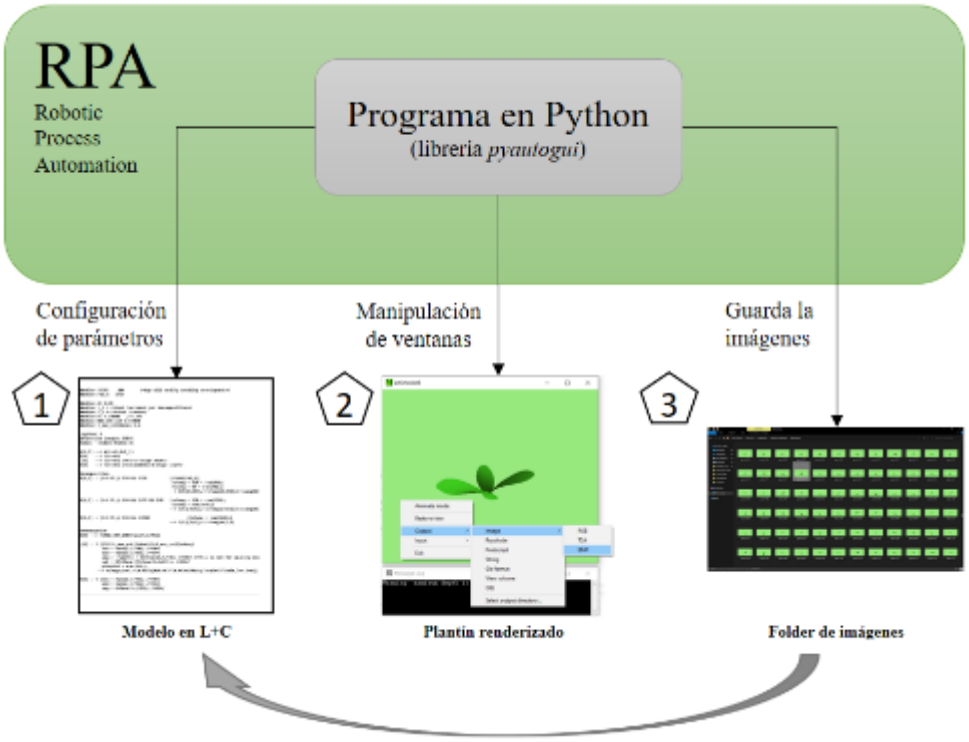


Fig. 11. RPA para la síntesis de plantines controlando la interfaz gráfica de usuario de forma automatizada. Se listan los pasos realizado por el programa en lenguaje Python. Fuente: elaboración propia.

desde código, lo que pemite automatizar la exportación de imagenes. Este aspecto es relevante debido a que se requieren miles de imágenes de plantines diferentes entre sí. Sin embargo, la versión para Windows OS de LStudio tiene un bug que no permite exportar a través de código (cabe mencionar que la versión de MacOS no tiene este bug por lo que si puede exportar desde código).

Una alternativa para automatizar la exportación de imágenes desde Windows OS es desarrollar un *Robotic Process Automation* (RPA) que es una estrategia para automatizar tareas manuales realizadas por un usuario en un computador (Hoffman et. al., 2020). Es decir, el RPA toma el control del teclado y de los movimientos del *mouse* sobre la interfaz gráfica de usuario a fin de exportar automáticamente las imágenes generadas. La implementación del RPA se realiza en Python según los siguientes tres pasos:

- 1: modificar los valores de las definiciones STEP y rnd_n, en el correspondiente programa C+L (presionando las teclas *Ctrl+G*) para obtener la versión renderizada del plantín virtual.
- 2: exportar las imágenes del plantín virtual mediante *output>Image>BMP*.

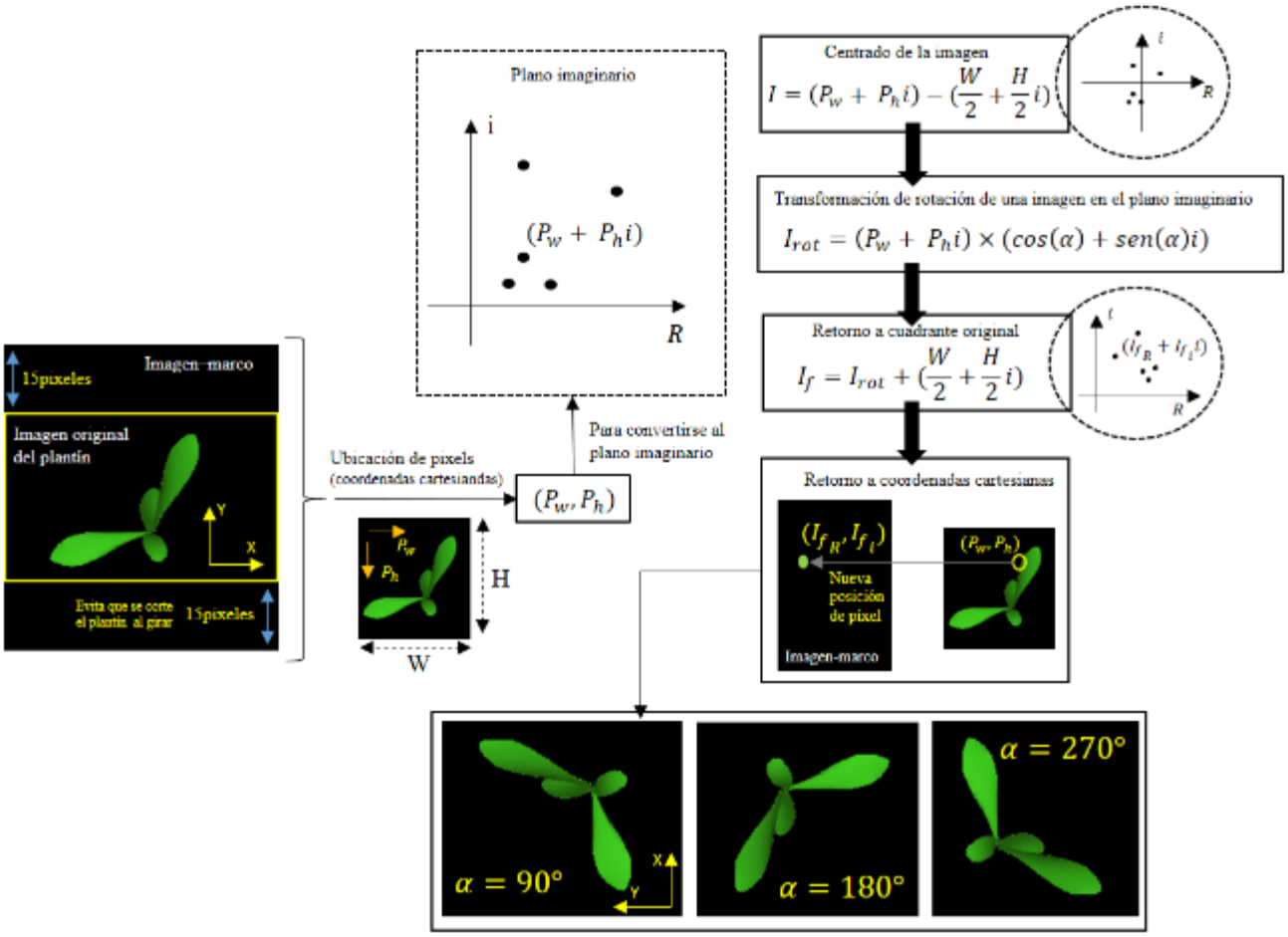


Fig. 12. Diagrama general para la manipulación de imágenes de plantines: traslación y rotación, convirtiendo las coordenadas de los pixels de la imagen a números complejos. Fuente: elaboración propia.

- 3: convertir el archivo de imagen del plantín a un formato .png (menor tamaño de archivo) y guardarlo en una carpeta específica.

Una vez ejecutado los tres pasos previamente descritos se generan otro tipo de imágenes, las que se denominan *plantines sintéticos agrupados*, que mediante una secuencia de traslaciones y reorientaciones de una imagen de plantin inicial se generan otras que se agrupan en un solo marco de. Para ello se traslada a la imagen inicial a un plano imaginario (Klein, 2013) que permite una mejor eficiencia en tiempo de calculo (comparándolo por ejemplo con las matrices rotacionales o matrices homogéneas) al ejecutar traslaciones y rotaciones (recuerde que pueden ser miles de imágenes de plantines a procesar).

En la Fig. 12 se describe el procedimiento para trasladar y rotar las imagenes de los plantines a un plano complejo. La imagen original necesita de un marco o fondo ("imagen-marco") a fin de evitar perder parte de la hoja del plantín al girar, considerando que el tamaño del plantín es casi del tamaño de la imagen, al girar la imagen del plantin el extremo de su hoja puede quedar fuera de la imagen. Posteriormente a cada coordenada de píxel (P_w, P_h) se le calcula su equivalente complejo $(P_w + P_h i)$, en el plano imaginario: se traslada la imagen del plantín al origen del plano restando la mitad del ancho y largo del plantín (buscando que el giro de la imagen sea con respecto a su centro); se realiza la multiplicación para generar la traslación de las coordenadas de los píxeles (efecto de rotación), obteniéndose, una vez retornada la imagen al primer cuadrante, la coordenadas finales (I_{f_R}, I_{f_i}) de los píxeles ya rotados. En estas coordenadas de píxeles se guardan los valores RGB de los píxeles en las coordendas originales de la imagen del plantín. De esta forma se obtienen tres diferentes rotaciones (según valores del angulo α, ver Fig. 12) los que asu vez definen tres imágenes de plantines con localizaciones diferentes contenidas en un mismo marco-imagen (*plantines sintéticos agrupados*). Este procedimeinto es de gran utilidad para construir un *dataset* con un gran número de imágenes de plantines de alcachofa virtuales. Adicionalmente se puede automatizar el etiquetado (cualidad de crecimiento del plantin virtual) tanto de plantines virtuales individuales como de agrupados.

3.6.1.2. Captura de imágenes de plantines de alcachofa reales

Para construir la parte del *dataset* compuesta por imágenes de plantines de alcachofa reales, primero se producen los plantines de alcachofa en un vivero industrial y después de 21 días de sembrado se los transporta al laboratorio a fin de que el sistema de visión artificial, acopaldo al robot manipulador de plantines (LABINM), genere las imágenes de plantines reales.

Mediante una interfaz gráfica de usuario (GUI, previamente desarrollada por el equipo investigador) se procede a la toma de imágenes según el siguiente procedimiento:

- Indicar donde se guardarán las imágenes generadas (clic en el botón *Select folder*).
- Capturar la imagen (clic en el botón *Capture*). La imagen generada se puede borrar pulsando el botón *Clear* y se vuelve a repetir el proceso de captura.

- Seleccionar la calidad de los plantines mediante los *comboboxes* de la GUI debajo de la etiqueta *Quality*: A, B o C.
- Generar la data para el *dataset* (clic en el botón *Generate Sample*).

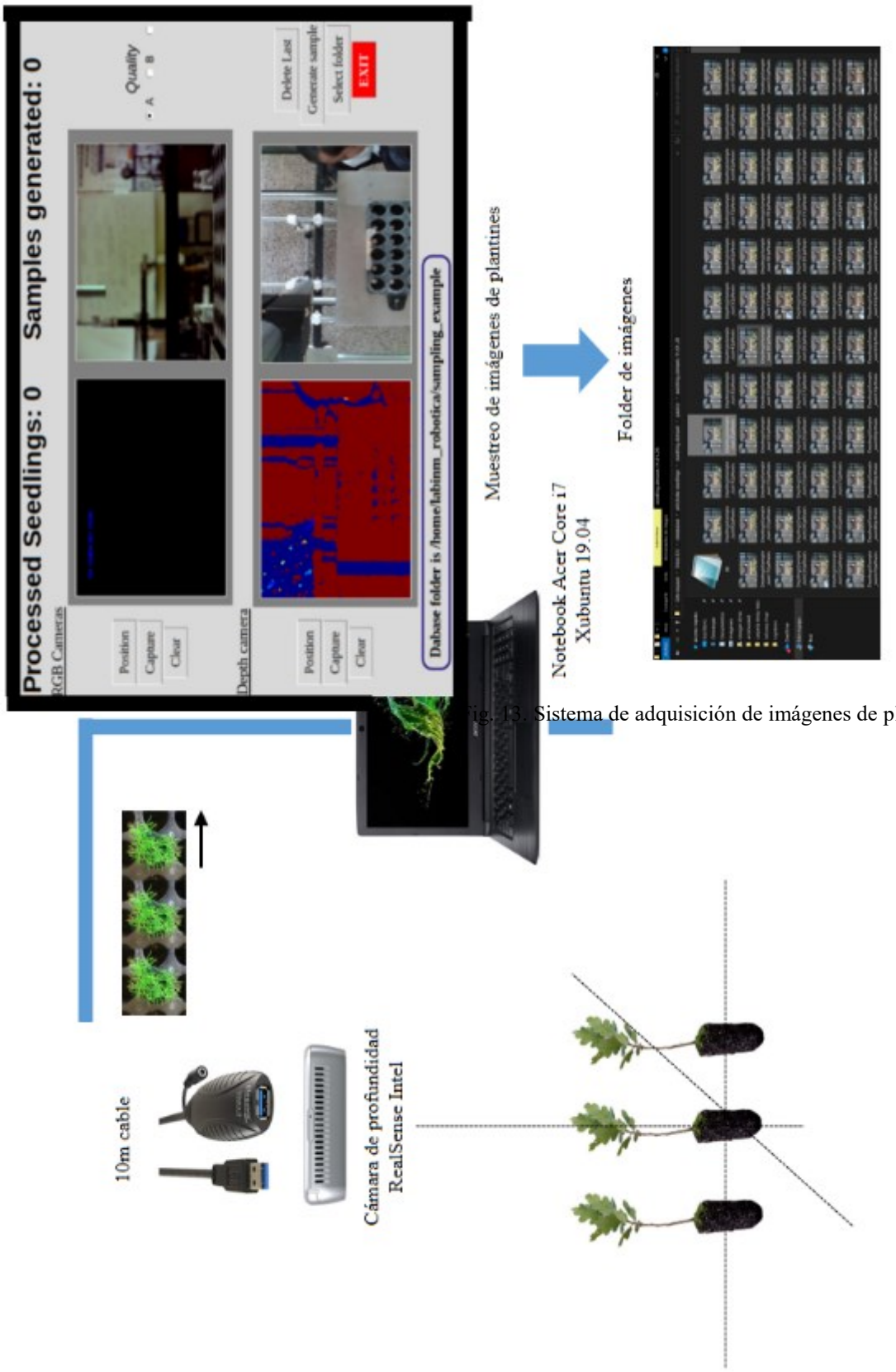


Fig. 13. Sistema de adquisición de imágenes de plantines de alcachofa reales

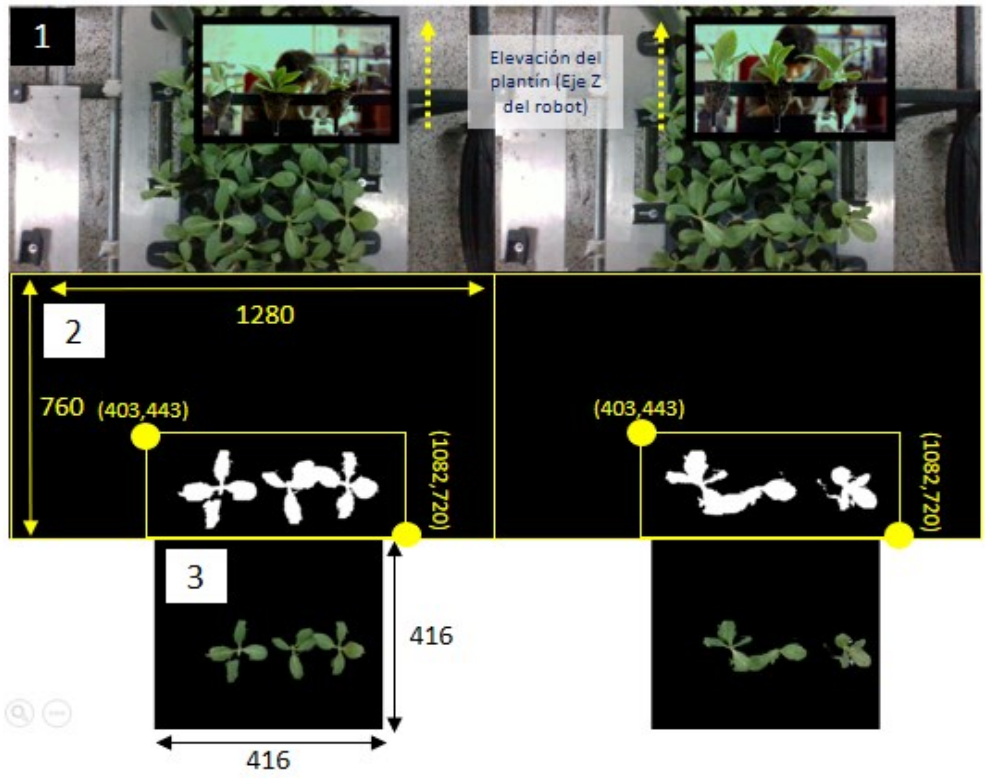


Fig. 14. Capturas de imágen mediante la cámara de profundidad RealSense del robot del Labinm (UPAO). (1) el robot levanta los plantines, (2) la imagen es segmentada, y (3) la imagen es cortada. Fuente: elaboración propia.

En la Fig. 14 se muestra la estrategia usada para el procesamiento digital de imagen de un plantin. Primero, el sistema robotico traslada la bandeja multicelda que contiene a los plantines a una posición dada, dentro de su espacio de trabajo, tal que un dispositivo mecanico-electrico desplaza (eleva) verticalmente (hasta quedar fuera de la bandeja) tres plantines a la vez. Segundo, se toma la imagen con data de profundidad (RGB + *Depth*) tal que los tres plantines elevados están representados en ella. Tercero, se segmenta (creación de la máscara) las regiones de la imagen previamente tomada que representan a los tres plantines elevados (se definen tres siluetas blancas). Finalmente, se enmascara la imagen RGB original con las siluetas blancas y se corta la imagen resultante tal que calce en un marco de fondo negro de 416×416, lo que permite dar mayor eficiencia al coste computacional total.

Es importante indicar que en este trabajo el etiquetado de la calidad de las imágenes de plantines de alcachofa reales, se realiza manualmente (según el modelo ML o DL a usar).

La automatización del mismo no se considero conveniente, por ahora, debido a su alta complejidad, variabilidad y elvado ruido ambiental.

3.6.2. Identificación y diseño de los modelos CNN

Se analiza los modelos más relevantes de CNN que permite trabajar adecuadamente con técnicas de segmentación, clasificación y detección de imágenes para determinar el (o los) modelo que servirá de referencia en esta investigación.

3.6.3. Entrenamiento de los modelos CNN previamente identificados

Se entrena a los modelos CNN identificados de la fase previa mediante un programa basado en las librerías TensorFlow y Keras, enlazadas con técnicas de *Transfer Learning* e integrado a la plataforma *Colaboratory* de Google, que permite usar unidades de procesamiento gráfico (GPU) que aceleran y optimizan el tiempo de entrenamiento. El entrenamiento tiene dos etapas, el de testeo y el de validación de resultados con plantines de alcachofa reales.

3.6.4. Correlación de predicciones y estimaciones

Se trabaja con plantines de alcachofa reales clasificados por el vivero industrial asociado al proyecto. Se obtiene la correlación de Pierson (del que se dispone de una librería en Python) una vez obtenida las predicciones de los modelos propuestos.

3.6.5. Puesta en marcha del modelo para la clasificación de plantines

Aquí se establece la integración del sistema de visión por computadora (SVC) con las siguientes tecnologías: IoT, PLC (Autómata programable) y el sistema robotico manipulador de plantines (Fig. 15). El SVC tiene un programa gestor de visión por computadora que integra y supervisa los módulos, librerías, modelo de IA y la lógica de funcionamiento. Recibe del PLC la orden de calcular las calidades (buena, regular o mala) del conjunto de tres plantines elevados (según procedimiento previamente descrito). Posteriormente dicha información es enviada al PLC, iniciándose el proceso de manipulación y repique de los plantines en sus bandejas correspondientes.

La comunicación entre módulos, se realiza mediante Broker MQTT de Mosquitto, por ser un protocolo de comunicación estable, robusto y muy usado en proyectos de IoT. Adicionalmente, se usa el protocolo de comunicación industrial MODBUS TCP entre el SVC y el PLC, lo que posibilita trabajar con ethernet y *AccessPoints* con antena y por lo tanto implementar comunicación inalámbrica.

El SVC desarrollado en este trabajo combina servidores de dashboard webs y base de datos que se ejecutan en la red local y en servidores en la nube, es decir, en la platforma de Google llamada *Google Cloud Platform* (GCP).

3.7. Diseño de contrastación

Tipo de Estudio:

• Investigación aplicada.

Diseño del estudio:

• Explicativo-experimental.

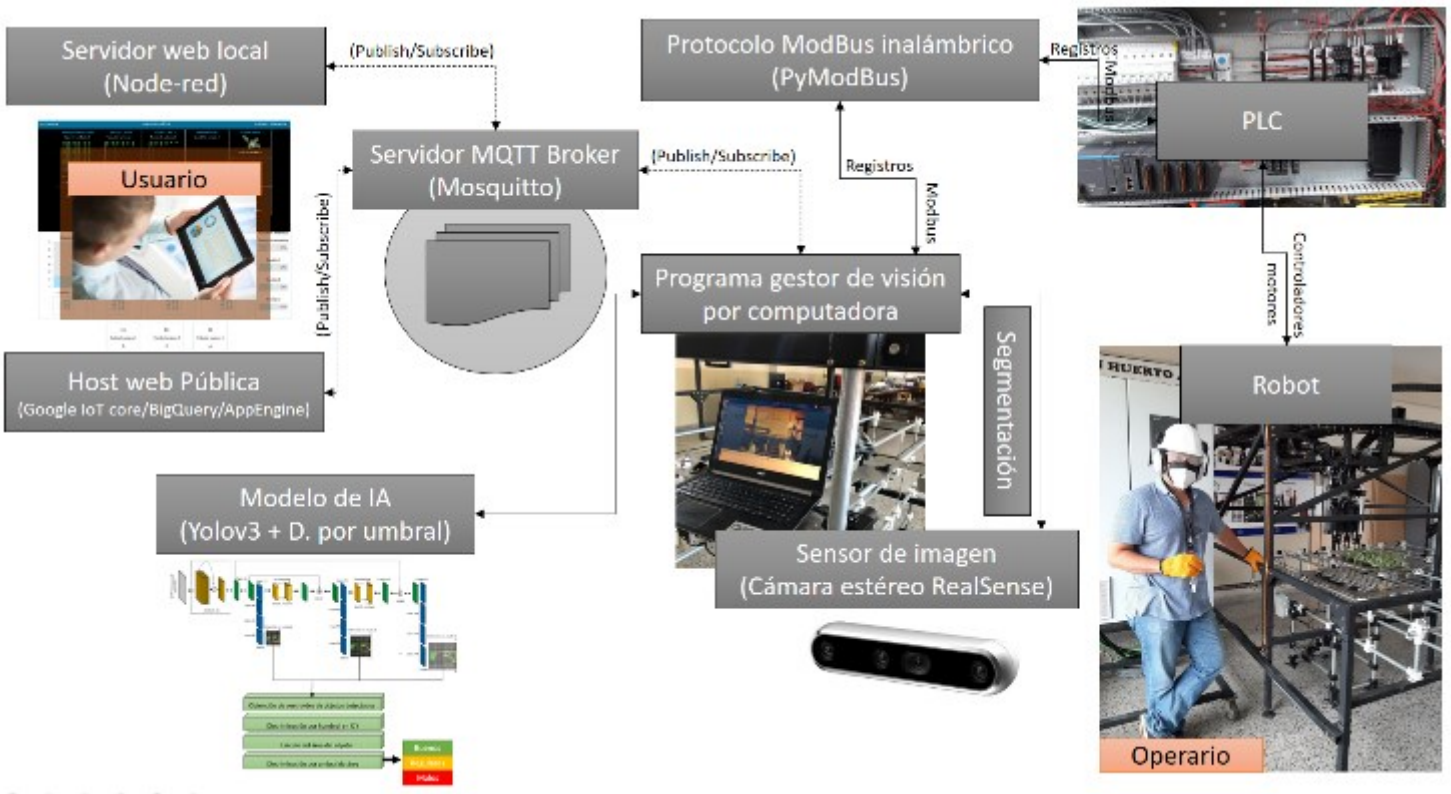


Fig. 15. Esquema general de la visión por computadora basada en IA integrada con las tecnologías IoT, PLC y robots industriales. Fuente: Elaboración propia.

3.8. Procesamiento y análisis de datos

Se realizan tablas y gráficas de las correlaciones entre las mediciones de las diferentes arquitecturas CNN con aquellas mediciones provenientes de los viveros industriales de la región La Libertad-Perú.

A través del coeficiente de correlación de Pierson se obtiene la correlación entre O1 y O2. Mientras que O1 está sujeto a X que está en función del tipo de arquitectura de CNN que se utilice, O2 está sujeta a la clasificación de los plantines de alcachofa según los parámetros (Tabla 1) de la agroindustrial Agrogénesis. Entonces se comparará cuál de las CNN tiene mejor correlación con O2 (Fig. 16).

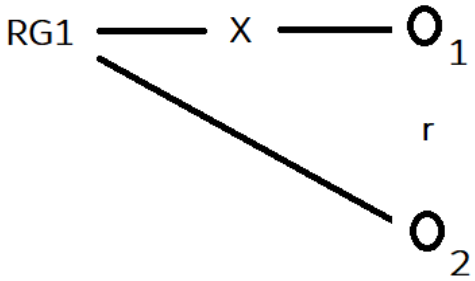


Fig. 16. Operacionalización de las variables. Fuente: Elaboración propia.

CAPÍTULO IV

RESULTADOS

"El que aprende y aprende y no practica lo que sabe, es como el que ara y ara y no siembra"

Platón.

CAPÍTULO IV: RESULTADOS

Los indicadores de las variables independiente y dependiente se obtienen según las cinco siguientes fases: elaboración del dataset de imágenes de plantines de alcachofa, identificación y diseño de los modelos CNN, entrenamiento de los modelos CNN previamente identificados, correlación de predicciones y estimaciones y finalmente, y la puesta en marcha del modelo para la clasificación de plantines.

4.1. Impementación del *dataset* de imágenes de plantines de alcachofa

Aquí se presenta el modelamiento del crecimiento del plantín de alcachofa (en lenguaje L+C), la síntesis de plantines virtuales de alcachofa, el muestreo de imágenes de plantines reales y el etiquetado de las imágenes de plantines virtuales y reales.

4.1.1. Modelamiento y síntesis de plantines virtuales de alcachofa

En la Fig. 17 se muestran los resultados del modelo del modelo del plantín de alcachofa (*cynara cardunculus scolymus*) en su versión 6 (en base a los atributos de crecimiento que presenta este modelo es que se determina una versión diferente). La que se explica a continuación:

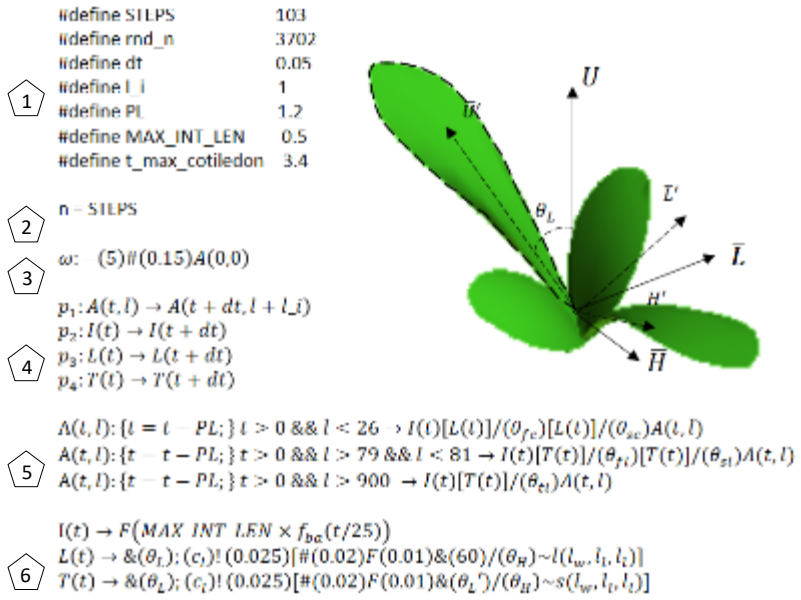


Fig. 17. Modelo del plantín de alcachofa en código L+C. (1) Definición de constantes, (2) steps o producciones, (3) declaración del axioma, (4) declaración de las reglas de producción, (5) decomposición y (6) homomorfismo. Fuente: elaboración propia.

- 1: se definen los siguientes parámetros:
  - **STEPS:** número de producciones a realizar.
  - **rnd_n:** número semilla para la función de generación de número aleatóreos.
  - **dt:** fracción de número que representa el tiempo que pasa de producción en producción. Es un número referencial, no representa estrictamente el tiempo; es regulable heurísticamente. Este valor se va acumulando en "t" cada producción.
  - **l_i:** constante de incremento por cada producción para reconocer la fase de la planta y poder diferenciar crecimiento de cotiledones (primeras dos hojas de germinación de una semilla) de hojas de alcachofa.
  - **PL:** parámetro que permite la generación de una hoja en una posición y orientación específicas y dentro del conjunto de una nueva rama. Al dividir PL por dt, se obtienen el número de producciones que se necesitan para llegar a iniciar la nueva rama. Hay una variable "l" que ayuda a establecer principalmente 2 fases de crecimiento que diferencia a las cotiledonas de las hojas de la alcachofa.
  - **MAX_INT_LEN:** limita la longitud máxima del internodo, la cual es la unión entre dos nodos (puntos en el que se unen dos o más ramas).
  - **t_max_cotiledon:** este parámetro limita el crecimiento del cotiledón una vez iniciada la fase de crecimiento de hojas de alcachofa.
- 2: n es el número de pasos (producciones) a considerar para el crecimiento del plantín.
- 3: el axioma dice que la tortuga debe rotar en sentido antihorario un ángulo de 5°, luego ensanchar la línea rastro de la tortuga 0.15mm y aplicar la descomposición de A con los argumentos de entrada $t = 0\ y\ l = 0$.
- 4: se tienen 4 reglas de producción. Cada una con su respectiva descomposición. "A" para definir la topología del plantín de acuerdo a cada fase de desarrollo de las hojas; en cada producción se considera $t = t + dt\ y\ l = l + l\_i$. "I" para definir el crecimiento del tallo (internodo) que es la unión entre ramas. "L" es la

orientación y forma de los cotiledones. “T” es la orientación y forma de las hojas originales de la alcachofa.

- 5: la descomposición presenta principalmente 2 fases de crecimiento del plantín, y 1 fase más de crecimiento de la quinta hoja de alcachofa hacia adelante (que no se consideran para la versión 6 porque representa una edad mayor del plantín no relevante para los fines de este estudio). En cada producción ($t = t + dt\ y\ l = l + l\_i$) a “t” se le resta el valor de PL lo cual dará un resultado mayor a 0 en la producción p = 25 $= (\frac{PL}{dt} + 1)$ siendo “l” menor a 26, por ende se inicia con los comandos para el renderizado del par de cotiledones. Esta es la primera reescritura, se reemplaza en el axioma lo concerniente a A($t \geq 0.05, l \geq 25$), obteniendo:

$$G_{p=25} = -(5)\#(0.15)I(t \geq 0.05)[L(t \geq 0.05)]/(\theta_{fc})[L(t \geq 0.05)]/(\theta_{sc})A(t \geq 0.05, l \geq 25) \quad (15)$$

donde:

$\theta_{fc} = 130 + ran(90)$, y representa el giro del ángulo ($\geq 130°$) para el segundo cotiledón con una función de generación de números aleatorios uniformemente distribuidos en un intervalo [0,90°]. Se busca que el segundo cotiledón gire con respecto al eje $\vec{U}$ (Fig. 9) un ángulo mínimo de 130° y máximo de 220°; lo cual le da variabilidad a la producción de los plantines. El valor de $\theta_{sc} = 60 + ran(40)$ es el giro del ángulo ($\geq 60°$) para el desarrollo de la primera hoja de la siguiente fase (siguiente reescritura) con una función de generación de números aleatorios uniformemente distribuidos en un intervalo [0,40°]. Los valores aleatorios permiten producir plantines con características singulares. Los homomorfismos $I(t \geq 0.05)\ y\ L(t \geq 0.05)$, son explicados más adelante. Se consideran desigualdades como valores de tiempo $t \geq 0.05$ debido a que en las siguientes producciones se tiene $0.05 + dt$, y se establece que A recibirá nuevamente $t \geq 0.05$ cada vez que se cumplan las condiciones de las descomposiciones y se reescriba nuevamente una palabra G.

Para la segunda fase se tiene que cumplir $t \geq 0$ y $l > 79 \wedge l < 81$ (lo cual se da en la producción $p = 80$), entonces se realiza la siguiente reescritura para así incluir las dos primeras hojas de alcachofa.

$$G_{p=80} = -(5)\#(0.15)I(t \geq 2.8)[L(t \geq 2.8)]/(\theta_{fc})[L(t \geq 2.8)]/(\theta_{sc})$$
$$I(t \geq 0.05)[T(t \geq 0.05)]/(\theta_{fl})[T(t \geq 0.05)]/(\theta_{sl})A(t \geq 0.05, l \geq 80) \quad (16)$$

donde:

$\theta_{fl} = 120 + ran(120)$ es el ángulo que ubica a la segunda hoja de alcachofa, la cual girando alrededor del eje $\vec{U}$ un ángulo mínimo de $120°$ y máximo de $240°$, se intenta ubicar a la hoja enfrente de la primera, con ligeras desviaciones para darle más realismo. Por otro lado el ángulo $\theta_{sl} = ran(360)$ ubica a la tortuga para que inicie dibujando la hoja de la tercera fase. Ésta última como se mencionó líneas arriba, está prácticamente deshabilitada ($p \geq 80$ para activarse) debido a que no se necesita de su ejecución para los fines que busca el proyecto. El homomorfismo de $T(t \geq 0.05)$ se explica a continuación.

- 6: la parte de homomorfismo inicia con $I(t)$ que representa el desarrollo de un internodo (tallo entre dos ramas). La tortuga traduce este homomorfismo como un desplazamiento atenuado por la constante MAX_INT_LEN y gobernado por la función normalizada $f_i(t/25)$ que a medida que va creciendo t el internodo se va extendiendo. El internodo (o tallo) para un plantín de alcachofa es un desarrollo minúsculo, casi imperceptible. La función f_i en la Fig. 18 viene a ser la identificada con el número 3.

En el homomorfismo $L(t)$ se consideran las siguientes funciones:

$$\theta_L = -ran(45) + 26f_{ba}(t/30) \quad (17)$$
$$c_l = 32 + floor(21f_c(t/20)) \quad (18)$$
$$\theta_H = ran(35) \quad (19)$$
$$l_w = f_{lw}(t/12) \quad (20)$$
$$l_l = f_{ll}(t/10) \quad (21)$$

Donde las funciones $f_{ba}, f_c, f_{lw}, f_{ll}$ en la Fig. 10 son las correspondientes a los número 1, 2, 4, y 5, respectivamente. Estas funciones son modeladas manualmente a través del LStudio y ayudan a establecer los parámetros de crecimiento del plantín.

En el homomorfismo $T(t)$ se consideran las siguientes funciones (c_l se mantiene igual que la Ec. 18):

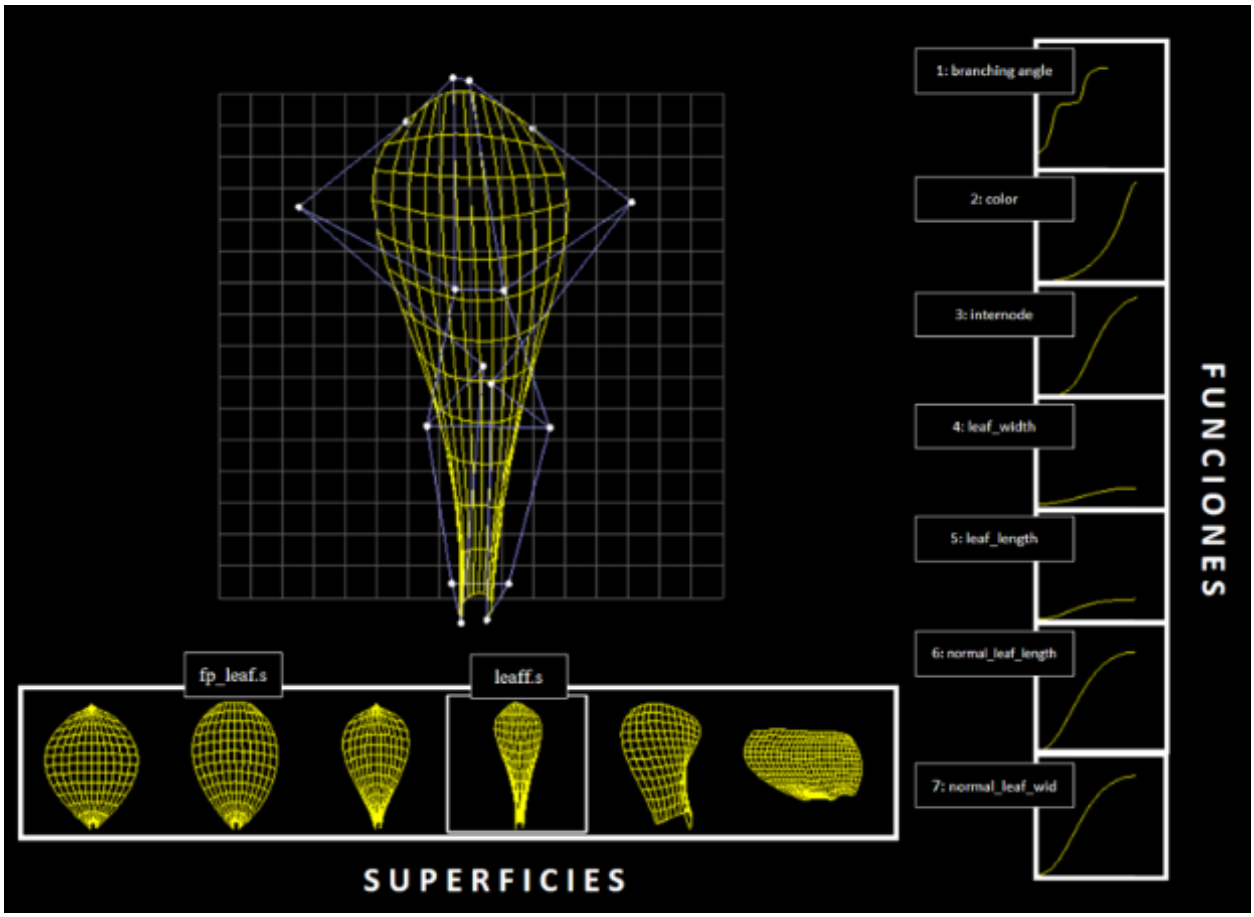


Fig. 18. Establecimiento arbitrario de parámetros a través del programa L-Studio: edición de superficies y funciones de crecimiento del plantín de alcachofa. Fuente: elaboración propia.

$$\theta_L = 6f_{ba}(t/33) \quad (22)$$

$$\theta_L' = ran(20) + 8 + 36f_{ll}(t/5) \quad (23)$$

$$\theta_H = ran(20) \quad (24)$$

$$l_w = f_{lw}(t/16) \quad (25)$$

$$l_l = f_{ll}(t/13) \quad (26)$$

Por otro lado, el modelamiento de las superficies de las hojas del plantín usa el signo el signo ~ seguido de una letra l o s, hoja del cotiledón y de alcachofa, respectivamente (Fig. 17, parte 6). Tanto l como s tienen como parámetros de entrada primero el ancho, el alto y el largo de la superficie. En la Fig. 18 se observa la interfaz gráfica del LStudio en la cual se hace el modelamiento manual de la superficie de la hoja, modificando la posición de los puntos blancos guiados por la cuadrícula del fondo: *leaff.s* para la hoja de alcachofa y *fp_leaf.s* para la hoja cotiledón.

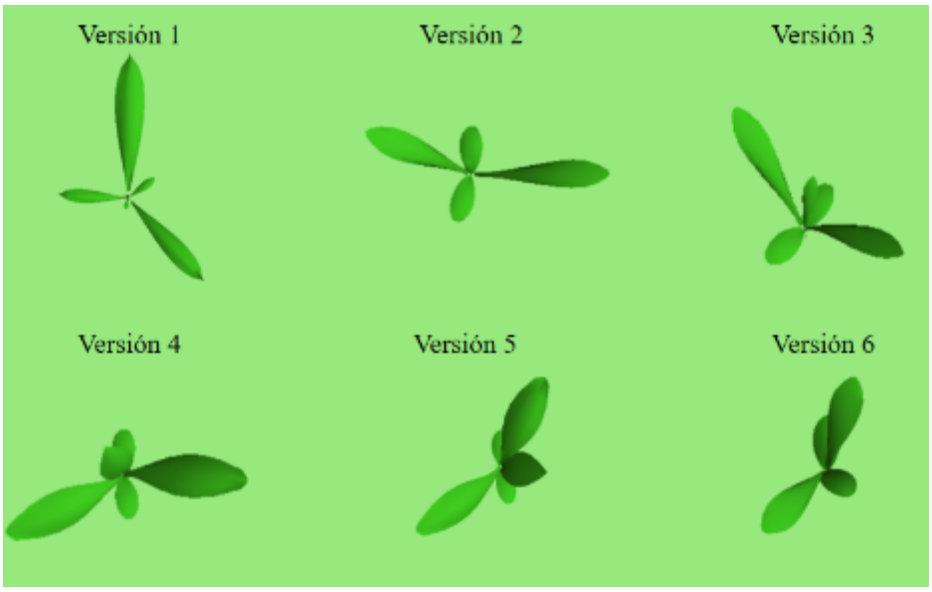


Fig. 19. Versiones desarrolladas de los plantines de alcachofa hacienda uso del lenguaje L+C en el software LStudio. Fuente: elaboración propia.

Es de relevancia recalcar la utilidad de los ángulos $\theta_{fc}, \theta_{sc}, \theta_{fl}, \theta_{sl}, \theta_L, \theta'_L$ y θ_H porque la variabilidad que le otorgan a la orientación de los cotiledones y hojas, así como internodos, hace posible que se puedan reproducir innumerables plantines diferentes; si se considera solo 20° de variación en cada uno de aquellos, habrían $20^7 = 1.28 \times 10^6$ plantines distintos.

En los Anexos II se presenta el programa completo del modelo del plantín desarrollado en esta tesis. Existen diferentes versiones de estos programas que generan plantines con diferentes atributos morfológicos. En la Fig. 19 se muestran las seis versiones más resaltantes.

A través del RPA se producen alrededor de 10 mil imágenes de plantines versión 2 (v2), y 3 mil imágenes de plantines v6 (Ver código fuente en Anexos III). A partir de estas imágenes de plantines se obtienen las imágenes de plantines agrupados.

Los plantines agrupados (Fig. 20), se distribuyen aleatoreamente en un arreglo de 3 x 3, emulando con ello la ubicación de plantines en una bandeja multicelda. Aquí se usan 3mil plantines v6 (en el Anexo IV se muestra el código del programa).

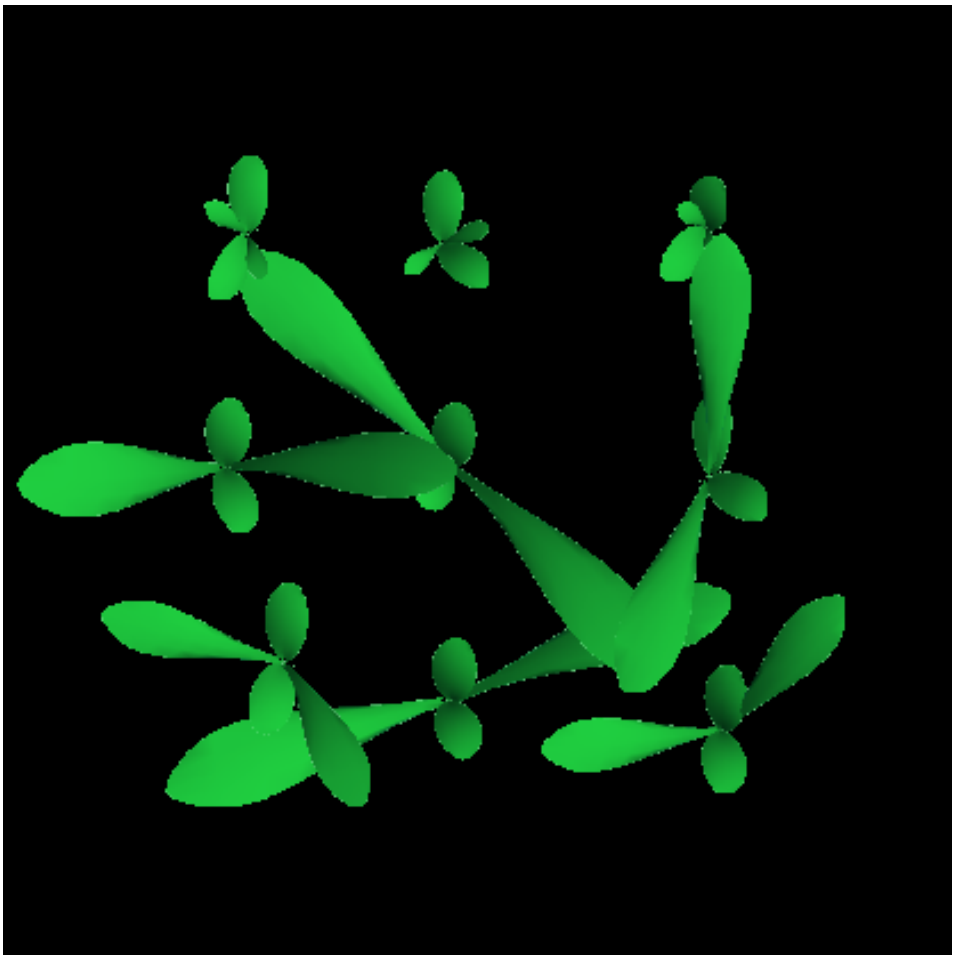


Fig. 20. Imágen de plantines de alcachofa sintetizados y agrupados. Fuente: elaboración propia.

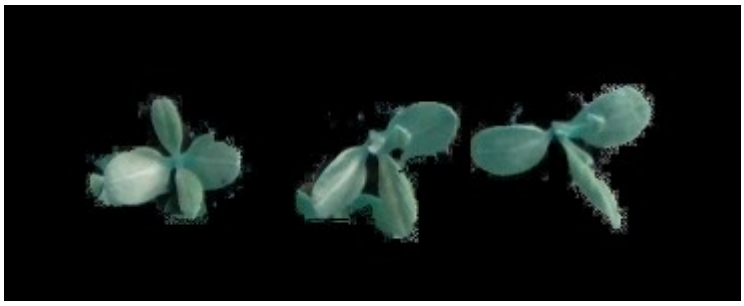
Se generan un total de 10 mil imágenes de los cuales 3mil son usados para entrenar los modelos propuestos. Todas las imágenes de plantines producidos se encuentran en el CD adjunto, carpeta “PLANTINET”.

4.1.2. Captura de imágenes de plantines reales de alcachofa

Los modelos de detección y clasificación de plantines propuestos en este trabajo son testeados mediante 619 imágenes de plantines reales capturadas por el sistema de visión artificial del robot. El código del programa de muestreo para la adquisición de dichas imágenes puede ser descargado en el siguiente repositorio GitHub: https://github.com/Vegnics/Seedling_vision_sampling.

Cada imagen capturada es segmentada haciendo uso de la información de profundidad teniendo en cuenta el umbral de altura. Como pruebas complementarias se considera un

(a)



(b)

Fig. 21. Plantines reales muestreados por el robot. (a) aplicando solo el umbral de profundidad, (b) luego de la segmentación con K-means. Fuente Elaboración propia.

algoritmo K-means para mejorar la segmentación. Los resultados de dicha segmentación se muestran en la Fig. 21. El código del algoritmo de segmentación con K-means puede ser descargado en el siguiente repositorio: https://github.com/Vegnics/Seedling_vision.

4.1.3. Clasificación, etiquetado y formato de dataset

Aquí se tiene dos formas de agrupar y etiquetar las imágenes. La primera usa carpetas de Windows (ver carpeta PLANTINES en el CD adjunto) según:

- Tres carpetas, uno por cada nivel de calidad: malo, regular y bueno. En cada una de ellas solo se ubican plantines con ese nivel de clasificación de calidad.
- La enumeración de cada plantín contenido en las carpetas previamente mencionadas, siempre empieza en 1, y se incrementa unidad por unidad; adicionalmente, se le agrea la letra b (malo), a (regular) y g (bueno). P. ej., el primer plantin en la carpeta malo es: "*b_1.png*". Se considera el formato *png* por ser éste un archivo de tamaño pequeño.

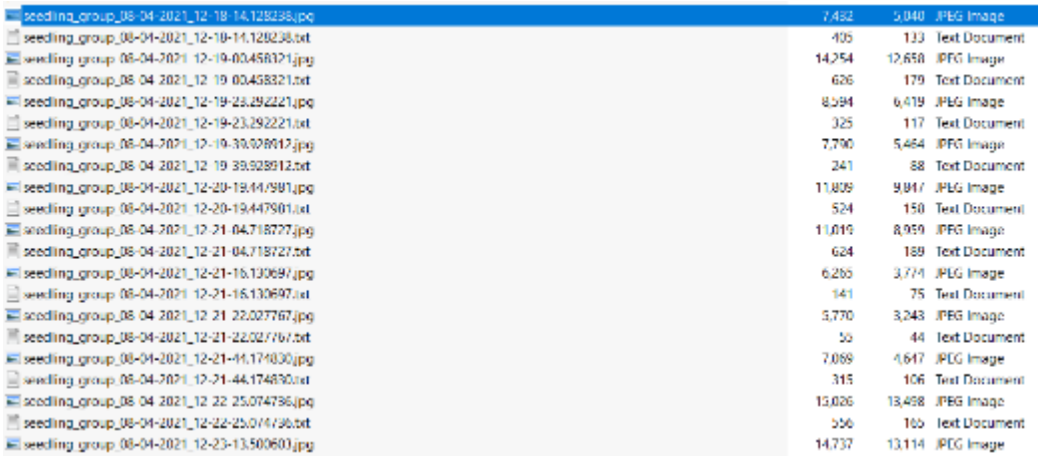


Fig. 22. Muestra de una lista de imágenes de grupos de plantines etiquetadas acorde al formato que requiere el modelo Yolov3 (en *Darknet*). Cada imagen tiene su archivo .txt donde se establecen los detalles mostrados en la Fig. 18. Fuente: elaboración propia.

La segunda forma de agrupar y etiquetar las impaenes define las entradas para el entrenamiento exclusivamente del modelo Yolov3. Aqupi se usa el archivo llamado *"classes.txt"* en el que se listan las clases de objetos a ser identificados; en este caso: regular y bueno. Por medio de un archivo *.txt* se especifican la clase de plantín, las coordenadas normalizadas del centroide del objeto a detectar, así como el ancho y alto (normalziado también) del objeto. Cada imagen tiene su objeto (plantín de alcachofa) así que cada imagen tendrá su correspondiente archivo *.txt* (Fig. 22). En la Fig. 23 se muestra el contenido para el caso de una imagen agrupada de plantines (Fig. 20).

En el Anexo IV se muestra el programa para el etiquetado de las imágenes generadas de los grupos de plantines sintéticos.

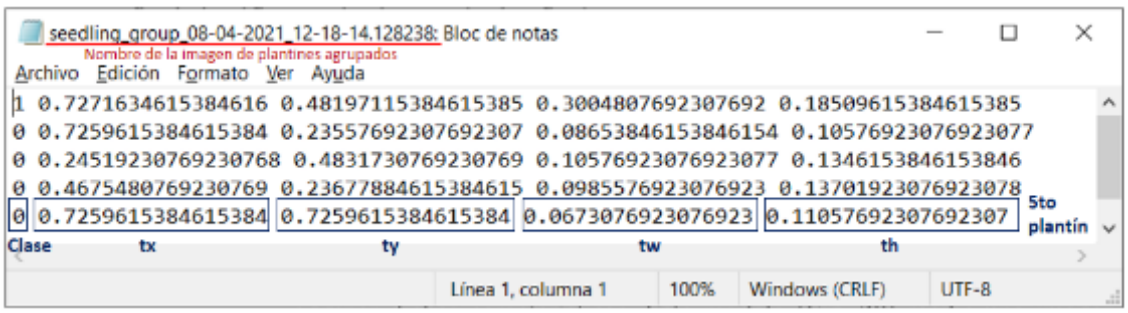


Fig. 23. Archivo .txt para el etiquetado que requiere el Yolov3 (basado en *Darknet*) de los 5 plantines sintéticos incluidos en la imagen en formato .jpg de nombre indicado en la figura. Fuente: elaboración propia.

4.2. Establecimiento y Diseño de los modelos de DL

En esta sección se muestran los cuatro modelos considerados para la detección y clasificación de plantines de alcachofa según la calidad de los mismos. Los resultdos de los mismos permiten tener una comparativa de predicción a fin de valorar el mejor modelo.

4.2.1. Modelo basado en PCA y K-means

Si bien es cierto se ha propuesto trabajar con CNNs, se considera también un modelo de ML basado en los algoritmos de PCA y K-means a fin de comparar desempeños. El primero se centra en la extracción de características (componentes principales) propias de la imagen del plantín, y la segunda enfocada en agrupar dichas características. Este modelo (Fig. 24) recibe una imagen RGB (plantín) a un tamaño que puede variar de acuerdo al tamaño del plantín. La imagen puede ser un tensor de $80 \times 30 \times 3$ píxeles (ancho, alto y 3 canales RGB). Cada pixel y su canal representa una característica de dicha imagen, por ende, se tendrían un vector con 2400 características. Lo que hace el PCA, en este caso, es disminuir la dimensión de dicho vector a 2 valores PC1 y PC2 (representables en un gráfico 2D).

La propuesta de este modelo PCA-Kmeans es agrupar las características de los diferentes tipos de calidad de plantines (basados en el tamaño de los mismos). De esta forma se sabe

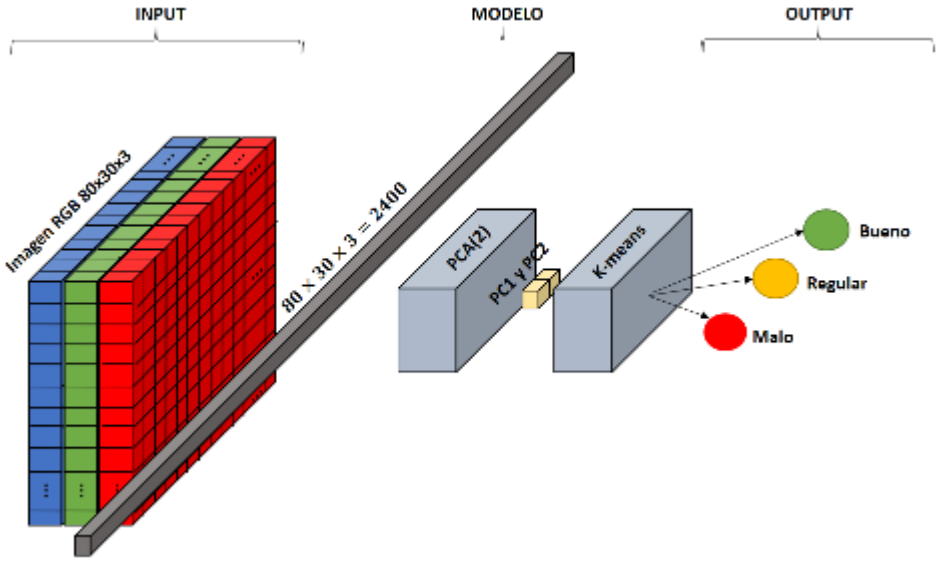


Fig. 24. Modelo de ML: PCA-Kmeans orientado a extraer características (PC1 y PC2) de la imagen del plantín para posteriormente agruparlas según las tres calidades. Fuente: elaboración propia.

si la calidad del plantín es 0 en caso no encuentre plantín en la imagen ($I_{w\times h\times 3}$), o 1 en caso encuentre un plantín de calidad mala (características PC1 y PC2 pertenecen al conjunto C_m), 2 para la calidad regular (PC1 y PC2 pertenecen al conjunto C_r) y 3 para calidad buena (PC1 y PC2 pertenecen al conjunto C_b). Lo mencionado se resume en la ec. (27).

$$C_{plantin} = \begin{cases} M_{PCA-kmeans}(I_{w\times h\times 3}) \in C_b \rightarrow 3 \\ M_{PCA-kmeans}(I_{w\times h\times 3}) \in C_r \rightarrow 2 \\ M_{PCA-kmeans}(I_{w\times h\times 3}) \in C_m \rightarrow 1 \\ M_{PCA-kmeans}(I_{w\times h\times 3}) \in \emptyset \rightarrow 0 \end{cases} \quad (27)$$

4.2.1. Modelo basado en VGG16

Se propone usar también un modelo de CNN basado en Visual Geometry Group o VGG16 (Simonyan-Zisserman, 2014), al cual se le modificó la cabecera a fin de poder adaptar nuevos pesos para la clasificación del plantín según su calidad. En la Fig. 25 se muestra la arquitectura de la propuesta. Está conformada por 24 capas, de las cuales 13 son de convolución, 5 son *max poolings* y en la cabecera se tiene 1 *average pooling* (valor promedio de un grupo de valores), 1 totalmente conectada. Se destaca la utilización de las funciones de activación ReLU y Softmax. Las dimensiones del tamaño de la imagen del plantín son fijas (224×224) por lo tanto se debe respetar el INPUT, en caso la imagen sea menor o mayor, se debe redimensionarla a fin de que encaje como se espera.

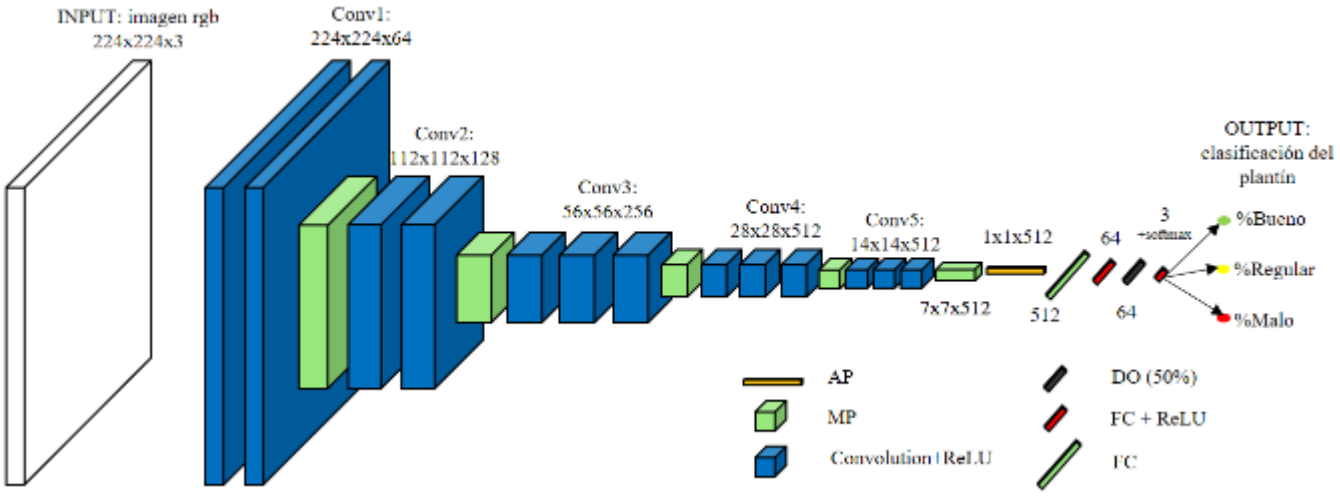


Fig. 25. Modelo VGG16 modificado (24 capas en total) para la clasificación del plantín de alcachofa según su calidad. AP acrónimo de: a*verage* pooling, MP: *max* poolingk, Conv: convolución, ReLU: *rectifier linear unit,* DO: *dropout,* FC: capa totalmente conectada, softmax: función de activación estadística. Los valores que acompañan a los bloques son las dimensiones de los tensores como salidas de dichos bloques. Fuente: elaboración propia.

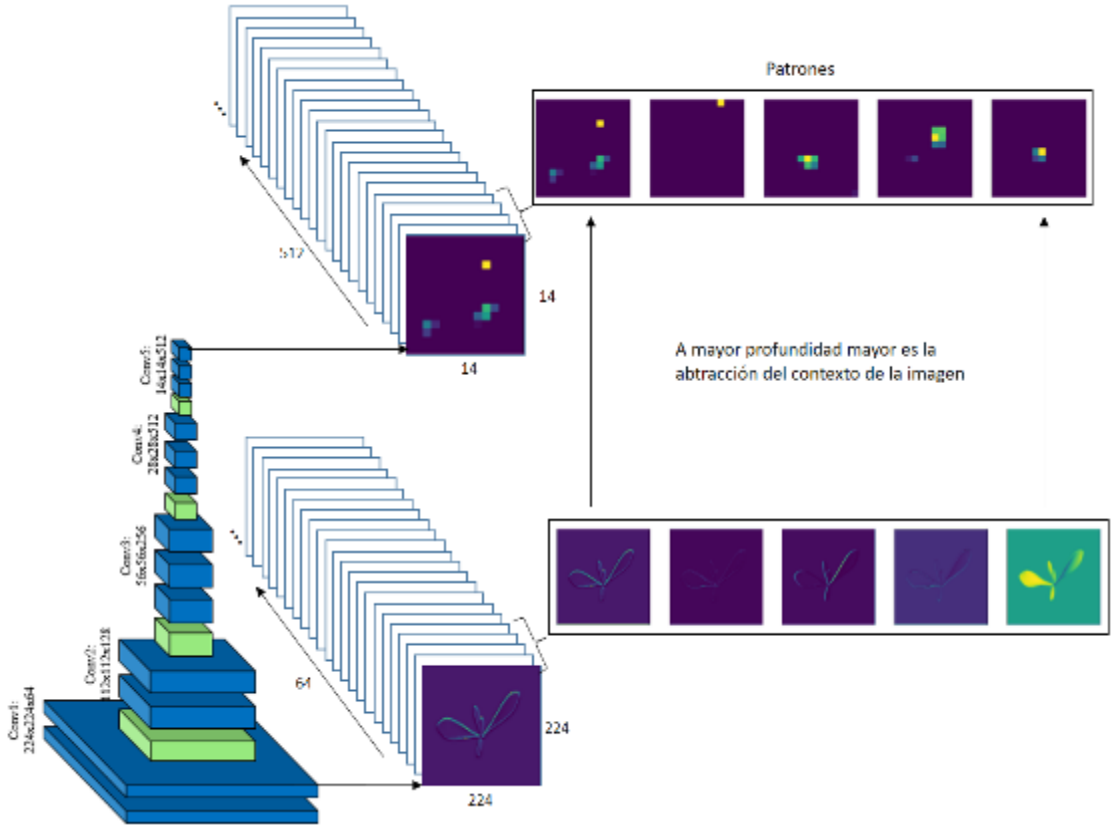


Fig. 26. Mapas de características, cinco primeras matrices de cada tensor salida de la Conv1 y Conv5. Cada una de estas imágenes representa una abtracción por parte del modelo, una interpretación del contexto en función a patrones que en la cabecera son clasificados por medio de las últimas capas. Fuente: elaboración propia.

El propósito de este modelo es extraer las principales características de los plantines como mapas de características (tensores de 64, 128, 256 y 512 de profundidad; ver Fig. 26) que a través de la cabecera compuesta por capas FC se reduce la dimensión (al igual que con PCA) hasta llegar a obtener un vector de 3 elementos (calculados por el softmax). Este vector contiene las probabilidades de cuán malo, regular y bueno es el plantín. El resultado (bueno, 3; regular, 2; malo, 1) es el porcentaje mayor, ec. (28).

$$C_{plantin_{VGG16}} = \begin{Bmatrix} \%bueno > \%malo \wedge \%bueno > \%regular \rightarrow 3 \\ \%regular > \%bueno \wedge \%regular > \%malo \rightarrow 2 \\ \%malo > \%bueno \wedge \%malo > \%regular \rightarrow 1 \end{Bmatrix} \quad (28)$$

$$\%bueno + \%regular + \%malo = 100\% \quad (29)$$

4.2.2. Modelo basado en Yolov3

En este trabajo se usa el modelo Yolov3 como detector de plantines en imagen digital (Fig. 27). Primero se extraen las características de la imagen RGB (tensor de $416 \times 416 \times 3$) a través del modelo *Darknet_53* obteniéndose un mapa determinado de características (semejante a lo mostrado en Fig. 26). Posteriormente los bloques residuales (ResNets) permiten el procesamiento de los mapas a tres diferentes escalas, para lo cual se hace uso de una FPN (*Feature Pyramid Network*). Se considera como parte del FPN a la capa de detección; por cada escala se obtiene una serie de predicciones (Fig. 27). La predicción se expresa mediante un vector que contiene los siguientes 8 elementos:

- t_x: coordenada en x del centroide de la caja que se ajusta a la imagen del plantín detectado,
- t_y: coordenada en y del centroide de la caja que se ajusta a la imagen del plantín detectado,
- t_w: *offset* predicho del ancho con respecto al centroide del *cluster*,
- t_h: *offset* predicho del alto con respecto al centroide del *cluster*,
- p_s: *score* de predicción,
- q_1: clase de la calidad: plantín malo,
- q_2: clase de la calidad: plantín regular,
- q_3: clase de la calidad: plantín bueno

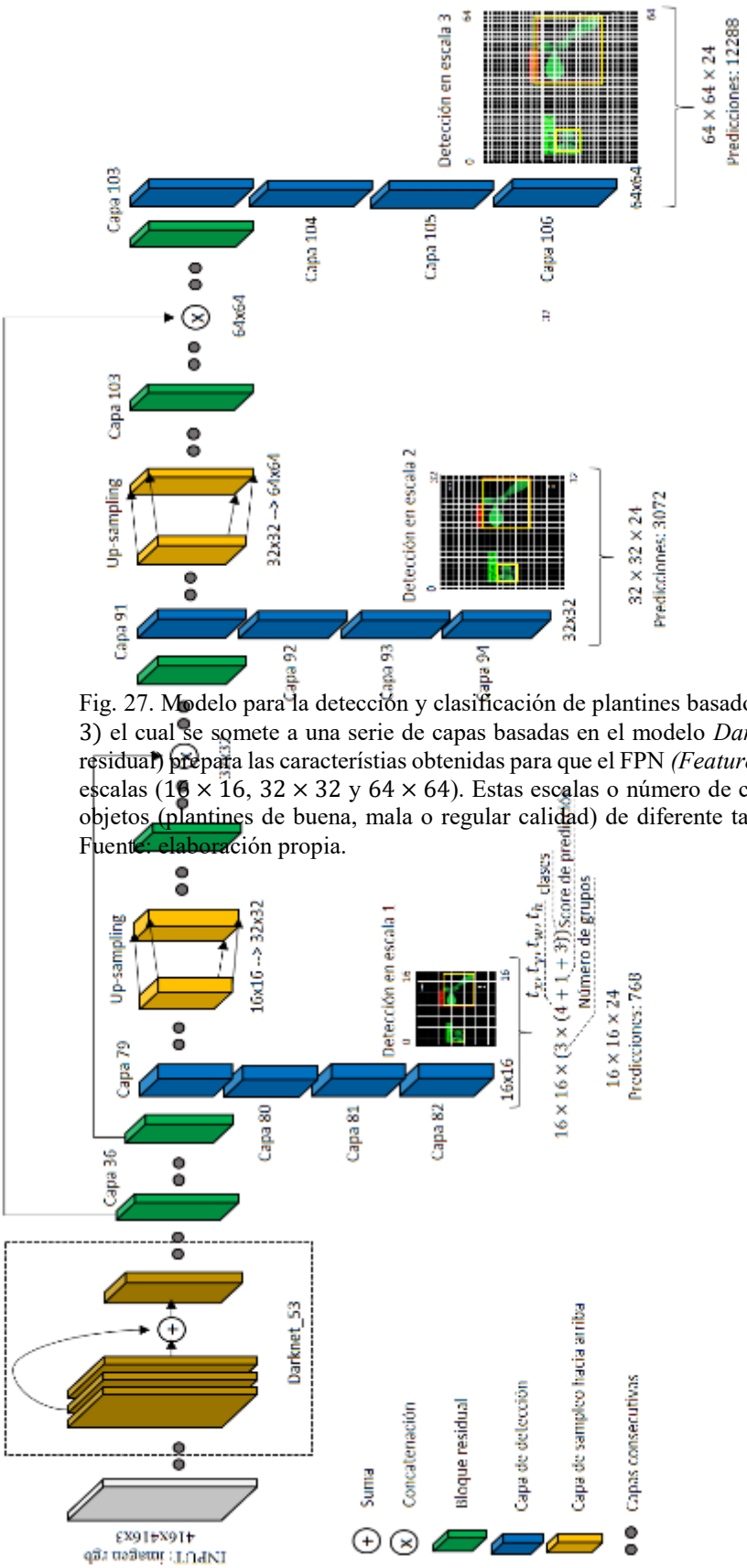


Fig. 27. Modelo para la detección y clasificación de plantines basado en Yolov3 (106 capas en total). La entrada es
3) el cual se somete a una serie de capas basadas en el modelo *Darknet 53* a fin de extraer los mapas de caracte
residual) prepara las características obtenidas para que el FPN *(Feature Pyramid Network)* permita aplicar las convolu
escalas (16×16, 32×32 y 64×64). Estas escalas o número de cuadrículas en la imagen permiten una granular
objetos (plantines de buena, mala o regular calidad) de diferente tamaño. Las capas 82, 94 y 106 entregan las pr
Fuente: elaboración propia.

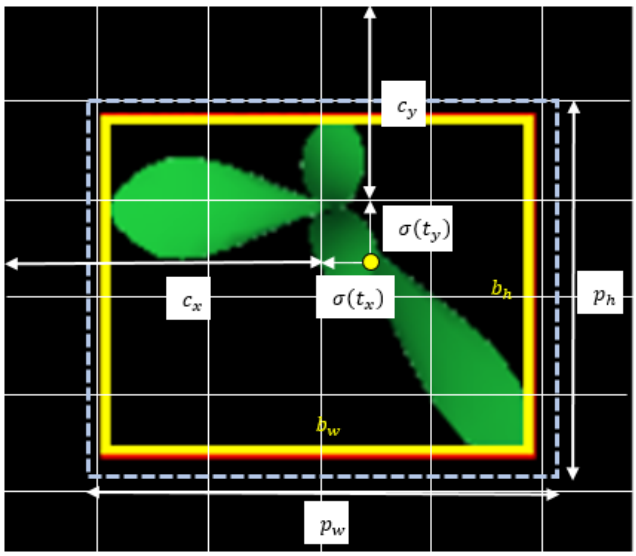


Fig. 28. Visualización de las componentes de las ec. (30) - (33) representados en la predicción de la detección de un plantín, considerando como ej. N=6. Cada cuadrícula realiza 3 predicciones. Fuente: elaboración propia.

La regla del cálculo de las dimensiones del tensor en la salida de las capas de detección (82, 94 y 106) se basa en la siguiente regla:

$$N \times N \times (G \times (T + Ps + Q)) \tag{30}$$

donde:

- N es una dimensión de la cuadrícula que divide la imagen en imágenes más pequeñas. En la capa 82 se consideran 16×16 cuadrículas. Cada cuadrícula es procesada por las CNN de las capas de detección como pequeñas imágenes que unidas dan la general. Cada cuadrícula entrega 3 predicciones, y cada predicción como se menciona líneas arriba, cuenta con 8 elementos. Los cuatro primeros permiten determinar las coordenadas del centroide y el ancho y alto de la imagen del plantín: b_x, b_y y b_w, b_h, respectivamente.

$$b_x = \sigma(t_x) + c_x \tag{30}$$

$$b_y = \sigma(t_y) + c_y \tag{31}$$

$$b_w = p_w e^{t_w} \tag{32}$$

$$b_h = p_h e^{t_h} \tag{33}$$

En la Fig. 28 se pueden visualizar las variables de estas 4 predicciones. La naturaleza de t_x y t_y depende de la función sigmoidea que entrega un valor entre 0 y 1 y que requiere de los valores de c_x y c_y para poder dar valores certeros del centroide del plantín con respecto a toda la imagen. Los valores c_x y c_y son los

offsets de la cuadrícula analizada por el extractor de características (*Darknet_53*), los cuales sumados al resultado de las funciones sigmoideas $\sigma(t_x)$ y $\sigma(t_y)$, respectivamente, se establece el centroide del plantín. Cada predicción por cuadrícula hace uso de los "predecesores" (*priors*) p_w y p_h obtenidos a partir de unas cajas de referencia (o *clusters*) que varían de tamaño según la escala de la capa de detección a fin de predecir el ancho (b_w) y alto (b_h) del marco amarillo que encierra al plantín. Estos predecesores se modifican de acuerdo al *offset* predicho (t_w ó t_h) con respecto al centroide del *cluster*. Los *clusters* facilitan la predicción del ancho y alto del marco (amarillo) que se ajusta al plantín detectado. El Yolov3 hace uso de 9 *clusters* obtenidos a partir del algoritmo k-means al agrupar el ancho y alto de los objetos en el *dataset* MS COCO. El promedio de estos valores agrupados permite obtener el ancho y alto de cada *cluster*, es decir, los antecesores (*priors*) p_w y p_h, respectivamente. El modelo trabaja con 3 *clusters* por cada escala (Fig. 25). Al entrenar este modelo con el *dataset* de los plantines, estos *priors* se actualizan.

  - G es el conjunto de *clusters* que actúan en cada escala, por ende, su valor es 3.
  - T es el grupo de 4 valores de predicción: t_x, t_y, t_w, t_h.
  - Ps es el *score* de predicción, o sea, un valor que permite medir la confianza de la predicción de cada cuadrícula. Es importante establecer que cada cuadrícula "considera" que tiene el valor del centroide de la imagen del plantín dentro de sus límites. Pero la cuadrícula que en realidad sí tiene el verdadero centroide se diferencia porque su Ps es el mayor de todos. En la sección 2.2.5.3 se puede encontrar la ec. (5) que define al Ps.

- Q es el grupo de clases que al igual que en el VGG16, se obtiene por medio de un *softmax* que permite una distribución de probabilidad. La clase con mayor porcentaje representa la predicción de clasificación.

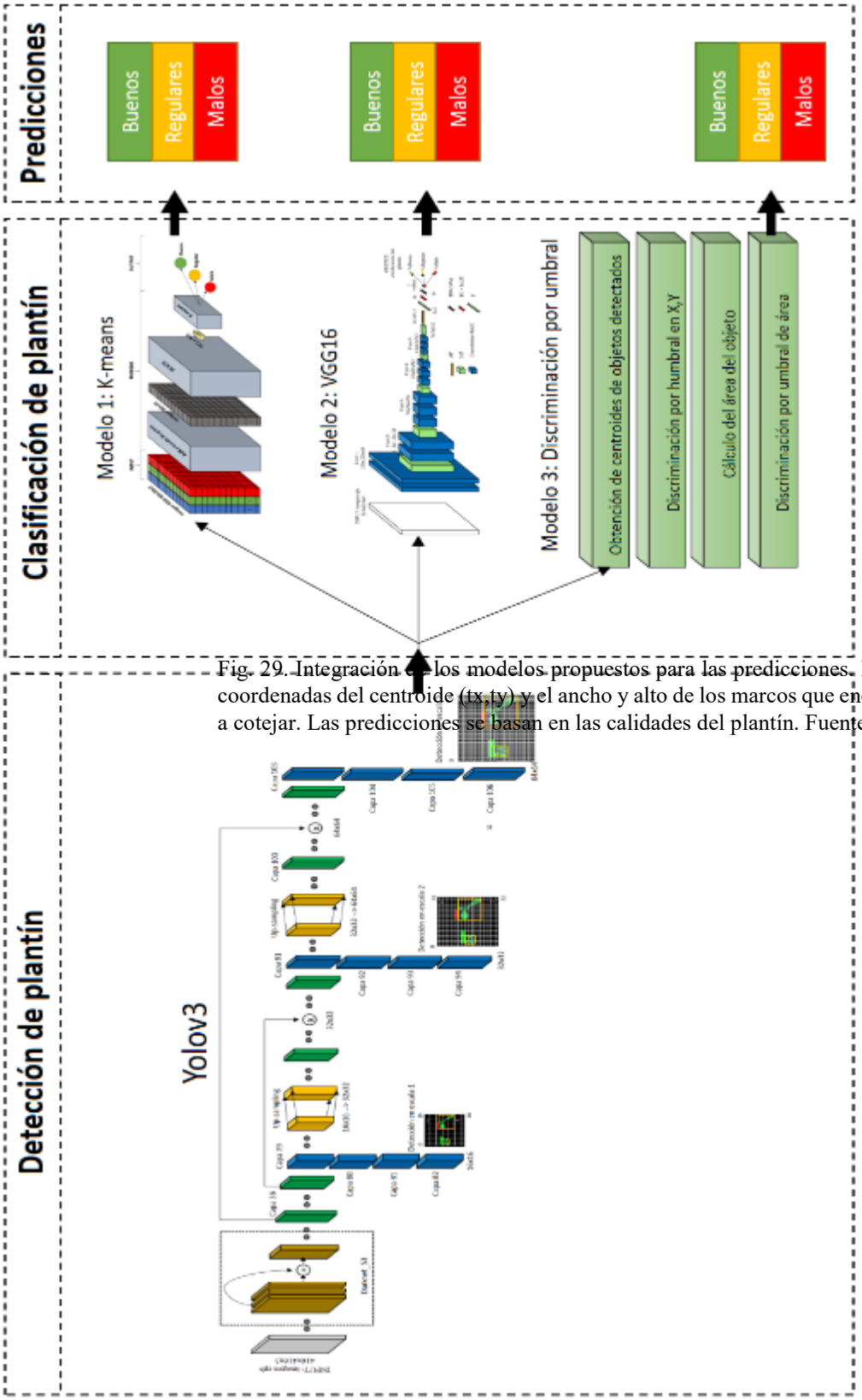


Fig. 29. Integración de los modelos propuestos para las predicciones. La parte de detección de plantín coordenadas del centroide (tx, ty) y el ancho y alto de los marcos que encierran al plantín. Con relación a a cotejar. Las predicciones se basan en las calidades del plantín. Fuente: elaboración propia.

Nótese que el modelo Yolov3 se usa, en este trabajo, para detectar el plantín en imagen digital con ello se corta el marco que se ajusta al plantín digital para que sirva como entrada al otro modelo encargado de clasificarlo. Se considera así debido a la experiencia que se tuvo al momento de entrenar y testear este modelo para detectar el plantín. Si bien es cierto el Yolov3 también puede clasificar, hay algunos aspectos por configurar en su tarea de clasificación con relación a plantines; la modificación directa del modelo Yolov3 incluye también un reentrenamiento desde cero del mismo, lo cual no está incluido en este trabajo debido al poder computacional que se necesita para tal fin. Por ello, la tarea de clasificación se la designa a los modelos mostrados en la Fig. 29.

Finalmente, en la Tabla 7 se muestra una comparación de los modelos propuestos con respecto al número de capas y al número de parámetros que requieren para funcionar.

Tabla 7. Comparación del número de capas y parámetros de los modelos propuestos.

Modelo	**Número de capas**	**Número de parámetros**
PCA-Kmeans	3	0
VGG16	24	138,000,000
YOLOv3	106	65,252,682
Discriminación por umbral	1	0

4.3. Entrenamiento, testeo y validación de los modelos propuestos

En esta Sección se muestra los resultados del entrenamiento, testeo y validación de los modelos para la clasificación de plantines (ML y DL, ver Sección 4.2) usando las imágenes de platines de alcachofa sintéticos (ver Sección 4.1).

4.3.1. Entrenamiento del modelo basado en PCA y K-means

Aquí se usan 1000 imágenes de plantines por cada nivel de calidad (mala, regular y buena) y se consideran las cuatro versiones de crecimiento virtual de plantines de alcachofa que más similidut tienen con respecto a unos reales (Fig. 30). A todas las imágenes se le aplica el algoritmo PCA obteniéndose en cada una de ellas dos componentes principales: PC1 y PC2 (ver Anexo V, el código del programa desarrollado).

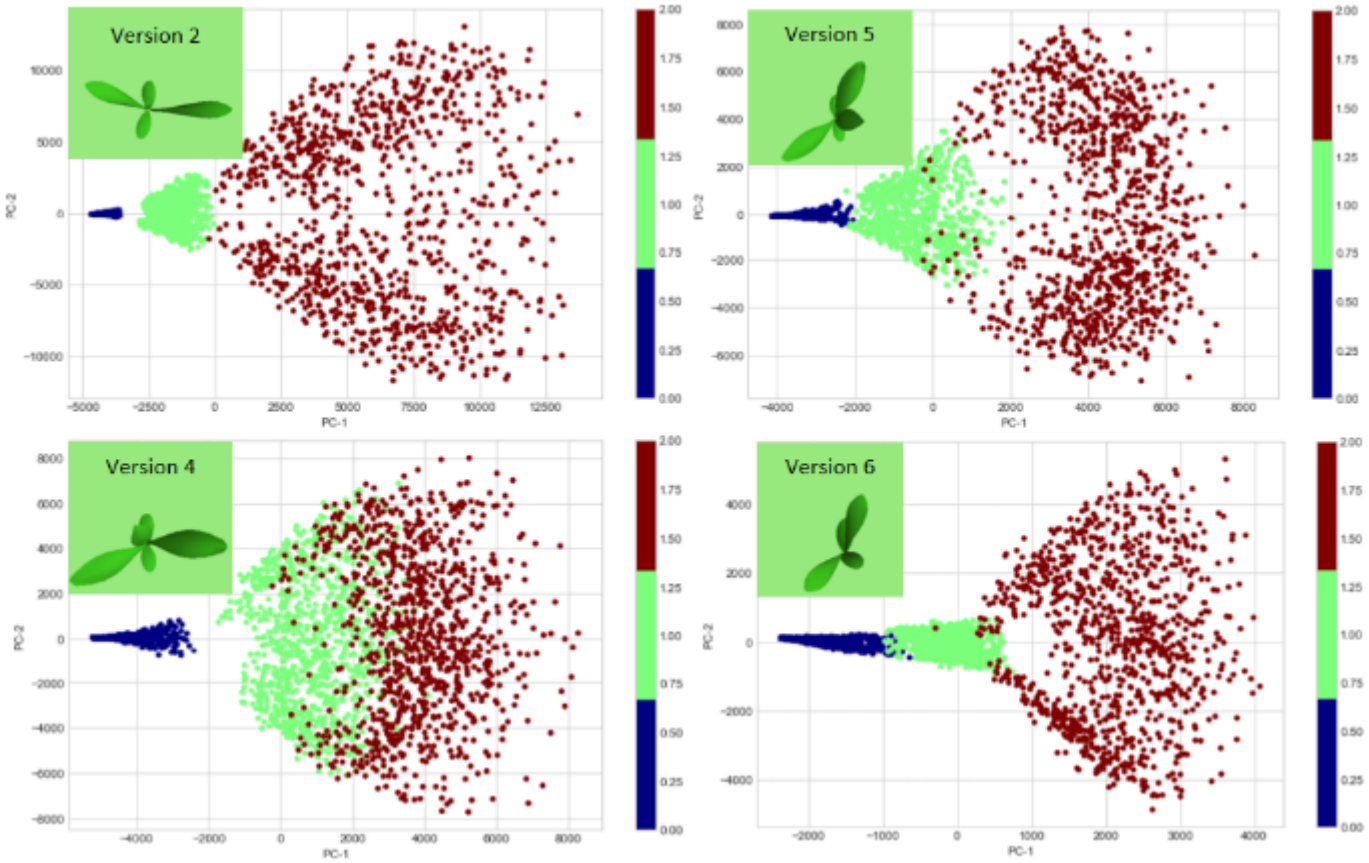


Fig. 30. Gráficas de componentes PC1 y PC2 de cuatro versiones de los modelos de plantines sintéticos. Se muestran las tres calidades: malos (azul), regulares (verde) y buenos (marrón). Fuente: elaboración propia.

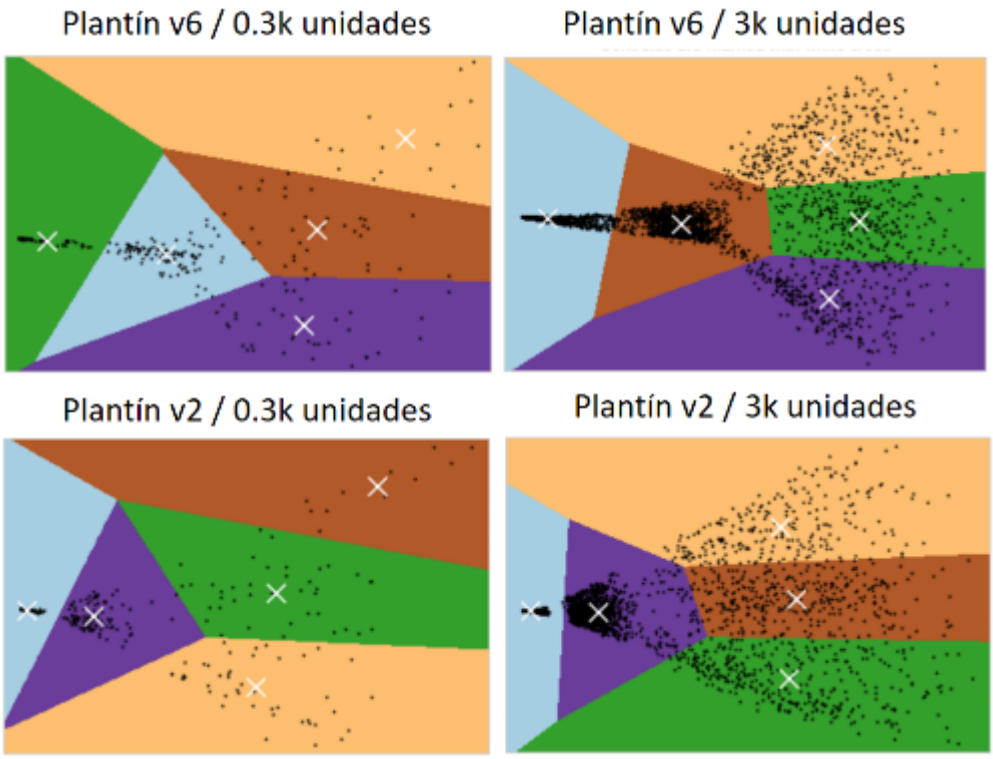


Fig. 31. Resultados de agrupación por medio del K-means teniendo en cuenta dos *datasets* diferenciados por la cantidad de plantines, 300 y 3000 plantines; cien y mil de cada clase, respectivamente. Fuente: elaboración propia.

Una vez obtenidos los PC1 y PC2, se eligen los modelos de plantines virtuales que mejor diferencien las clases de calidad decrecimiento (p. ej., aquellos que minimizan las zonas de intersección entre clases) para entrenar al K-means. Según los resultados (ver Fig. 30) se eligen las versiones 2 y 6 (además la versión 6 genera plantines virtuales que se asemejan más a los plantines reales). El número de imágenes de plantines es una variable relevante en el entrenaminto del K-means (300 y 3000 imágenes de plantines sintéticos). Donde las regiones apiladas de la derecha contienen a los plantines virtuales buenos, la región del centro contiene a los plantiens virtuales regualres y la región de la izquierda contiene a los plantines virtuales malos.

A fin de determinar la precisión del entrenamiento y testeo se usa el 10% de las imágenes. En la Fig. 32 se muestran los resultados del entrenamiento con plantines sintéticos versión 2; mientras que en la Fig. 33, para la versión 6.

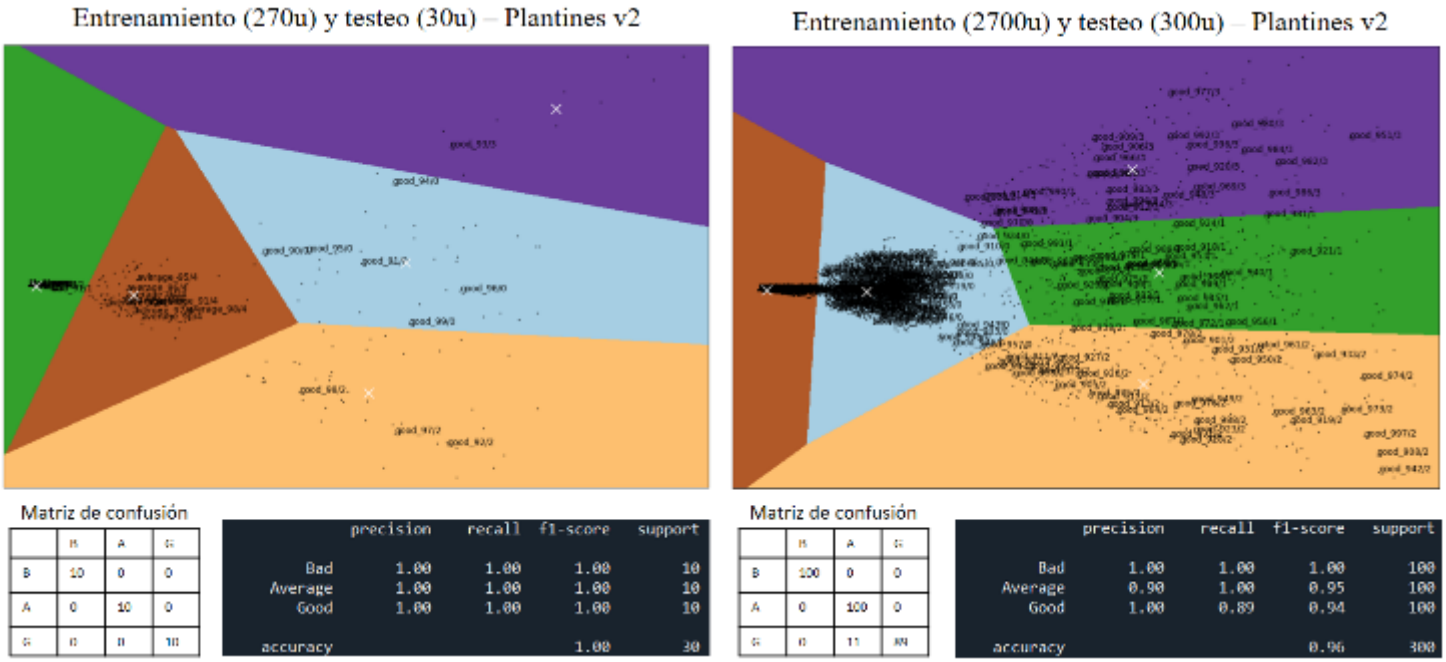


Fig. 32. Entrenamiento y testeo considerando 2 *datasets* con plantines de versión 2 y con diferente cantidad de imágenes. A la izquierda 270 imágenes de plantines sintéticos, y 30 imágenes para el testeo. En los resultados de la derecha se hace uso de una cantidad mayor de plantines sintéticos. Fuente: elaboración propia.

Por otro lado, para la validación se consideran 48 muestras obtenidas de plantines reales clasificados según la Tabla 1 en la Sección 2.2.3 (ver Anexos VI). Los resultados para el modelo k-means considerando plantines sintéticos versión 2 se muestra en la Fig. 34; mientras que que los resultados con plantines sintéticos versión 6, en la Fig. 35.

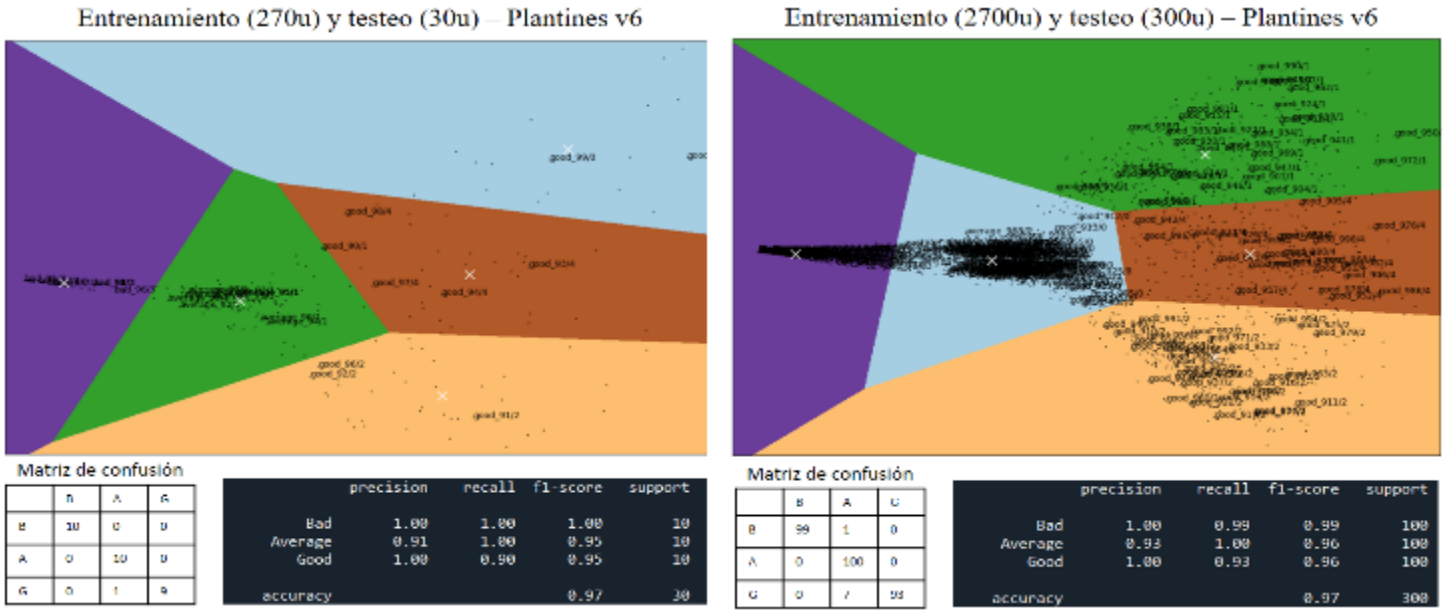


Fig. 33. Entrenamiento y testeo considerando 2 *datasets* con plantines de versión 6 y con diferente cantidad de imágenes. A la izquierda 270 imágenes de plantines sintéticos, y 30 imágenes para el testeo. En los resultados de la derecha se hace uso de una cantidad mayor de plantines sintéticos. Fuente: elaboración propia.

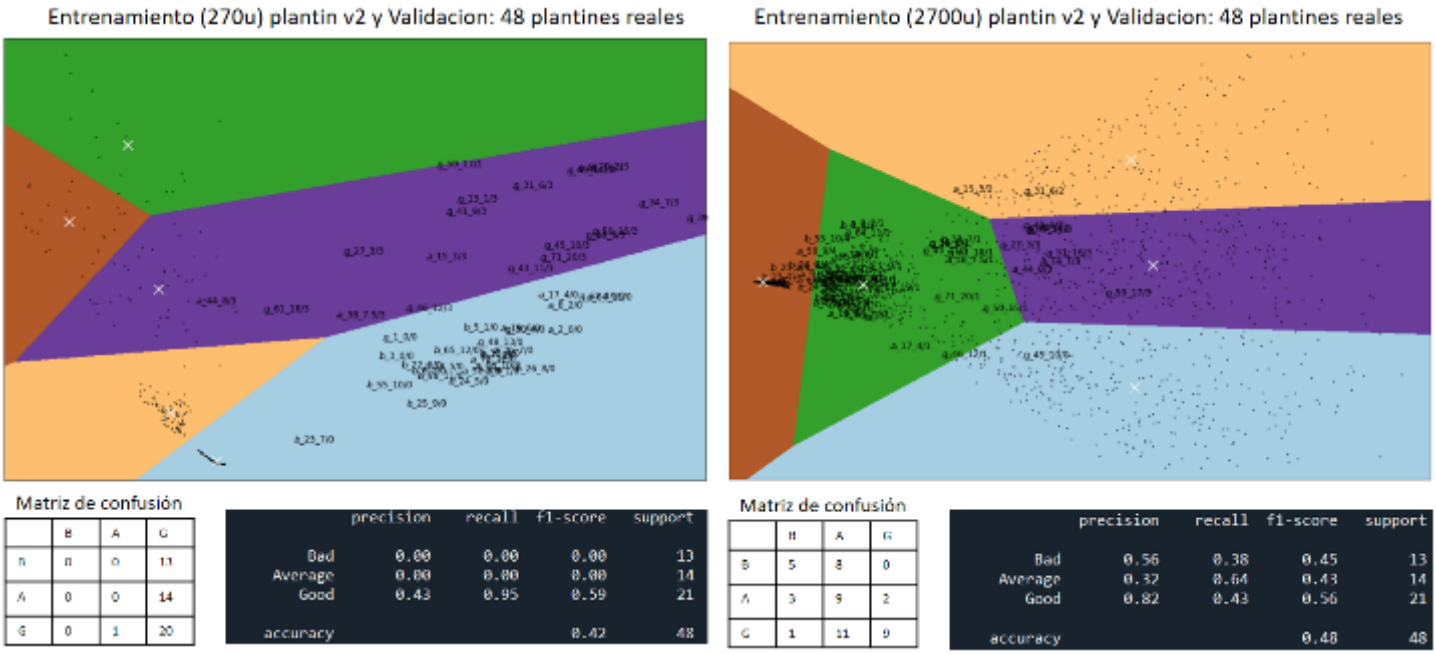


Fig. 34. Se muestran los resultados de la validación considerando plantines sintéticos versión 2. En la izquierda 270 imágenes, en la deracha 2700. Fuente Elaboración propia.

El programa usado para el entrenamiento, testeo y validación se puede encontrar en Anexos VII.

4.3.1. Entrenamiento del modelo basado en VGG16

Para el entrenamiento del modelo VGG16 se consideran las tres versiones de plantines sintéticos (2, 5 y 6) que mejor respuesta de agrupación han tenido según el K-means. En

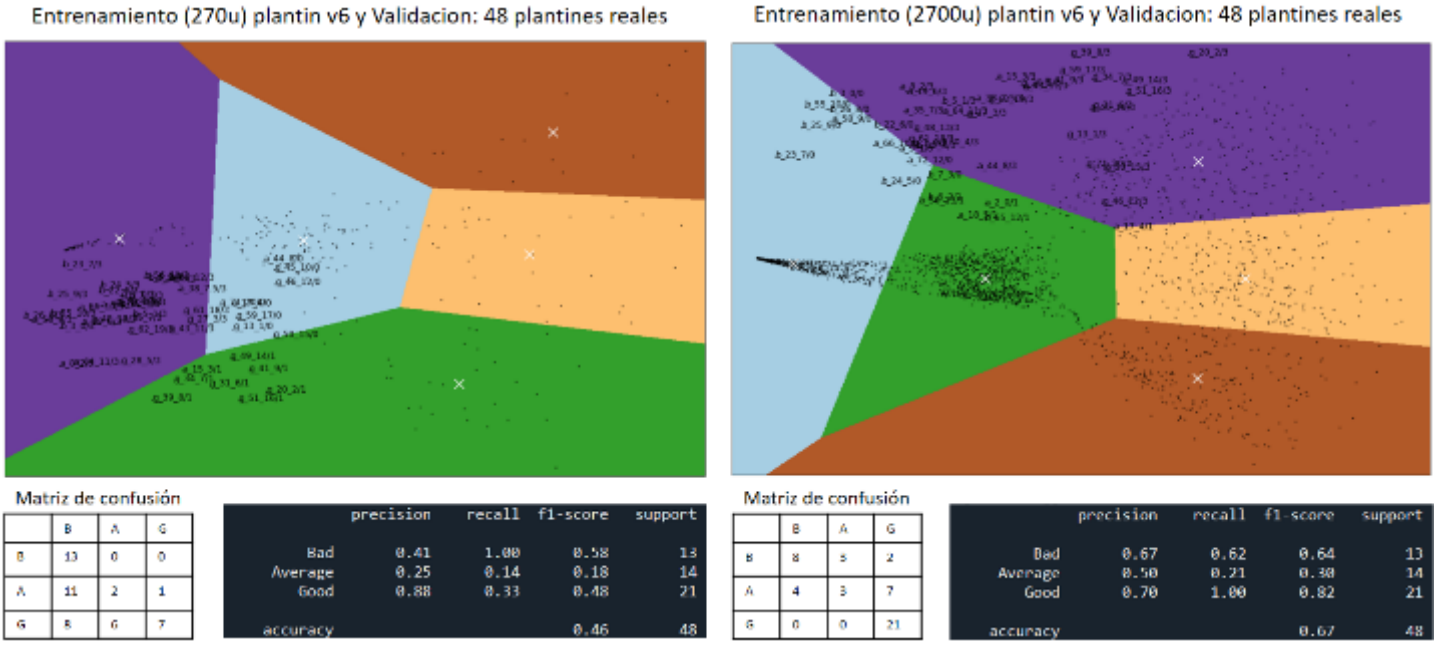


Fig. 35. Se muestran los resultados de la validación considerando plantines sintéticos versión 6. En la izquierda 270 imágenes, en la deracha 2700. Fuente Elaboración propia.

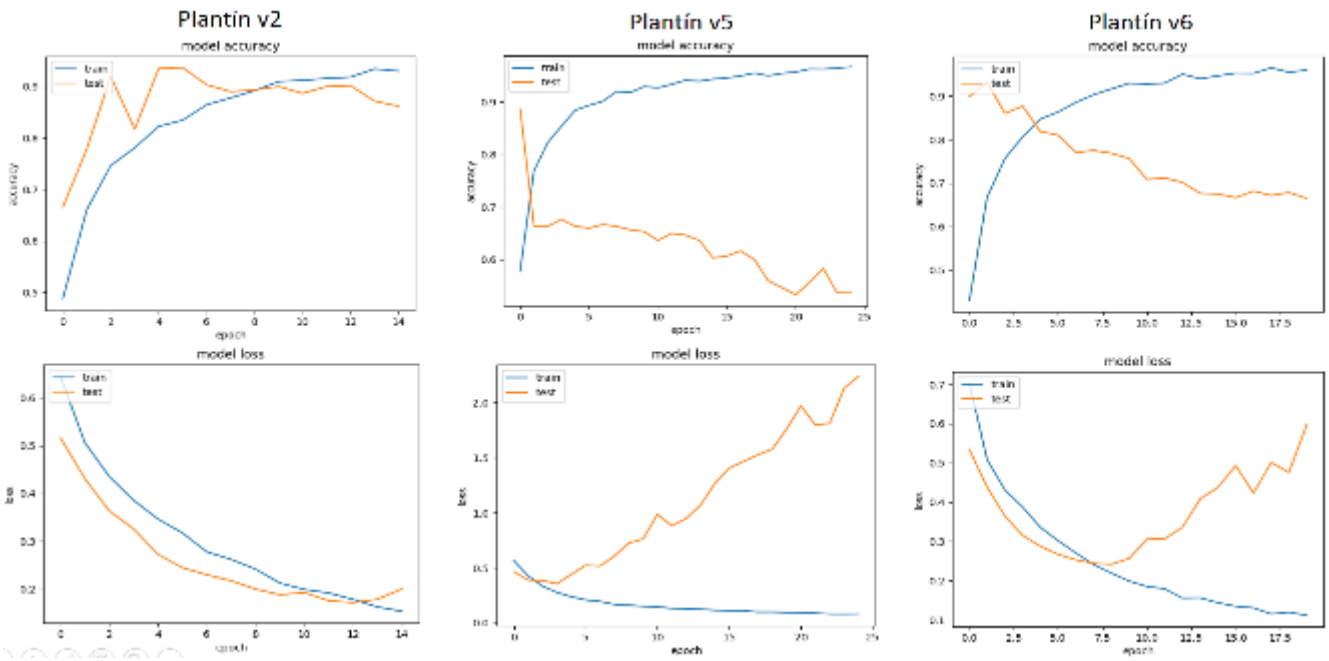


Fig. 36. Resultado de entrenamiento del modelo VGG16 para tres versiones representativas de los plantines sintéticos. Fuente Elaboración propia.

la Fig. 36 se muestran los resultados de entrenamiento y testeo, según los siguientes hiperparámetros:

- Razón de aprendizaje: 0.001, la cantidad de variación de los pesos cada vez que van a actualizarse. Debe ser un valor muy pequeño para asegurar un entrenamiento convergente (el error tienda a un mínimo).
- Épocas: la cantidad de veces que se entrena con el *dataset* completo. Varía según el *dataset*. Plantín v2: 14; Plantín v5: 25; Plantín v6: 20 (Fig. 32). Estos valores son arbitrarios, y se han obtenido heurísticamente.
- BS (*BachSize*): 50, se realiza un entrenamiento por *batches* (grupos). Valor obtenido heurísticamente.
- Tamaño de conjunto de testeo: 20% del total de data en el *dataset.* Obtenido heurísticamente.
- Objeto de aumento de data: que permite aplicar transformaciones básicas a las imágenes del *dataset.* Es un módulo del TensorFlow v2.1 llamado *ImageDataGenerator*. Valores de parámetros obtenidos heurísticamente.
    - *featurewise_center: True*
    - *featurewise_std_normalization: True*
    - *rotation_range:* 20

Matriz de confusión

	B	A	G
B	7	2	4
A	3	1	10
G	0	1	20

	precision	recall	f1-score	support
Bad	0.70	0.54	0.61	13
Average	0.25	0.07	0.11	14
Good	0.59	0.95	0.73	21
accuracy			0.58	48

Matriz de confusión

	B	A	G
B	10	1	2
A	8	3	3
G	2	2	17

	precision	recall	f1-score	support
Bad	0.53	0.77	0.62	13
Average	0.50	0.14	0.22	14
Good	0.76	0.90	0.83	21
accuracy			0.65	48

Fig. 37. Resultado de la validación del modelo VGG16 usando las dos versiones de plantines sintéticos con mejores resultados (v2: resultado tabla superior, v6 resultado tabla inferior). Fuente Elaboración propia.

  - *width_shift_range*: 0.2
  - *height_shift_range:* 0.2
  - *horizontal_flip: True*
- *Decay*: se considera la división: razón de aprendizaje/épocas, y permite complementar a la razón de aprendizaje en el algoritmo optimizador de aprendizaje Adam.

Como se menciona líneas arriba el algoritmo de optimización Adam (método estocástico basado en gradiente descendiente) incluído en la librería TensorFlow v2.1 se usa para disminuir el tiempo de entrenamiento del modelo VGG16, lo cual quiere decir, ayuda a llevar a cero la función de pérdida: *BinaryCrossEntropy,* incluída también en el TensorFlow v2.1.

El programa de entrenamieto, testeo y validación del modelo VGG16 se puede encontrar en los Anexos VIII.

4.3.2. Entrenamiento y validación del modelo Yolov3

El entrenamiento del modelo Yolov3 usa el *framework Darknet*, que requiere que las imágenes estén etiquetadas (ver Sección 4.1.3). El código de entrenamiento, testeo y validación de este modelo, puede encontrarse en los Anexos IX.

El entrenamiento debe realizarse con tarjetas de video o GPU (*Graphics Processing Unit*) debido a que el Yolov3 es una red neuronal de gran profundidad considerable (ver Sección 4.2.2) que consume grandes recursos computacionales. Por tal motivo se usa la

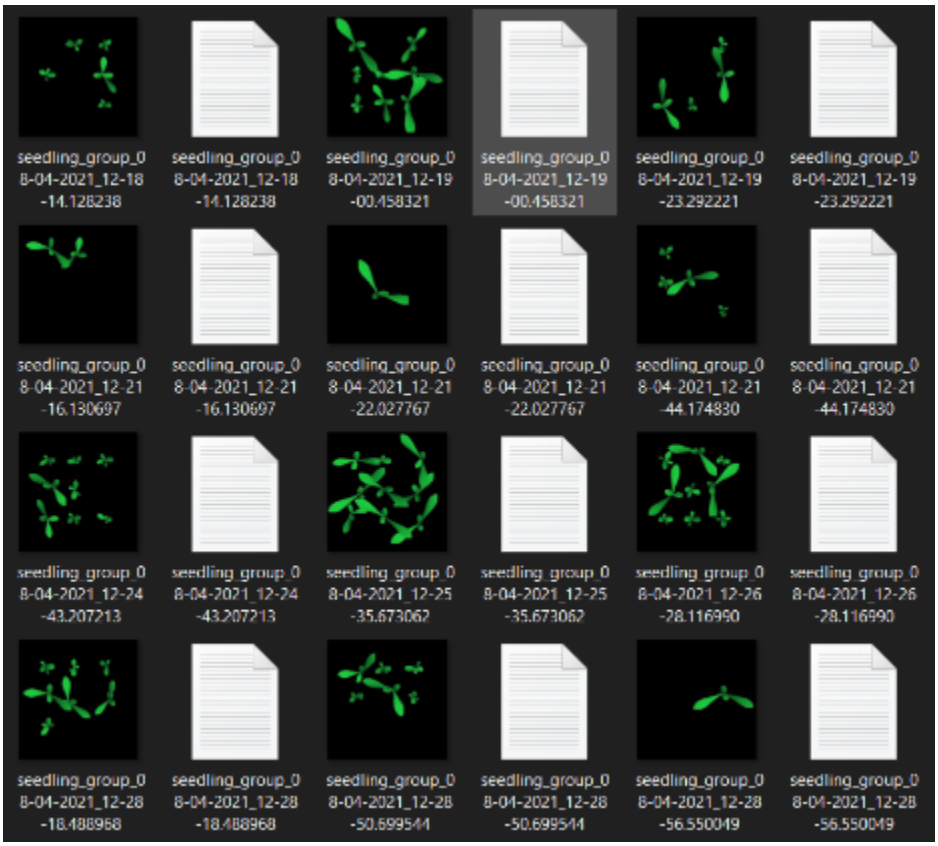


Fig. 38. *Dataset* de 1000 imágenes de plantines sintéticos v6 para el entrenamiento del modelo Yolov3 basado en el Darknet. Cada imagen tiene su par en .txt donde se indica la clase, centroide (x,y), ancho y alto de cada plantín. Se consideran solo plantines regulares y buenos. Fuente: Elaboración propia.

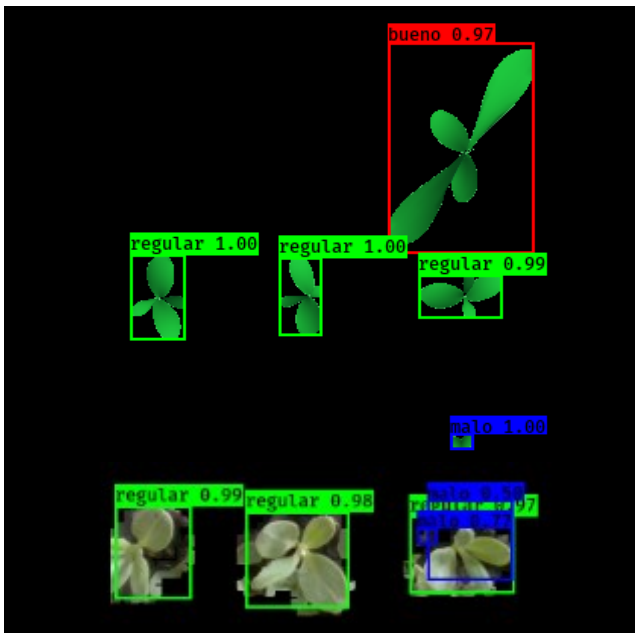


Fig. 39. Cuando se consideran plantines de mala calidad como parte del *dataset* al momento de validar el modelo con plantines reales, la tendencia a equivocar manchas pequeñas con plantines de mala calidad es muy recurrente. Fuente Elaboración propia.

plataforma *Google Colaboratory*. El entrenamiento dura 8 horas, teniendo en cuenta 4000 batches (grupos de imágenes) y 1000 imágenes de grupos de cinco plantines en promedio.

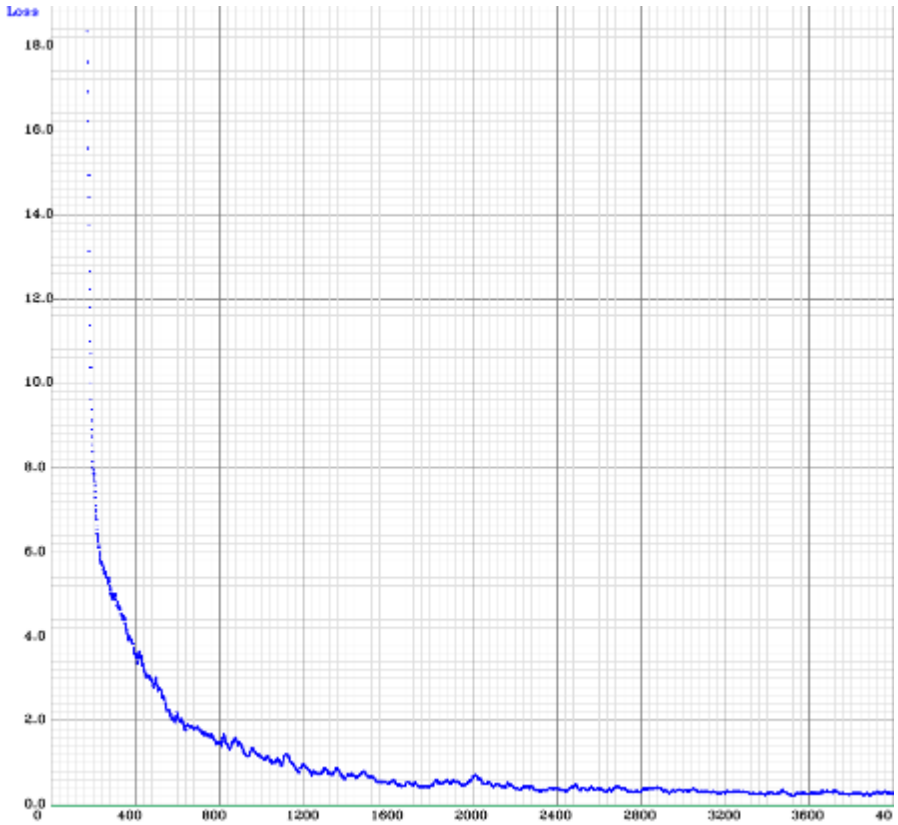


Fig. 40. Evolución de la función de pérdida durante el entrenamiento del Yolov3. Se obtiene un error de pérdida promedio de 0.2957%. Fuente Elaboración propia.

Por lo tanto, el número aproximado de plantines en total es de 5000 (Fig. 38). Es importante notar que, para el entrenamiento del Yolov3, solo se trabaja con plantines regulares y buenos. Dado que los plantines de mala calidad pueden llegar a ser muy pequeños y ser confudidos con manchas de las imágenes de plantines reales (Fig. 39). A partir de ahora la predicción de detección del plantines, por parte del modelo, está dada por las coordenadas de la centroide, el ancho y alto del plantín. En la Fig. 40 se observ la evoluación de la función de pérdida, logrando un 0.2957% de error.

En la Fig. 41 se muestran los resultados de las tareas de detección llevada a cabo por el modelo Yolov3 entrenado con el *dataset* descrito líneas arriba. Habiendo entrenado al modelo teniendo en cuenta un *dataset* compuesto por dos clases (Bueno y regular), se observa que los plantines son clasificados generalmente como Regulares.

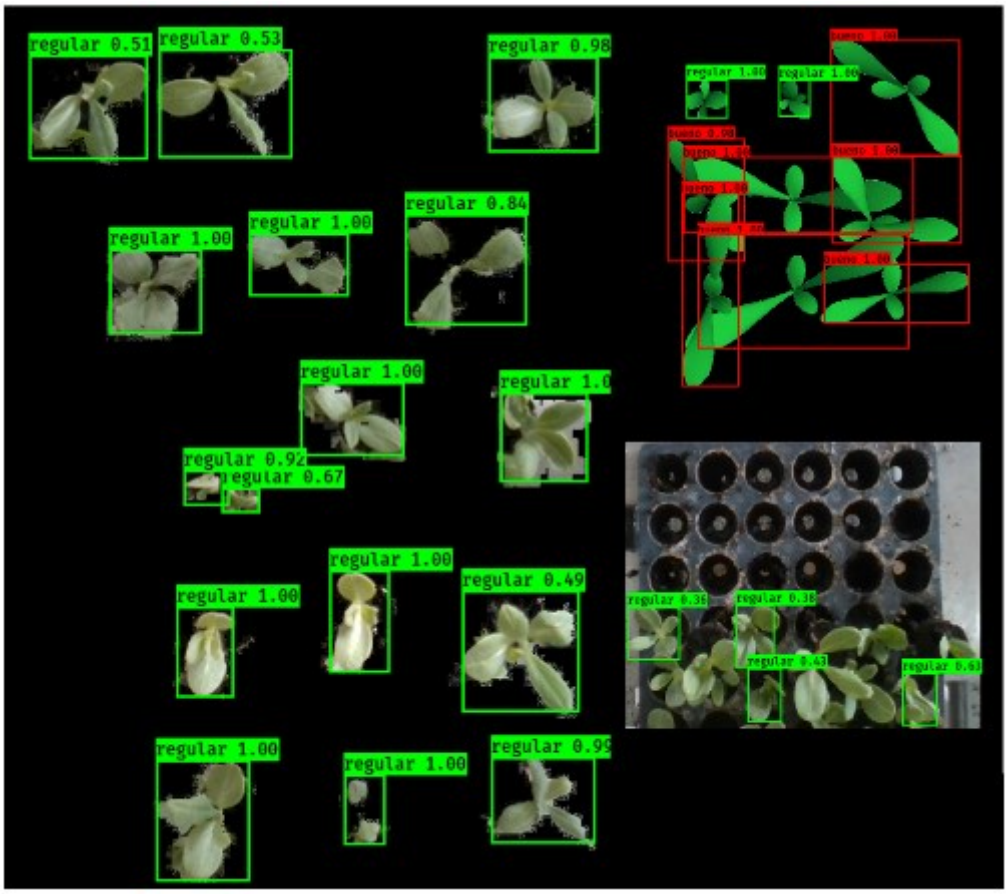


Fig. 41. Validaciones del modelo Yolov3 entrenado con plantines sintéticos versión 6. El modelo trabaja correctamente para la detección de los plantines reales, incluyendo plantines en sus correspondientes bandejas; no sucede lo mismo para la clasificación de éstos, el modelo los detecta a todos como de clase Regular. Fuente Elaboración propia.

A continuación se considera solo la predicción de detección de plantines que entrega el Yolov3 (t_x, t_y, t_w, t_h) para desarrollar la discriminación por umbral (Fig. 29). Para tal fin se inicia con la detección de las regiones de ubicación de los plantines elevados por las agujas del robot. En la Fig. 42 se muestran las tres regiones establecidas por medio de las centroides de 72 plantines muestreados en el laboratorio (Ver programa para la obtención de centroides en Anexos X, y lista de plantines muestreados en Anexos XI) y procesados, luego, por el Yolov3. También se observa que cada región cuenta con una subdivisión; es el resultado de levantar y capturar la imagen de los 6 plantines a lo largo de la fila de la bandeja. Los plantines se ubican dentro de los límites se los clasifica según el área y el valor 0 (no hay plantín), 1 (platín malo), 2 (regular) y 3 (bueno), los cuales se almacena en una matriz resultante. Puede haber más de 1 detección en la misma posición (grupo); en este caso se consideran la detección de mayor calidad. Entonces, según la matriz de resultados de la Fig. 42, el índice 0 de la matriz, devuelve un plantín 0 (vacío), índice 1, un plantín de calidad regular e índice 2, uno de calidad buena. Los límites indicados en la Fig. 42 sirven para establecer los umbrales tanto en X como en Y a fin de discriminar

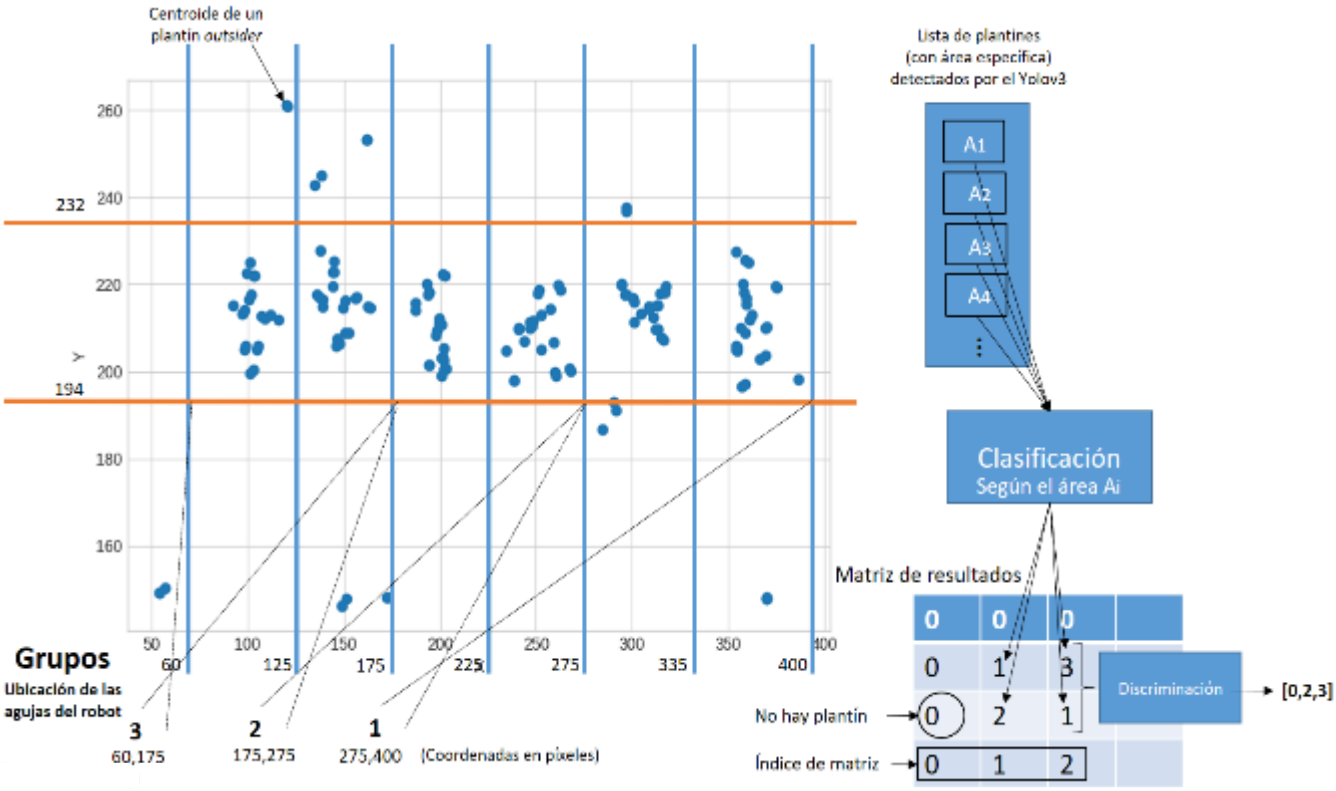


Fig. 42. Consideraciones para la clasificación de plantines. Los puntos representan la centroide de plantines, que según las muestras recolectadas en el laboratorio, estos plantines debido a las agujas del robot que los elevan (observar grupos del 1 al 3), se concentran mayormente en las regiones mostradas en la imagen. Los "plantines" fuera de dichas regiones son en realidad manchas o plantines volteados que son descartados en el análisis. Fuente Elaboración propia.

detecciones de “plantín” que o bien puede ser ruido en la imagen o plantines volteados que no se elevaron correctamente con las agujas del robot. Estos se descartan. Consiguientemente, el área del plantín detectado es calculado multiplicando el ancho por el alto ($t_w \times t_h$) y permite diferenciar la calidad de acuerdo a umbrales de área. Con las muestras totales de imágenes de plantines sintéticos obtenidas, las cuales suman 72 (Ver Anexos XII), y clasificadas según los parámetros de calidad de la Agroindustrial Agrogénesis, Tabla 1 (Sección 2.2.3), se establecen los umbrales de área, haciendo uso de la desviación estándar (Fig. 43) de la clase 1 (malo) y 3 (bueno):

- Menor: ≤ 2780
- Regular: > 2780 y < 4920
- Bueno: ≥ 4920

Con los umbrales de área se clasifican los plantines y como se observa en la Fig. 42, el resultado de la clasificación se almacena en “una matriz de resultados” distribuidos según la posición del plantín en las regiones correspondientes a las agujas. El índice 0 de dicha

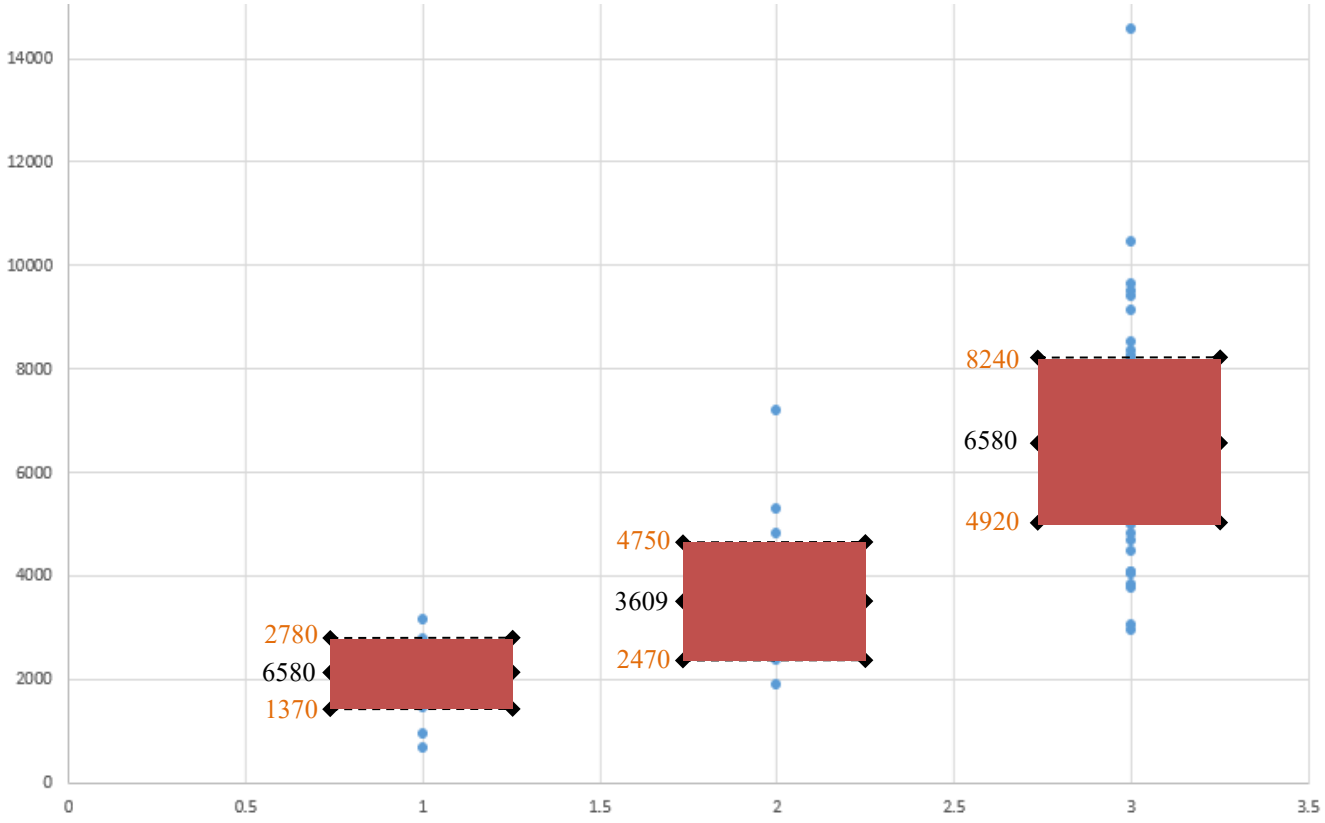


Fig. 43. Distribución del área calculada con el ancho y alto entregados por el modelo Yolov3 luego de procesar 72 imágenes de plantines capturadas en el sistema robotizado del laboratorio. Se muestra también el promedio y la incertidumbre (obtenida a través de la desviación estándar) para las calidades: 1 (mala), 2 (regular) y 3 (buena). Fuente: Elaboración propia.

matriz contiene la clase de los plantines detectados en la tercera región (Fig. 42), (la cuenta empieza desde el extremo izquierdo); en el índice 1, se ubica la clase de los plantines detectados correspondientes a la región de la segunda aguja, y así sucesivamente. Este aspecto se implementa debido a que el modelo puede que detecte más de un plantín en la misma región a causa de ruido como manchas verdes que suelen aparecer. Este aspecto es poco frecuente, pero "la matriz de resultados" ayuda a ordenar las predicciones y a discriminar considerando el plantín de mayor calidad en cada región (que se traduce en mayor área), dado que las manchas suelen ser pequeñas.

Finalmente, la métrica de desempeño (intersección sobre la unión, IoU) del Yolov3 para la parte de predicción de detección de objeto (t_x, t_y, t_w, t_h) como se indica en la Sección 2.2.5.3., ayuda a reconocer la utilidad del Yolov3 para detectar los plantines en las imágenes. En la Fig. 44 se observan los resultados luego de procesar 47 plantines. Vale recalcar que hubo un plantín que no fue detectado por el modelo. Al ser un *outsider*, se puede dejar de lado para el análisis del desempeño promedio del Yolov3 para la detección de plantines, el cual es 57%. En el programa de los Anexos IX se calcula esta métrica por cada plantín que el modelo Yolov3 va detectando.

Es de relevancia recalcar que por los resultados obtenidos con el modelo Yolov3, se considera hacer uso de éste exclusivamente para la tarea de detección de plantín. Tal y

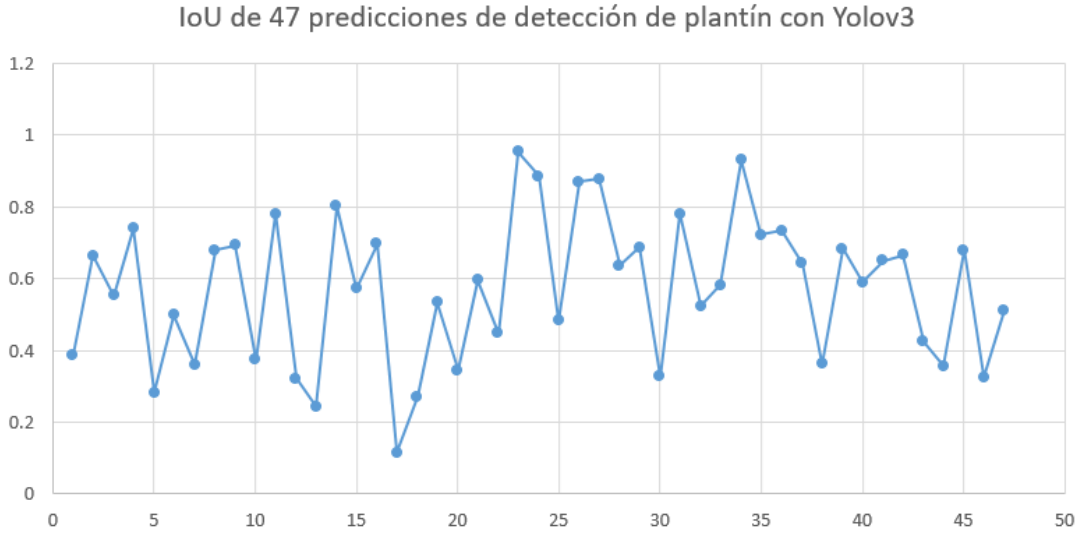


Fig. 44. Métrica de desempeño de detección de plantines del modelo Yolov3: intersección en la unión (IoU), para una población de 47 plantines. El promedio es 0.57. Fuente: Elaboración propia.

Matriz de confusión

	B	A	G
B	10	1	2
A	8	2	4
G	1	1	19

	precision	recall	f1-score	support
Bad	0.75	0.92	0.83	13
Average	0.57	0.62	0.59	13
Good	0.94	0.76	0.84	21
accuracy			0.77	47

Fig. 45. Resultados de la validación del modelo de discriminación por umbral. Fuente Elaboración propia.

como se plantea en la Fig. 29 (Sección 4.2.2). La parte de la predicción de la clasificación del plantín, siendo por un lado a través del K-means y el VGG16 en este caso se opta por la discriminación por umbral, la cual viene a nombrarse como el modelo 3 (Fig. 29). El desempeño del Yolov3 con las discriminaciones por umbral para la detección y clasificación de plantines de alcachofa según su calidad se muestra en la Fig. 45.

4.4. Correlación de predicciones y estimaciones

Considerando la numeración de los modelos para la clasificación de la calidad de plantines expuesto en la Fig. 29., se muestra en la Tabla 8 los mejores resultados obtenidos durante la validación. Previamente el modelo Yolov3 realiza la detección del plantín, que como se explicó en la Sección 4.3.2., se consigue un 57% de IoU. Esto último influye también directamente en los resultados de las predicciones de clasificación.

Tabla 8. Resumen comparativo del desempeño para clasificación de calidad de plantín.

N° Modelo	F1-score			Exactitud	TEPD* (miliseg.)	TEPC** (miliseg.)	T*** (miliseg.)
	Bueno	Regular	Malo				
Modelo 1 Kmeans	0.82	0.30	0.64	0.67	≈ 237	≈29000	≈ 29237
Modelo 2 VGG16	0.83	0.22	0.62	0.65	≈ 237	≈ 407	≈ 644
Modelo 3 Discriminación por umbral	0.83	0.59	0.84	0.77	≈ 237	< 1	≈ 237

*TEPD: tiempo estimado de predicción de detección haciendo uso del Yolov3 para 47 plantines, se calcula unos 5 milisegundos/plantín.
**TEPC: tiempo estimado de predicción de clasificación de los modelos 1 al 3 para aprox. 48 plantines, se calcula unos 8 milisegundos/plantín para el M2 y 604 milisegundos/plantín para el M1. El M3 tiene un valor muy por debajo de los milisegundos como para considerarlo.
***TT: tiempo total estimado de predicciones: TEPD + TEPC

A fin de obtener la correlación final, se realizó un último experimento ingresando 10 bandejas de plantines de alcachofa al robot clasificador, con las cuales se pudieron

clasificar 619 plantines. Dichas bandejas fueron entregadas por un vivero industrial, habiendo clasificado previamente los plantines por sus jornaleros. La tabla de los resultados de las clasificaciones se encuentra en los Anexos XV. En la Fig. 42, 43 y 44 se observa gráficamente la correlación de los modelos.

En la Tabla 9 se detallan los valores de R (correlación de Pierson) y del valor-p (significancia de la estadística). Los programas de cada modelo incluyen también el cálculo de la correlación (Anexos VII, VIII y IX, respectivamente).

Tabla 9. Resultados de correlación: Modelos propuestos *vs* Agrogénesis

	VGG 16	**K-means**	**D. por umbral**
R	0.67	0.71	0.85
p	2.51×10^{-7}	1.91×10^{-8}	5.22×10^{-14}

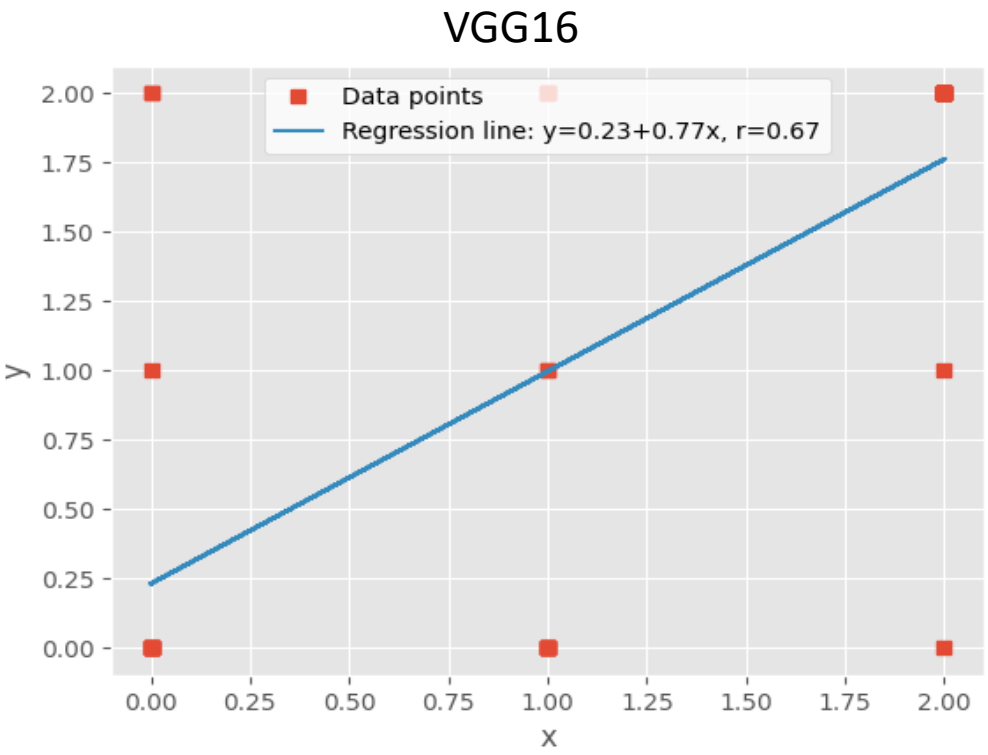


Fig. 46. Resultado de la correlación entre el modelo VGG16 con la clasificación hecha por los jornaleros. Fuente: Elaboración propia.

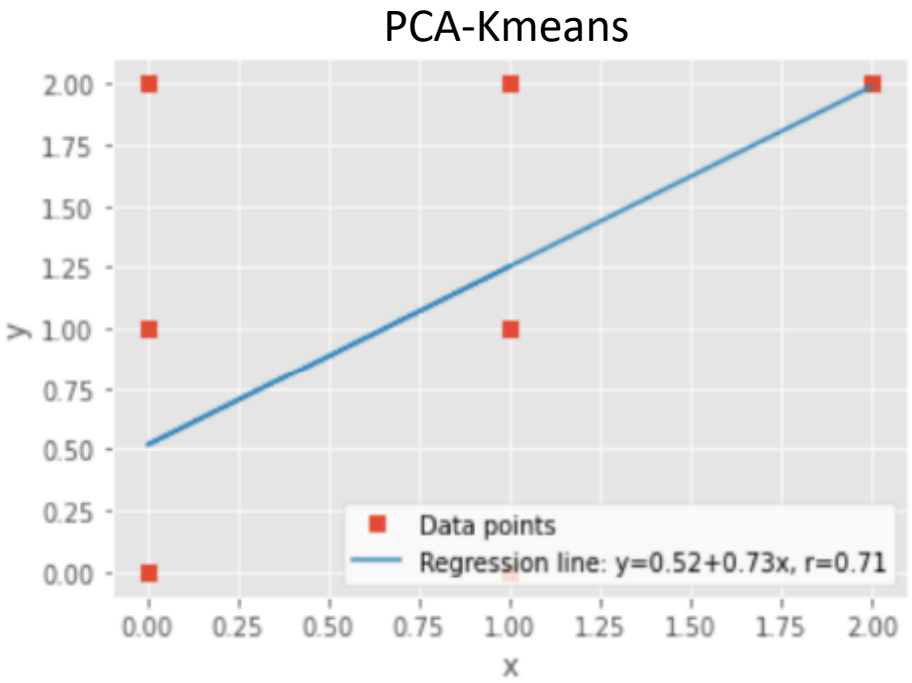


Fig. 47. Resultado de la correlación entre el modelo PCA-kmenas con la clasificación hecha por los jornaleros. Fuente: Elaboración propia.

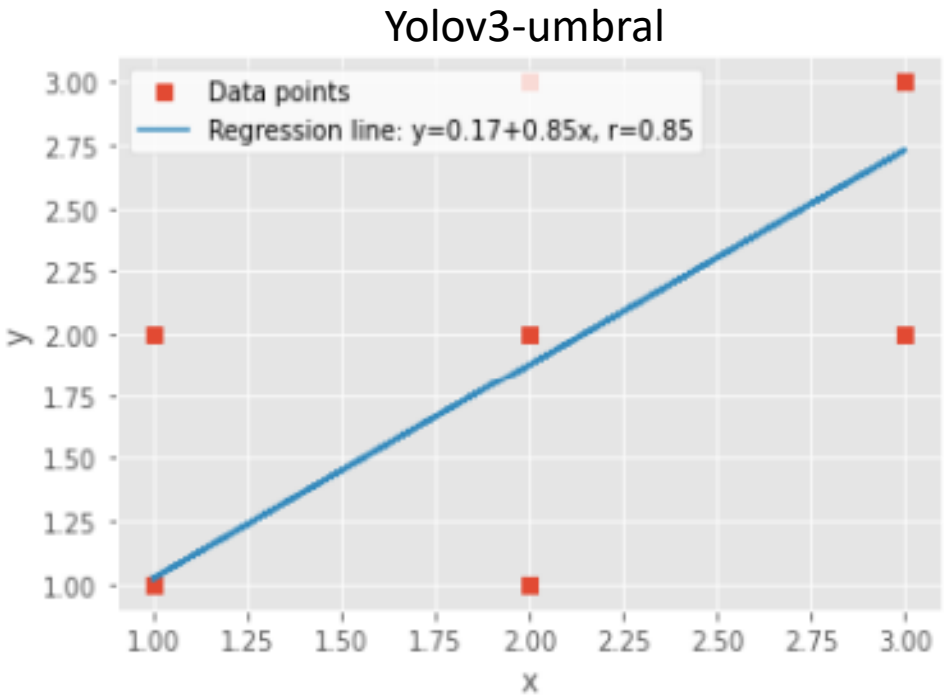


Fig. 48. Resultado de la correlación entre el modelo Yolov3-umbral con la clasificación hecha por los jornaleros. Fuente: Elaboración propia.

4.5. Puesta en marcha de la estrategia para la clasificación de plantines

En la Fig. 49 se muestra el diagrama de la lógica principal del sistema de Visión por Computadora (SVC) desarrollado en este trabajo. El programa soporte del SVC primero crea la clase llamada "mqttModbusClient" (ver código en Anexo XIII) que permite manipular objetos de los dos clientes de comunicación que se usa en este trabajo: MQTT y MODBUS TCP. El "mqttModbusClient" maneja dos hilos (procesos independientes que corren en paralelo), uno para leer mensajes provenientes del bróker MQTT y el otro para los cambios en los registros MODBUS (Fig. 50) configurados para el PLC (Delta AS320–B). Finalmente, las predicciones de clasificación de plantines son escritos en: los registros del MODBUS configurados para ser leídos por el PLC, en el node-red (incluyendo imágenes de plantines detectados) y en la página web pública. Posteriormente, el PLC (controlador del robot) ejecuta los programas que permiten que el robot coja a los plantines previamente clasificados, los traslade y finalmente los deje

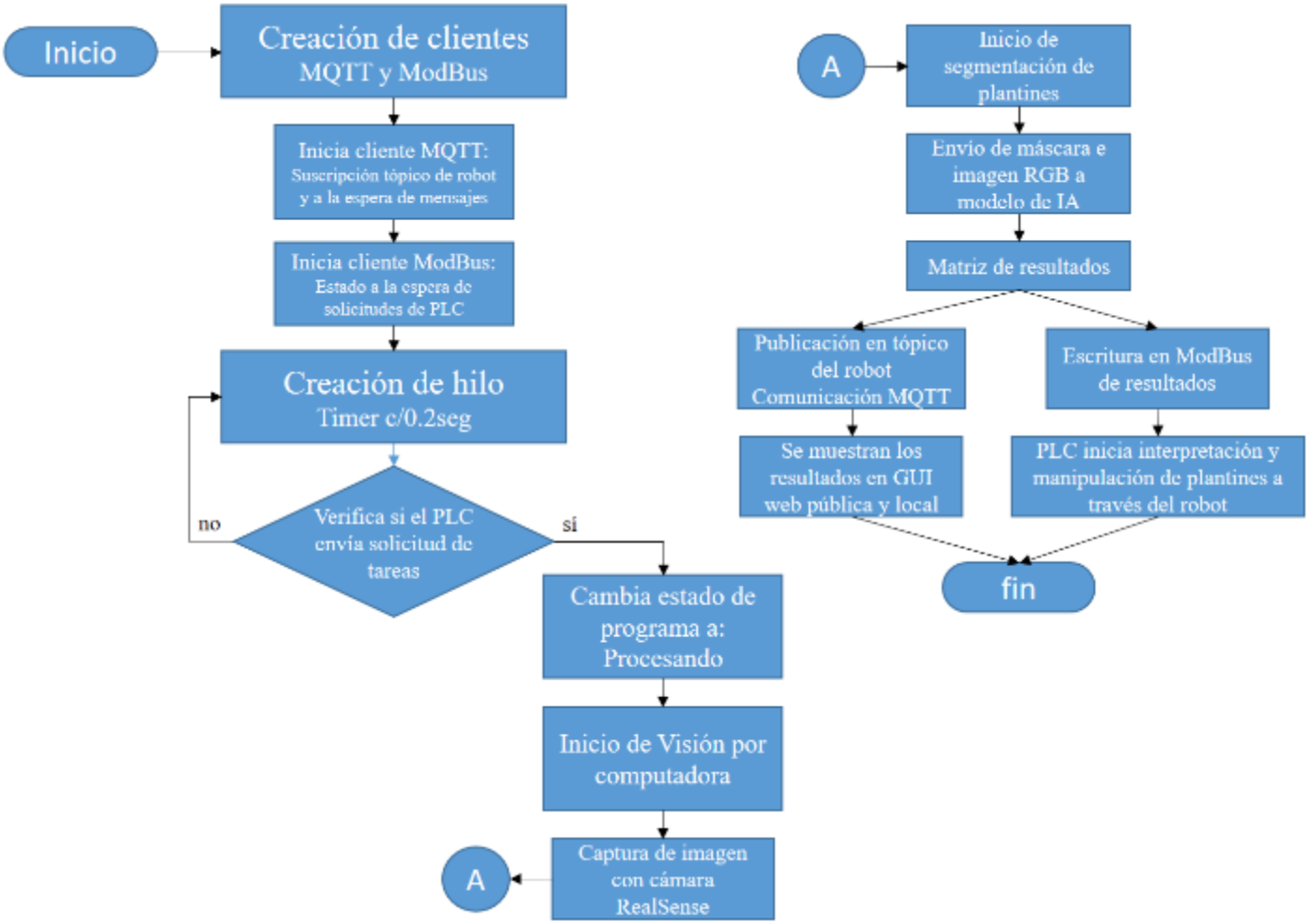


Fig. 49. Diagrama de flujo del programa gestor de visión por computadora. Fuente: Elaboración propia.

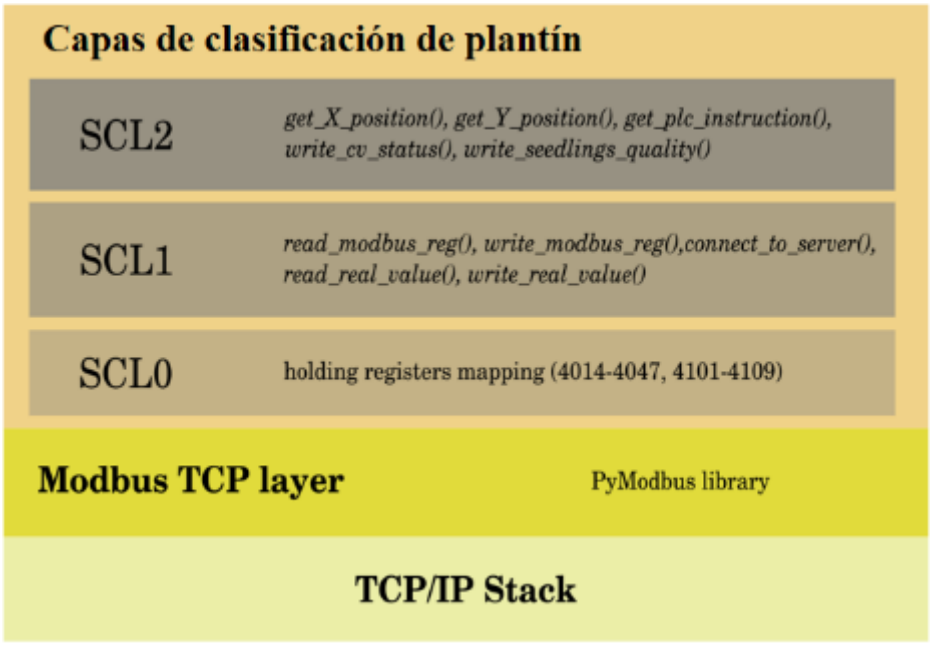


Fig. 50. Diagrama de capas de la comunicación MODBUS desarrollada para la clasificación del plantín. Fuente: elaboración propia

en sus correspondientes bandejas multi-celda. Para lograr ello, se implementa una comunicación MODBUS (librerías PyModbus) de escritura/lectura con registros de 16bits y las siguientes capas de comunicación TCP/IP (Fig. 50):

- SCL0: establece todos los registros orientados a realizar la automatización de la clasificación, es decir, para almacenar los datos del estado del robot como posición de efector final, bandejas, etc., así como de las predicciones, entre otros (Fig. 51).
- SCL1: codifica o decodifica los datos almacenados en los registros que son enviados tanto por el algoritmo de detección y clasificación de plantines, como por el mismo robot solicitando predicciones.
- SCL2: realiza (subcapa de mayor nivel) acciones específicas como obtener la posición del robot en milímetros, leer las instrucciones enviada por el PLC o enviar los resultados de la clasificación desde el gestor de visión por computadora al PLC.

Fig. 51. Registros establecidos para el MODBUS TCP a fin de poder comunicar al PLC con el programa gestor de visión por computadora. Fuente: Elaboración propia.

En la Tabla 10 se muestra una breve descripción de los registros involucrados en la comunicación entre el gestor de visión por computadora y el PLC.

Tabla 10. Lista de algunos registros usados para la comunicación con el MODBUS

Descipción de registro	**Ejemplos**	**Tipo de registro**	**Rango de dirección**
Contadores	Contadores	Unsigned integer (16-bit)	4014− 4021
Posición actual del robot	Posición de los ejes X, Y, posición del mecanismo de agujas de elevación de plantines, etc.	Float (32-bit)	4022− 4042
Estado de las garras para plantines	Estado de la garra 1, garra 2 y garra 3	Unsigned integer (16-bit)	4044
Alarmas	Código de alarma	Unsigned integer (16-bit)	4045
Registro del SVC	Procesamiento de la orden por parte del controlador del robot, estado del SVC (a la espera, procesando, fin de procesamiento, etc.), resultados de clasificación de plantines, etc.	Unsigned integer (16-bit)	4046, 4100− 4104

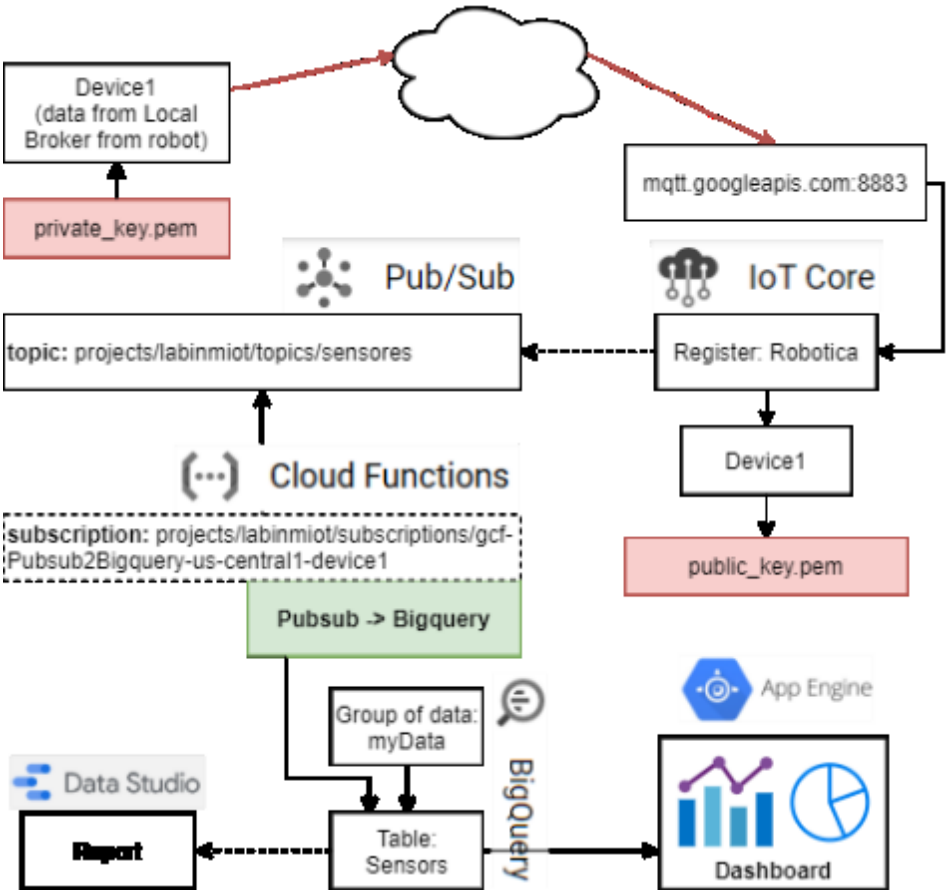


Fig. 53. Arquitectura de la nube. Los datos del robot (*Device 1*) comparten información con el módulo del *Google Cloud IoT* usando llaves públicas y privadas (capa de seguridad). Los datos son asociados a un tópico específico y transferidos al *Goolgle BigQuery* usando *Google Cloud Functions*. Finalmente la información del robot es mostrada en un Dasborad que leen la data del *BigQuery*. Fuente: Elaboración propia.

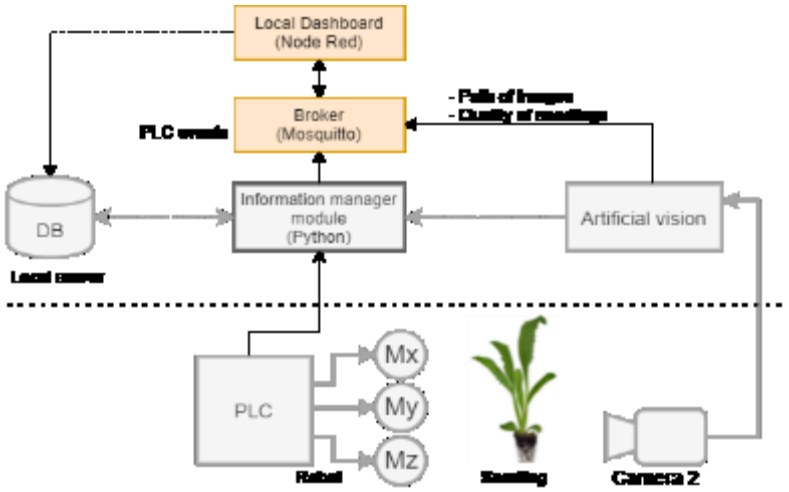


Fig. 52. Arquitecutura de software en la red local. Localmente se ubican el PLC y el programa gestor de visión por computadora (*information manager module*). Localmente se realizan las predicciones de calidad. Fuente: elaboración propia.

Las arquitecturas propuestas a nivel nube y local se muestran en las Fig. 52 y Fig. 53, respectivamente. Con respecto a la red local, en la Fig. 54 se muestra las páginas webs creadas por medio del Node-Red que tienen por finalidad visualizar las predicciones realizadas por cada plantín, así como también realizar algunas calibraciones de parámetros para el modelo de Discriminación por umbral. Se muestra también una ventana de configuración de parámetros y un botón de testeo (que reemplaza la orden del PLC) para realizar la predicción pertinente. Los programas del Node-red se pueden encontrar en los Anexos XIV.

Por el lado de la nube, en la Fig. 55 se muestra la página web pública mediante la cual se puede acceder remotamente con el fin de conocer el estado actual de la clasificación de los plantines. El funcionamiento de esta página web pública se basa primero en el uso de un servidor público MQTT que corre en Google IoT Core, a través del cual se configura dicho servicio, así como una función (Google Functions) para redireccionar la información que llega a los tópicos *Publish* (según protocolo MQTT) a una base de datos en Google BigQuery. Posteriormente, un aplicativo en Google AppEngine lee los datos de la base de datos en BigQuery y los muestra en las gráficas (Fig. 55). Se puede observar también en este dashboard la función de poder llevar la contabilidad de los plantines que entran a través de la bandeja alimentadora y cuántos se han clasificado y ubicado en su respectiva bandeja. Al lado derecho del dashborad se muestran el porcentaje de llenado de las bandejas, y en la parte inferior, la calidad de los plantines que acaban de ser clasificados.

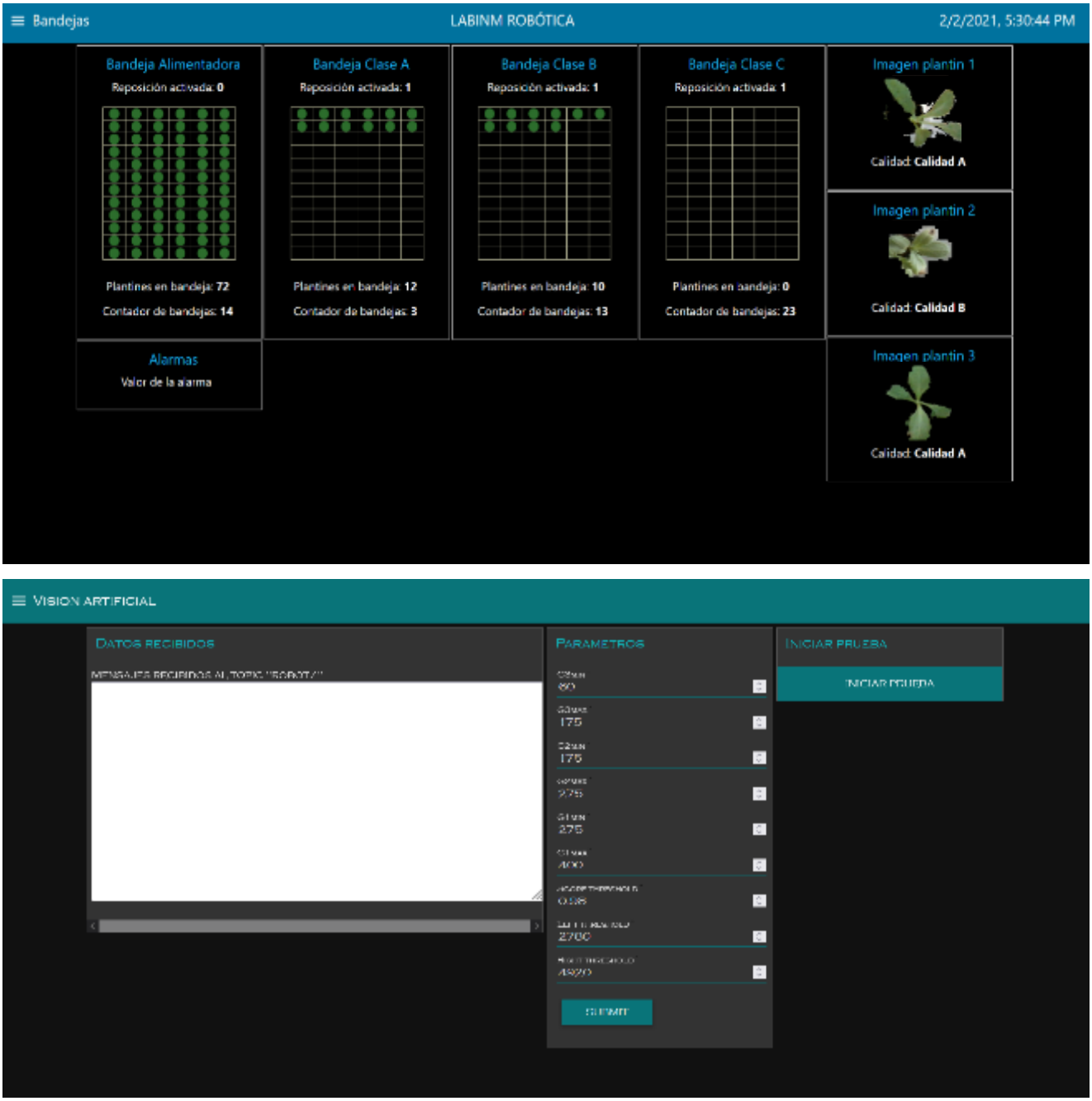


Fig. 54. Interfaces gráficas de usuario basadas en web mediante el servidor Node-Red usadas para la visualización de las predicciones de plantines. Arriba la ventana Dashboard/Bandejas, para el conteo de plantines con la bandeja de alimentación. Abajo la pantalla de Parámetros IA para realizar calibración de parámetros del modelo Discriminación por umbral, y un botón de prueba para correr el modelo de IA sin necesidad de recibir órdenes del PLC para tal fin. Fuente: Elaboración propia.

Fig. 55. Se cuenta también con una interfaz gráfica en web pública como complemento del Node
acceder remotamente a los datos recopilados en el proceso, los cuales pueden servir para hacer *Busines*

Para el programa gestor de visión por computadora (Fig. 49) y las interfaces en Node-Red y *Google Cloud Platform* basadas en web (Fig. 54 y 55), las versiones usadas de los módulos, librerías y servidores principales son las siguientes:

3. Broker MQTT: Mosquitto 2.0.11 para windows
4. Node-Red: v1.1.3 que hace uso de librerías extras: node-red-dashboard v2.23.3, node-red-contrib-image-tools 2.0.1, node-red-contrib-image-tools 2.0.1 y node-red-contrib-fs-ops 1.6.0
5. Node.js: v.12.18.3
6. Python: v3.8.3
7. Pyrealsense2: v2.48.0.3381
8. PyModbus: v2.5.2
9. Paho MQTT: v1.5.1
10. Cv2: opencv-python v4.5.2.54
11. Sistema Operativo: Windows 10

Otro aspecto relevante a considerar son los archivos necesarios para cargar el modelo Yolov3. Se consideran los tres siguientes (se pueden encontrar en la carpeta PROGRAMAS del CD adjunto):

- yolov3_testing.cfg: arquitectura del modelo yolov3.
- yolov3_training_last.weights: pesos del modelo yolov3 que son cargados por medio de la librería NumPy.
- classes.txt: clases de plantines considerados al momento del entrenamiento (regular y bueno).

El programa de entrenamiento del Yolov3 se puede encontrar en la siguiente cuenta de github: https://github.com/3rickfs.

CAPÍTULO V

DISCUSIÓN

"El hombre más poderoso es aquel que es totalmente dueño de sí mismo"

Aristóteles.

CAPÍTULO V: DISCUSIÓN

La discusión del presente trabajo se divide en las siguientes cinco partes:

- Elaboración del *dataset* de imágenes de plantines (virtuales y reales) de alcachofa.
- Modelos de inteligencia artificial (ML y DL) para la detección y clasificación de plantines de alcachofa.
- Entrenamiento y validación de los modelos de IA propuestos.
- Correlación existente entre las predicciones de clasificación obtenidas en este trabajo y la clasificación estándar de plantines usadas en la agroindustria AgroGénesis.
- Integración de diferentes módulos para obtener el sistema de visión por computadora que permite que un robot realice las acciones de coger, transportar y dejar plantines en sus correspondientes bandejas según calidad de crecimiento.

La síntesis de plantines virtuales de alcachofa mediante un lenguaje híbrido C+L permiten obtener un modelo (ver ec. 9) de crecimiento de plantín (Fig. 17) con un alto grado de semejanza a unos reales.

El uso de modelos paramétricos facilita sintetizar el crecimiento del plantin (en forma y color, según crecimiento) y capturar imágenes del mismo desde cualquier ángulo, registrando digitalmente el desarrollo del plantin tal que sirva para futuros análisis y entrenamiento de modelos de DL. El software LStudio, usado en este trabajo, presenta gran flexibilidad para el modelamiento de la forma de las hojas, lo que demuestra su gran potencial para hacer síntesis de imágenes de diferentes plantas y cultivos (flor, hoja, planta, u organismo en general).

La complicación que se tiene con el LStudio es que la versión en Windows 10 presenta un *bug* que no le permite renderizar los modelos en C+L a través de comandos sino solamente por medio de la interfaz gráfica de usuario (GUI). Según el desarrollador que participó con Ubbens et. al. (2018), por medio de *e-mails*, informó que la versión para MacOS es la que se mantiene en constante desarrollo y en la que dicho *bug* no está presente. Al no contar con una MacOS, se propone como alternativa usar la tecnología

RPA (*robotic process automation)* que automatiza, eficaz y eficiente, los procedimientos manuales de un usuario en el computador. En este trabajo el RPA se desarrolla en Python dado que con ello se puede manipular la GUI del LStudio, automatizando de esta manera la síntesis de los plantines virtuales 3D (Fig. 11). El RPA permite modificar el código C+L desde el GUI o también directamente desde los archivos que componen el modelo de los plantines (ver, Sección 4.1.1.2). Esto ultimo sería la forma más óptima de producción de modelos paramétricos 3D, sin embargo requiere del uso de una Mac, librerías del LStudio (para usar líneas de comandos), Python y el RPA (pyautogui o una "pythonización" del C+L). Obteniendose un lenguaje mixto de gran flexibilidad integrado a todo el ecosistema de librerías que Python ofrece al modelado paramétrico y síntesis de formas digitales (no solo plantas) en 3D.

En este trabajo se desarrollan 6 versiones de modelos de crecimiento virtual de plantines siendo la versión 6 (la versión mejorada con respecto a las previas) la que presenta mayor similitud con los plantines reales de alcachofa. Sin embargo este es un problema abierto aún dado que la codificación en C+L del modelo virtual del plantín puede optimizarse y lograr versones mas exactas o similares a unos reales. De lo mostrado en la Fig. 19 se desprende que puede mejorarse la proporción de las cotiledonas (primeras dos hojas del plantín), ya que la versión virtual tiene las cotiledonas más pequeñas.

A fin de aumentar el "realismo" de los plantines sintéticos y facilitar el entrenamiento del modelo Yolov3, se hace girar a los plantines virtuales obtiniendo con ello diferentes enfoques o angulos que sirven para enriquecer el *dataset* de entrenamiento y validación. Para ello se usan números complejos que facilitan la traslación y orientación de imágenes en el plano imaginario a un menor coste computacional. Posteriormente, los plantines son agrupados en una sola imagen siguiendo un arreglo de 1 a 9 plantines (ver Fig. 20, el arreglo varia según un número aleatorio generado por el módulo *Random* de Python). Al final se genero aproximadamente 15,000 mil imágenes las que fueron archivados en una carpeta para la etapa de entrenamiento (carpeta PLANTINET en el CD adjunto).

Por otro lado, se obtienen aproximadamente 13 mil imágenes de plantines independientes de diferentes versiones (seis versiones), de los que 3 mil son de la versión 2, y 3 mil de la versión 6.

La captura de imágenes de plantines de alcachofa reales se realiza usando el sistema de visión srtificial del robot clasificador de plantines desarrollado en el laboratorio LABINM (UPAO). Cabe mencionar que el sistema de visión computacional (SVC) desarrollado aquí se integra a este robot como parte fundamental de su trabajo.

Mediante el sistema robotico se obtienen 72 imágenes de plantines para el testeo de los modelos de detección y clasificación (DL, ML). A estos plantines se les mide (mediante un vernier) el largo y ancho de la hoja así como la altura de la planta (medida desde el inicio del tallo hasta el extremo de la hoja más alta). Es importante que las imágenes se obtengan de plantines con 21 días de sembrado, dado que es el periodo que se les aplica el control de calidad en los viveros industriales. Ademas la toma de imágenes debe darse un vez llegado los plantines del vivero (hacia el sistema robótico) para evitar distorciones en su crecimiento o morfología por el cambio brusco de ambiente y clima y con ello determinar una falsa clasificación de calidad.

Los modelos de IA propuestos en este trabajo se basan en el modelo Yolov3 para la detección de los plantines. Este modelo da como información las coordenadas del centroide (t_x, t_y), el ancho y alto (t_w, t_h) de la región asociada al plantín en imagen digital (Sección 4.2.2.). Esta información sirve para cortar la imagen y obtener solo la de interés (región asociada al plantin) que a su vez actúan como *input* para los otros tres modelos considerados en este trabajo:

- Modelo 1 (M1): PCA y K-means (Fig. 20).
- Modelo 2 (M2): VGG16 (Fig. 21).
- Modelo 3 (M3): Discriminación por umbral (Fig. 25).

El M1 es un modelo *Machine Learning* (ML), mientras que el M2 y el Yolov3 son modelos de *Deep Learning* (DL). El M3 es un modelo basado en umbrales de centroide en X y Y, así como de área. Aquí se compara la performance de cada modelo según las

métricas descritas en la Sección 3, así como su complejidad y coste computacional (tiempo para realizar la predicción).

Sobre M1, el PCA obtiene dos componentes principales de una matriz de $ancho \times alto \times 3$ (imagen RGB del plantín entregado por el Yolov3).. Pero es de considerar que el PCA se ajusta a un *dataset* específico para descomponerlo en componentes ortogonales sucesivos que permiten conocer los máximos valores de varianza. Es decir, como se observa en la Fig. 30, las cuatro versiones elegidas para ser comparadas tienen una distribución distinta, se refleja diferentes variaciones de tipo de plantines; con lo cual se trabaja en la agrupación de estos. La versión de plantines sintéticos 4 y 5 no son tan buenas con el PCA ya que como se puede ver estos *datasets* contienen clases (mil por cada uno) de plantines que sus componentes principales PC1 y PC2 se traslapan, intersectándose los grupos. Esto complica la agrupación en el plano, por ende. No sucede lo mismo con las versiones 2 y 6. Es más fácil separarlos, así como es más fácil reconocer también que la versión 6 es la más parecida a los plantines reales de alcachofa.

Un aspecto importante del PCA es que la transformación lineal para la reducción de las dimensiones de las muestras con características específicas en un *dataset* influencia el resultado del ajuste del PCA para el establecimiento del PC1 y PC2. O sea, si el *dataset* contiene, p. ej., solo 2 imágenes de plantines (de $120 \times 120 \times 3$), se dice entonces que el PCA ajusta 2 muestras con 43200 características cada una, entregando como resultados los valores PC1 y PC2 para cada muestra. Pero, si a estas 2 mismas imágenes se le incluye una más (habiendo entonces 3 muestras con 43200 características cada una), los valores PC1 y PC2 de las 2 primeras muestras resultantes cambiarán, se ajustarán de forma diferente, entregando valores distintos de PC1 y PC2. Esto quiere decir, que las componentes principales varían no solo por las características que puedan tener las muestras, sino también por el número de éstas.

Este aspecto es importante debido a que el *dataset* de entrenamiento para el PCA-Kmeans (Sección 4.3.1) debe empezar primero por ajustar el PCA no solo al *dataset* sino también a los datos de validación (plantines reales o *inputs*). Caso contrario, estos últimos datos diferirán mucho de los valores PC1 y PC2 del *dataset.* En consecuencia, el M1 errará más

con sus predicciones. Esto deja entrever que la parte crítica del M1 es más con el PCA, ya que el k-means más se centra en agrupar los valores PC1-PC2.

Considerando, entonces, el ajuste del PCA con las muestras del *dataset* y la muestra de entrada (plantín/nes real/es) es considerado como parte del procedimiento para la predicción de la calidad del plantín.

Entonces, como no solo las características de las muestras sino también el número de éstas influyen en el PCA, se consideró hacer una comparación de 4 tipos de *dataset* de entrenamiento.

- Conjunto de 300 plantines sintéticos versión 2 (v2), 100/clase, 270 de entrenamiento y 30 de testeo, 3 clases: bueno, malo y regular.
- Conjunto de 3000 plantines sintéticos v2, 1000/clase, 2700 de entrenamiento y 300 de testeo, 3 clases: bueno, regular y malo.
- Conjunto de 300 plantines sintéticos v6, 100/clase, 270 de entrenamiento y 30 de testeo, 3 clases: bueno, malo y regular.
- Conjunto de 3000 plantines sintéticos v6, 1000/clase, 2700 de entrenamiento y 300 de testeo, 3 clases: bueno, regular y malo.

Los resultados del entrenamiento de las cuatro propuestas de *dataset* para el PCA-kmeans se observan en las Fig. 32 y 33. Lo primero que salta a la vista es el establecimiento de 5 regiones por parte del k-means. Estas regiones se deben clasificar, siendo la región extrema de la izquierda correspondiente a plantines de clase mala, la región subsiguiente a la derecha, plantines de calidad regular. Y las tres últimas regiones del extremo derecho, corresponden plantines de calidad buena. Teniendo esto en cuenta, los resultados de entrenamiento usando los *datasets* mencionados líneas arriba, permiten lograr exactitudes no menores a 96%.

Al momento de la validación se usó 48 imágenes de las 72 que se tenían mapeadas, debido a que las 24 restantes no se encontraban en las condiciones adecuadas (giradas, inclinadas, fuera de foco, etc.). Estos 48 plantines fueron previamente clasificados considerando los parámetros que provee el vivero industrial Agrogénesis (Tabla 1), se observa que el

resultado más favorable está ligado al *dataset* de 3000 plantines sintéticos v6 que cuenta con una exactitud de 67% frente al 46% del *dataset* con los mismos plantines, pero a una cantidad de 270 imágenes de entrenamiento. Los resultados con el *dataset* (plantines v2) para ambas cantidades de imágenes no pasa del 48%, lo cual termina por descartar tal propuesta. Es importante observar también las capacidades que tienen estos modelos para reconocer la clase de plantín. Pues para el *dataset* con mejores resultados, mencionado líneas arriba, la precisión para determinar plantines de buena calidad es del 70% y de mala, un 67%. Es importante recalcar que la precisión considera, de un conjunto de afirmaciones, cuál es el porcentaje correctas de éstas, mientras que el *recall* (conocido también como sensibilidad) permite reconocer la habilidad de un modelo para establecer del total de afirmaciones que se saben correctas el porcentaje de cuántas de éstas se han predicho bien. Ambas métricas son relevantes, y para considerarlas ambas como uno solo el *f1-score* entra a tallar (Sección 3.5). Y el *f1-score* para el mejor resultado del M1 favorece también al *dataset* de plantines v6, ostentando un 82% para la clase buena y un 64% para predecir la clase mala, mientras que para la clase regular un 30%. En cuanto a esta última clase, solo el *dataset* de 3mil plantines v2 permite un *f1-score* de 43%.

Se puede deducir que, a mayor cantidad de muestras el ajuste del PCA con plantines sintéticos y reales favorece la exactitud. Sin embargo, también se observa que a medida que el *dataset* incluye más muestras, mayor es el tiempo de entrenamiento y predicción.

Con respecto al modelo 2 (M2) que trabaja con el VGG16, cabe resaltar que la arquitectura propuesta (Fig. 25) es una modificación ligera del VGG16, al extraer la cabecera del mismo e incluir capas que lleven a solo 3 valores de salida (porcentajes de predicción según clase malo, regular y bueno). Esta cabecera puede incluir otro tipo de capas que mejore la interpretación de la extracción de los patrones que entrega este modelo. Al observar la Fig. 26 es posible entender la forma en que las convolusiones van "abstrayendo" la observación (imagen de plantín), pudiendo establecer pequeñas imágenes que disminuyen su variabilidad según la entrada de las imágenes de plantines. Un estudio más profundo del aprendizaje de esta red neuronal artificial puede mejorar significativamente la arquitectura de la cabecera (modificable) del M2 en pro de mejorar la exactitud de las predicciones finales. Se hace énfasis en la modificación solo de la

cabecera a fin de mantener las capacidades del VGG16 pre-entrenado en *Imagenet* que, de la mano con la transferencia de aprendizaje, permite desarrollar por encima un modelo más robusto y exacto. De lo contrario tendría que entrenarse desde cero toda la red VGG16 eliminando las capacidades obtenidas en el entrenamiento con las imágenes del dataset *Imagenet.* Esto no es recomendable. A este proceso de usar un modelo pre-entrenado se le conoce como *Transfer Learning*

En la Fig. 36 se muestran los resultados de entrenamiento del VGG16 considerando tres *datasets*:

- Conjunto de 3000 plantines sintéticos v2, 1000/clase, 2400 de entrenamiento y 600 de testeo, 3 clases: bueno, malo y regular. Épocas de entrenamiento: 14.
- Conjunto de 3000 plantines sintéticos v5, 1000/clase, 2400 de entrenamiento y 600 de testeo, 3 clases: bueno, regular y malo. Épocas de entrenamiento: 24.
- Conjunto de 3000 plantines sintéticos v6, 1000/clase, 2400 de entrenamiento y 600 de testeo, 3 clases: bueno, malo y regular. Épocas de entrenamiento: 20.

Si bien es cierto no se menciona ni se incluye como resultados de entrenamiento, el uso de una mayor cantidad de plantines sintéticos (>3k) no tuvo cambios significativos en el error ni exactitud del modelo. Este aspecto no se profundizó, por lo que queda pendiente un mejor análisis. De hecho, si se incluyen más detalles en los modelos de plantines sintéticos, aumentando la cantidad de muestras se espera se mejoren las predicciones.

Los plantines sintéticos de versión 2 y 6 son los que se eligieron para realizar la validación con los plantines reales. Esto es debido a que las exactitudes de testeo de ambas versiones son mejores. Aunque las gráficas del Fig. 36 indiquen que los modelos de plantín de versión 5 y 6 producen *overfitting* (es decir, error de testeo tendiendo a la subida mientras que el error de entrenamiento a la bajada), la versión 6 igual es considerada para la posterior validación, pues su exactitud de testeo parece llegar a estabilizarse en un 60%.

Es interesante mencionar que durante las pruebas de entrenamiento se observa que hay una tendencia a que el *dataset* que parece más “agrupable” considerando las distribuciones obtenidas por el PCA en la Fig. 30, tiende a ser más estable para las durante

de entrenamiento como la exactitud de testeo. El *dataset* de plantines sintéticos v2 al diferenciarse mejor sus clases en las gráficas PC1 *vs* PC2, responde mejor durante la etapa de entrenamiento con el VGG16. Aunque este aspecto no influye tanto durante la etapa de validación, como se analiza a continuación.

En la Fig. 37 se observa que la validación favorece a los plantines sintéticos v6 con una exactitud de 65%. Esta validación se realiza considerando 47 imágenes de plantines entregados por el Yolov3 y una imagen más que el Yolov3 no pudo detectar (fue ingresado manualmente en este *dataset* de validación, sumando 48 imágenes).

Como se muestra en las matrices de confusión de la Fig. 37, el VGG16 entrenado con el *dataset* de plantines sintéticos v6, es bueno también para predecir plantines de calidad mala como buena. Este dato se confirma directamente con el *f1-score*, un 83% para plantines buenos, y 62% para los malos; los regulares se mantienen bajos con un 22%. La ventaja es clara con respecto al VGG16 entrenado con el *dataset* de plantines v2, que solo puede equiparse al anterior con un *f1-score* de 61% para clasificar plantines malos; no teniendo el mismo resultado para los buenos con un 73%, ni mucho menos para los regulares con un 11%, siendo este último el resultado más bajo.

Con respecto al modelo Yolov3 como se plantea en la Fig. 27, es una red neuronal artificial y convolucional considerablemente profunda (a comparación de la VGG16 de 24 capas) denotando como arquitectura la constitución de 106 capas. Esta red está basada en el *framework Darknet 53*, originalmente escrito en C++ y extrapolado al Python para el entrenamiento de 4 versiones diferentes de Yolo. En este trabajo no se optó por la versión 4 del Yolo debido a que ésta para las predicciones necesita GPUs, las cuales no se han considerado en este trabajo. La versión 3 se debe entrenar sí considerando GPUs, que pueden estar disponibles públicanmente en la plataforma *Google Colaboratory*; no obstante, al momento de las predicciones se puede prescindir de GPUs y hacer uso de las librerías del OpenCV y el CPU de un computador con el Python instalado. Esto hace posible no solo usar computadores convencionales de 64bits sino también computadores con procesadores ARM de 32bits (se puede incluir el *TensorFlowLite*).

La complejidad del Yolov3 a través del *darknet 53* no permite realizar modificaciones rápidas como sí lo permite la arquitectura del VGG16. Sobre el Yolo se conocen solo aspectos generales explicados en la Fig. 27; entonces, este modelo es tratado como una caja negra porque inclusive observar cómo se va abstrayendo la red, requiere de un esfuerzo considerable en ir interpretando el código de la arquitectura y liberando las imágenes que se producen después de las convoluciones como se hace con el VGG16 en la Fig. 26. Este aspecto tampoco se ha considerado en este trabajo por escapar del objetivo principal.

La preparación del *dataset* para el Yolov3 requiere de un tratamiendo especial de las imágenes (Sección 4.1.3). Básicamente es establecer la centroide del plantín sintético y el ancho y alto de la caja que se ajusta a éste. Las imágenes de plantines sintéticos v6 (3mil en total) son las elegias para este proceso ya que son las más parecidas a las imágenes de plantines reales de alcachofa. Se cuenta con 2 modos de generar el etiquetado de dichas imágenes: automático y manual. Esta segunda opción requiere abrir cada imagen en un programa de edición de imágenes (p.ej., *Paint*) para que a través del cursor se pueda ubicar la centroide de la imagen, y a continuación, se establece el ancho y alto de una caja imaginaria que encierra o se ajusta a la figura del plantín. Este proceso es muy tedioso al tener que repetirlo para 3mil plantines; prácticamente es inviable. El modo automático es el más amigable, pero requiere de un programa que permita detectar automáticamente la centroide y el ancho y alto de la imagen. Éste es el modo que se trabaja en este proyecto. Se procesa la imagen independiente del plantín, se obtienen los mencionados parámetros y se lo incluye en la imagen de plantines agrupados. El código de la manipulación de planintes y su etiquetado se puede encontrar en los Anexos IV.

Con respecto al entrenamiento del Yolov3, en la Fig. 38 se muestran el *dataset* de plantines agrupados que se propone usar. Si bien es cierto el *dataset* contiene 1000 imágenes de plantines agrupados, un total aproximado de plantines en estas imágenes es de 5000, ya que cada imagen de grupos de plantines sintéticos incluye entre 1 a 9 plantines, en promedio unos 5. La fuente de imágenes de planites sintéticos v6 usados es de 3000 (1000/clase) pero al incluir la manipulación de plantines tanto en ubicación y

orientación (Fig. 14) se tienen figuras de plantines únicas; siendo, entonces, un *dataset* de gran variabilidad debido a la diversidad de la forma de sus objetos.

Vale recalcar que el entrenamiento del Yolov3 se hace por medio del *Google Colaboratory*. La duración del entrenamiento es aprox. 8hrs. A más capas, más profunda la red neuronal artificial, mayor el tiempo de entrenamiento. También aumenta este tiempo de considerarse un *dataset* con más muestras.

El Yolo a parte de la tarea de detección de objetos también cumple con la clasificación de estos. Sin embargo, en este trabajo con relación al Yolov3 solo se trabaja con la predicción de detección. La razón es que se encontraron complicaciones con la detección de plantines de mala calidad que, como se muestra en la Fig. 39, las manchas o inclusive partes más pequeñas de un plantín de buena calidad, p. ej., son detectados como plantines y clasificados como de mala calidad. Esto no es tan conveniente, por lo que se opta solamente por entrenar al Yolov3 con plantines de calidad regular y buena solo para la detección de los mismos. Esto no quiere decir que el modelo no detecta a los plantines de mala calidad; sí lo hace, solo que lo clasifica como regular. Por consiguiente, la predicción de clasificación del Yolov3 al final se omite y se deja dicha tarea a los modelos M1, M2 y M3 (Fig. 29).

En la Fig. 41 se pueden observar los resultados de las detecciones de plantines sean de buena, regular o mala calidad. Todos son clasificados como regulares. Pero como se explicó en el párrafo, esta clasificación se omite. Inclusive el modelo puede llegar a detectar plantines con fondos diferentes como la bandeja de los plantines. Parece interesante poder profundizar dicho análisis y usarlo en la detección de plantines no solo en imágenes sino a través de videograbación *offline* o en tiempo real, inclusive.

La métrica de la detección de plantines usado para medir el desempeño del Yolov3 es la Intersección sobre la Unión (IoU) como se explica en la Sección 2.2.5.3. El resultado en este aspecto es de un 57% de IoU, lo cual según la bibliografía revisada (Redmon et. al., 2016) un porcentaje mayor al 50% es aceptable; lo cual confirma las observaciones en la Fig. 41. Este aspecto, según el criterio del autor del presente trabajo, representa un punto

muy importante a considerar. El hecho de haber entrenado al modelo Yolov3 no con imágenes reales de plantines de alcachofa sino con imágenes sintéticas que emulan la forma de la planta logrando un parecido relevante, marca un precedente para el entrenamiento de modelos de detección de objetos entrenados (vale recalcar) no con objetos reales sino más bien con sus representaciones matemáticas e ideales; ahorrándose así, el trabajo en los siguientes aspectos:

- Muestreo automatizado de gran cantidad de imágenes de los objetos, en este caso de las plantas.
- Reducción de costos de adquisición de objetos o producción de plantas reales.
- Reducción de tiempos de desarrollo de modelos para la detección de objetos o plantas
- Aumento de la flexibilidad de formas y tamaños, así como de colores, de los objetos o plantas. Sumado a este aspecto la utilización de librerías como la *ImageDataGenerator* de *Tensorflow* para el aumento del *dataset*, es decir, modificación del tamaño, forma y orientaciones del mismo *dataset* en tiempo de entrenamiento del modelo, mejora aún más los resultados de exactitud.

Con relación al modelo M3 (Discriminación por umbral) es necesario considerar plantines reales para establecer su centroide y área del marco que se ajusta a la forma de la planta. A mayor cantidad de imágenes de plantines reales, más fino se vuelve este modelo. Se consideró un total de 2 bandejas de plantines reales, las cuales permitieron el crecimiento de 72 plantines (se consideran los plantines que están volteados, o movidos a fin de conocer adónde se ubican las centroides para estos plantines); en algunas celdas de dichas bandejas no se manifestó desarrollo alguno de plantín o las agujas del robot que los elevaron, presentaron inconvenientes por el sustrato y el proceso mismo del desarrollo de las agujas del robot.

En la Fig. 42 se muestran la ubicación de los centroides de los 72 plantines, mediante los cuales se puede apreciar una tendencia bien clara a agruparse en básicamente 3 regiones. Cada una de éstas corresponde a las agujas del robot. Inclusive por cada región se diferencian 2 subregiones, que corresponden pues, al movimiento de la misma aguja al

elevar los plantines pares e impares de las filas con 6 plantines de la bandeja. Esto hace posible no solo detectar la región de la aguja sino también el plantín par o impar elevado.

Se usan también los 72 plantines reales muestreados previamente para que el Yolov3 entregue la predicción de la caja que encierra al plantín. Una vez obtenida la predicción del Yolov3 para estos plantines reales, se calcula el área de estos multiplicando $t_w \times t_h$. Consiguientemente, estos valores son graficados en la Fig. 43 obteniéndose, además, el promedio y la desviación estándar para los plantines reales de mala, regular y buena calidad. La incertidumbre para los plantines de mala y buena calidad son elegidos para establecer los umbrales de área del M3.

La exactitud de predicción de la tarea de clasificación de los plantines de alcachofa para M3 es de 77%, considerando 47 plantines reales (Fig. 41). Este modelo presenta como matriz de confusión un resultado favorable para la predicción de plantines buenos y malos, con lo cual se puede calcular el *f1-score,* lográndose un 84% para la predicción de la calidad de plantines buenos, 83% para los malos y 59% para los regulares.

Un panorama más claro a fin de comparar los modelos propuestos (M1, M2 y M3) en la Fig. 29, se puede observar en la Tabla 8. En ésta se exponen la mejor versión de cada modelo con relación a la tarea de clasificación de plantines según su calidad: bueno, regular y malo. En resumen, el modelo de PCA-k-means (Fig. 24) ajustado con el *dataset* de 2700 plantines sintéticos v6 (900/clase) para M1; el modelo VGG16 propuesto en la Fig. 25 y entrenado con un *dataset* basado en 3000 plantines sintéticos v6 (1000/clase) para M2; y la discriminación por umbrales de centroide y área propuesta en la Fig. 29, determinados a través de 48 imágenes de plantines reales recolectadas en el laboratorio y clasificadas según lo explicado en las Fig. 38 y 39, y la Tabla 1.

Según lo que se observa en la Tabla 8, M3 se asemeja a M2 y M1 con respecto a la predicción de clasificación de plantines buenos, con un 83%. Los tres modelos trabajan correctamente para este tipo de plantines. Sin embargo, para la clasificación de plantines regulares y malos, el M3 supera en creces a los otros dos, al contar con un *f1-score* de 59% y 84%, respectivamente. Es por ello que la exactitud del modelo M3 es la mejor, con

un 77%. Agregado a ello el M3 también supera a los otros dos modelos con relación al tiempo total (TT) estimado de la predicción (incluyendo detección y clasificación), ya que permite predecir en 5 milisegundos/plantín, mientras que el M2 a 8 milisegundos/plantín y el M3 a 604 milisegundos/plantín (Tabla 8).

El tema de los tiempos de predicción probablemente no sea tan significativo con los resultados obtenidos. No obstante, si el sistema de visión por computadora evalúa 3 plantines por imagen *input*, e incluye otros procesos también que representan un tiempo en específico como la recepción y envío de datos al PLC, la captura y segmentado de las imágenes, etc., podría finalmente, acumular un tiempo que vaya en detrimento de la eficiencia del robot clasificador.

Para llevar a cabo la correlación se consideró más imágenes de plantines. Específicamente se trabajaron con 619 plantines obtenidos de 10 bandejas entregadas por un vivero agroindustrial de la región, estas bandejas fueron previamente clasificadas por el personal del vivero. Los detalles de la clasificación se pueden observar en los Anexos XV. Como se muestra en la Fig. 48 y en la Tabla 9, se puede afirmar que M3 representa la mejor opción al contar con un 85% de correlación positiva, con un valor-p muy por debajo del *alfa* (0.01), permitiendo entonces asegurar que la correlación es estadísticamente significativa.

Finalmente, considerando que la propuesta del modelo 3 tiene la correlación más fuerte, es ésta la que se usa para el sistema de visión por computadora (SVC) del robot. Sistema esquematizado en la Fig. 15. El sistema integrado SVC compuesto de ocho partes, fuera del PLC y el robot en sí, participa como un esclavo a la escucha de las órdenes programadas en la secuencia operativa del PLC. La arquitectura de integración SVC propuesta en las Fig. 52 y 53 puede servir como referencia en proyectos de IoT industrial basado en IA ya que permite integrar equipos industriales, algoritmos avanzados de *Deep Learning* (DL) y computación en la nube. La interfaz entre los equipos industriales y los algoritmos de DL (manejados en un programa gestor de visión por computadora) es el protocolo de comunicación MODBUS TCP; mientras que la interfaz entre este programa gestor y la nube (o la internet) es el protocolo de comunicación MQTT. El hecho de usar

localmente el modelo de IA para las respectivas prediccinoes, permite abaratar costos en cuanto a un sistema IIoT, donde el PLC funciona de forma local. De esta forma no se tiene que enviar solicitudes ni imágenes a la nube para realizar las predicciones, sino más bien solo se envían a la nube los resultados de las mismas, y datos pertinentes de dicho proceso; usando más que todo la computación en la nube, como una forma de hacer solo *Business Intelligence*, o sea visualización y análisis de datos.

CAPÍTULO VI

CONCLUSIONES Y RECOMENDACIONES

"Haz sólo lo que amas y serás feliz, y el que hace lo que ama está benditamente condenado al éxito, que llegará cuando deba llegar, porque lo que debe ser, será; y llegará naturalmente"

Facundo Cabral.

CONCLUSIONES

En el presente trabajo se desarrolló una estrategia inteligente para la medición de la calidad del crecimiento de plantines de alcachofa producidos en viveros industriales de la región La Libertad, haciendo uso de redes neuronales convolucionales y todo un despliegue de tecnologías de la información integradas en un sistema denominado Sistema de Visión por Computadora (SVC).

La estrategia inteligente basada en el SVC se desarrolló considerando una arquitectura compuesta por los modelos Yolov3 y discriminación por umbral (D. por umbral) que se desempeñan en las tareas de detección y clasificación de plantines, respectivamente. Para la predicción de detección se logró un IoU de 57%, lo cual significa una buena detección de las imágenes de plantines segmentados e inclusive con fondo de bandejas, consiguiendo cuatro parámetros relacionadas a la imagen del plantín: la centroide (t_x, t_y) y el ancho (t_w) y alto (t_h); parámetros con los cuales es posible cortar un recuadro que se ajuste al plantín de la imagen, de forma que se pueda usar como entrada para el modelo de clasificación, el discriminador por umbral. Por medio de este modelo se logró un 77% de exactitud, un *f1-score* para la clasificación de plantines buenos, regulares y malos de 83%, 59% y 84% respectivamente.

Se calculó una correlación positiva de Pierson existente del 85% (con un valor-p de 5.22×10^{-14}) entre las clasificaciones realizadas mediante el SVC y la basada en el criterio de un vivero industrial de la región que usa para la clasificación de los plantines de alcachofa durante el proceso de control de calidad. Es una correlación fuerte y estadísticamente significativa al ser el valor-p menor que un $alfa = 0.01$. Los indicadores de las correlaciones se encuentran en la Fig. 48 y en la Tabla 9, con lo cual se afirma que la variable independiente satisface la demanda de la variable dependiente. Es decir, el SVC puede equiparse al criterio de un jornalero para el repique de un vivero industrial con una correlación del 85% para cumplir con la tarea de clasificación de plantines de alcachofa según su calidad: bueno, regular o malo.

Para cumplir las propuestas de la variable independiente se consideró desarrollar, primero, un conjunto de *datasets* de imágenes de plantines sintéticos en formato digital y tridimensional. Se tuvo que seguir aquello debido al contexto en el que se desarrolló el presente proyecto, durante la pandemia del covid-19 las cuarentenas impuestas por el gobierno dificultaron por un largo período (aprox. 6 meses) los trabajos presenciales y por ende la recolección de plantines de alcachofa. Por ende, para superar tal aspecto se usaron Sistemas L libres de contexto (Sistema-DOL) que mediante la librería CPFG, el software LStudio, y el lenguaje mixto C+L (combinación de C y Sistemas L), hicieron posible modelar el crecimiento de la versión digital del plantín de alcachofa, para ser exportado, posteriormente, en formato .png como una captura de imagen. Para la producción automática y masiva de las imágenes de los plantines sintéticos se desarrolló un método basado en *Robotic Process Automation* (RPA) el cual hizo posible "pythonizar" el lenguaje C+L, es decir, controlar la síntesis de plantines a través de código hecho en Python usando la librería *pyautogui*. Se tiene tres conjuntos de *datasets*, uno de imágenes de plantines sintéticos (los 3 tipos) individuales (15mil) y otro agrupados (mil); y un tercero con imágenes que fueron recolectadas mediante el robot clasificador implementado, con un total de 72 imágenes de plantines de alcachofa reales, previa medición de la altura, el ancho y largo de la hoja de alcachofa (ver Tabla 1).

Posteriormente, el segundo indicador de la variable independiente relacionado a las propuestas de modelos predictivos, fue obtenido a través del modelo Yolov3 para la predicción de detección de plantines de alcachofa y tres modelos para la predicción de clasificación de la calidad del plantín, tal y como se observa en la Fig. 25: uno basado en PCA y K-means (M1), el otro en el VGG16 (M2) y uno basado en discriminación por umbrales (M3).

Los resultados del entrenamiento para la tarea de clasificación de plantines según su calidad (bueno, regular y malo), como parte del tercer indicador, fueron logrados considerando los *datasets* de plantines sintéticos y reales, así como los modelos propuestos. El M1 fue entrenado con dos *datasets* de plantines sintéticos (versión 2 y 6) con dos números de muestras, uno de 300 y el otro de 3000. Se observó que el

dataset con mayor cantidad de muestras de plantines sintéticos (v6, los más parecidos a los reales) es el que tiene mejores resultados, aunque a expenses del tiempo de predicción, el cual aumenta a medida que el *dataset* crece. El M2 fue entrenado con tres tipos de *dataset* de plantines sintéticos (v2, v5 y v6), observándose que con los de v2 hacen posible lograr exactitudes de testeo más estables, siendo éste el menos parecido a los plantines reales. Las otras versiones tienden al *overfitting*. No obstante, aun presentando los resultados con la v6 baja exactitud de testeo, al momento de la validación (testeo del modelo con plantines reales) éste es el que presenta mejores resultados. El M3 se diseñó como un modelo basado en umbrales, es decir, toma los valores que le entrega el modelo Yolov3 (t_x, t_y, t_w, t_h) para establecer el centroide (t_x, t_y) y el área $(t_w \times t_h)$. Para establecer dichos umbrales fue necesario hacer uso del *dataset* de plantines reales, graficar los centroides y hallar el promedio y la desviación estándar del área de los plantines de calidad mala y buena, que fueron determinados como la referencia. Tal y como se muestar en la Fig. 43.

Con relación al cuarto indicador, los resultados de los modelos propuestos para la clasificación fueron resumidos en la Tabla 8. Se observó que los tres modelos (M1, M2 y M3) cuentan con un *f1-score* muy cercano de 83% para la clasificación de plantines buenos; sin embargo, no sucede lo mismo para clasificar los platines regulares, con lo cual se tuvo 22% para el M2 y 30% para el M1. Ambos mejoran sus capacidades para la clasificación de los plantines malos, a un 62% y 64%, M1 y M2, respectivamente. Mientras que, con relación a la exactitud, el M3 es el mayor (como se mencionó párrafos arriba) con un 77%, mientras que el M2 logra un 65% y el M1, un 67%.

Cabe resaltar que también fue analizado el tiempo de predicción por modelo, contando también con el tiempo de predicción del Yolov3; siendo el M3 el más eficiente al registrar un aprox. de 5 milisegundos/plantín. El M1 registró aprox. 604 milisegundos/plantín y el M2, aprox. 8 milisegundos/plantín. Si bien es cierto se observa que el M1 y M2 no registran tiempos considerables, vale mencionar que sí podría ser un problema al incluir en la cuenta los tiempos que se incurren en la

comunicación entre dispositivos, el movimiento del robot, la captura de la imagen y la segmentación, entre otros.

Se desataca también lo logros obtenidos en cuanto al despligue del modelo con una arquitectura de software híbrida basada en una red local y en la nube. Con este aspecto se satisface el quinto indicador. Con lo cual se puede decir que esta propuesta hace posible trabajar con el SVC teniendo integrado un modelo de Deep Learning en un programa que trabaja con componentes industriales como PLC, así como tecnologías en la nube *Google Cloud Platform*.

Finalmente, el desempeño basado en la intersección sobre la unión (IoU) del Yolov3 habiendo sido entrenado solo con plantines sintéticos para la predicción de detección fue de un 57% validando con el *dataset* de 47 plantines reales. Este resultado es de notable relevancia al considerar que entre 50 y 60%, según la bibliografía consultada, se ubican los resultados de IoU para las tareas de detección de objetos. Por lo tanto, se puede afirmar que el modelo Yolo versión 3, no necesita de plantines reales para entrenar su capacidad de detección; lo cual, en resumidas cuentas, permite reducir costos de entrenamiento y tiempo de desarrollo de modelos de IA orientados a las tareas de detección de plantas, flores, y otros objetos que se puedan producir con los Sistemas L.

RECOMENDACIONES

Las siguientes recomendaciones son divididas en cuatro partes: elaboración del *dataset*, diseño del modelo de IA, entrenamiento de los modelos y despliegue de los mismos.

Con relación a la elaboración del *dataset*, se recomienda básicamente tres puntos. El primero es sobre la mejora del modelo en C+L de los plantines sintéticos v6. Aún hay algunos aspectos de proporción entre las hojas cotiledonas y las de alcachofa por mejorar para que se parezca aún más a los plantines reales. Este aspecto mejora los resultados del entrenamiento del Yolov3 para la tarea de detección. Como segundo aspecto, producir los plantines sintéticos haciendo uso del L-Studio en una MacOS para poder sintetizar por línea de comandos los archivos del modelo C+L previamente modificados a través de RPA (*Robotic Process Automation*); esto haría mucho más eficiente el proceso de producción. El tercer punto, es sobre la recolección de imágenes de plantines reales. Se recomienda incluir una cámara que permita capturar la imagen vertical del plantín o en hacer uso de la información de profundidad de la cámara *Realsense* para entrenar un modelo de predicción de clasificación con capacidades de procesar un *input* de imagen RGB-D. Esto último sería novedoso, así como también más exacto debido a que la profundidad permite calcular con mayor precisión el área y alturas reales del plantín; pero probablemente haya un coste mayor de entrenamiento al tener que diseñar un modelo de DL (*Deep Learning*) particular para este tipo de imágenes con profundidad. En caso se opte por usar una segunda cámara para capturar la imagen vertical del plantín, se pueden generar también imágenes de los plantines sintéticos con la misma perspectiva como insumo para el posterior entrenamiento del Yolov3 en miras de la detección.

Relacionado al diseño del modelo, complementando lo dicho líneas arriba, se puede proponer el desarrollo o entrenamiento de un modelo ya existente para la mejora de la segmentación de los plantines. Lo recomendable sería directamente de la información RGB-D que entrega la *Realsense*, o sino solo con la imagen RGB. De ser esto último se podría usar la predicción del Yolov3 como *input* para la tarea de segmentación que, una vez realizada, es posible aplicar un modelo mejorado para la predicción de clasificación de los plantines, o el objeto que se encuentre en estudio. La tarea de detección, para ubicar

el objeto y extraerlo de la imagen; la tarea de segmentación, para establecer el contorno de los objetos y separar el fondo de la forma entera de los mismos; y la tarea de clasificación, para determinar la clase a la que pertenece la forma extraída. Estas tres tareas se pueden establecer como un procedimiento repetitivo en el procesamiento de imágenes digitales para la visión por computadora.

Es importante considerar también el entrenamiento de un modelo de DL, ya que puede requerir éste de hardware especializado como GPUs o TPUs (*TensorFlow Processor Unit*). Se recomienda en este caso optar también por probar con otros tipos de experiencia como el entrenamiento auto-supervisado (*Self-supervised learning*), propuesta del estadounidense Yan LeCun, la cual se presenta en la actualidad como una alternativa interesante para la búsqueda de patrones de una forma más rigurosa y automatizada.

Finalmente, con relación al despliegue de los modelos, se podría involucrar más a la nube haciendo uso de los servicios del GCP (*Google Cloud Platform*) o AWS (*Amazon Web Service*) a fin de poder correr el modelo en algún servidor que se tenga acceso a éste de forma remota.

REFERENCIAS BIBLIOGRÁFICAS

Agrolalibertad. (2020). Gerencia Regional de Agricultura La Libertad: exportaciones alcachofa conserva 2012. Expo Alcachofa | GRA La Libertad. Retrieved November 8, 2021, from http://www.agrolalibertad.gob.pe/index.php?q=node%2F1480.

Agrolalibertad (2015). Gerencia Regional de Agricultura La Libertad: Exportaciones de alcachofa. Expo Alcachofa | GRA La Libertad. Retrieved November 8, 2021, from: http://www.agrolalibertad.gob.pe/sites/default/files/Nota%20Informativa_%20ALCACHOFA%20EN%20LA%20LIBERTAD_2015.pdf

An, J., Li, W., Li, M., Cui, S., & Yue, H. (2019). Identification and classification of maize drought stress using deep convolutional neural network. *Symmetry*, *11*(2), 1–14. https://doi.org/10.3390/sym11020256

Atzori, L., Iera, A., & Morabito, G. (2017). Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, *56*(December), 122–140. https://doi.org/10.1016/j.adhoc.2016.12.004

Brownlee, 2019. 14 Different types of learning in machine learning. Descargado de: https://machinelearningmastery.com/types-of-learning-in-machine-learning/

Balsys, (2018). Pylessons – Yolov3 object detection mAP metric. Retreieved November 8, 2021, from: https://pylessons.com/YOLOv3-TF2-mAP/

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020, April 23). Yolov4: Optimal Speed and accuracy of object detection. arXiv.org. Retrieved November 9, 2021, from https://arxiv.org/abs/2004.10934.

Choque Moscoso, C. J. (2018). Desarrollo de un robot cnc tipo cartesiano de la marca farmbot como soporte tecnológico para el proceso de control de calidad de plantines (Tesis de pregrado). Universidad Particular Antenor Orrego. Trujillo, Perú.

Choque Moscoso, C. J., Fiestas Sorogastúa, E. M., & Prado Gardini, R. S. (2018). Efficient implementation of a Cartesian Farmbot robot for agricultural applications in the

region La Libertad-Peru. *Memorias del Congreso Internacional IEEE Colombia ANDESCON*.

Durvea, M. L. (ed.). 1985. Proceedings: Evaluating *seedling* quality: principles, *procedures, and predictive* abilities *of* major tests. Workshop held October 16-18, 1984. Forest Research Laboratory, Oregon State University, Corvallis, ISBN 0-87437-000-0

Dabbura, 2018. K-means clustering: algorithm, applications, evaluation methods and drawbacks. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a#:~:text=Kmeans%20algorithm%20is%20an%20iterative%20algorithm%20that%20tries,keeping%20the%20clusters%20as%20different%20%28far%29%20as%20possible.

Fiestas Sorogastua, E., & Prado Gardini, S. (2018). Modeling and Simulation of Kinematics and Trajectory Planning of a Farmbot Cartesian Robot. *25th IEEE International Conference on Electronics, Electrical Engineering and Computing, INTERCON 2018.* Lima, Perú.

Goodfellow, I., Bengio, Y., & Courville, A. (2016) . *Deep learning*. MIT Press, http://www.deeplearningbook.org

Grossnickle, S., & MacDonald, J. (2018). Seedling Quality: History, Application, and Plant Attributes. *Forests*, *9*(5), 283. https://doi.org/10.3390/f9050283

Hennings, T. (2017). Stages of the Marijuana Plant Growth Cycle in Pictures | Leafly. Retrieved July 21, 2019, from https://www.leafly.com/news/growing/marijuana-plant-growth-stages

Hofmann, P., Samp, C. & Urbach, N. Robotic process automation. (2020, Aug 08) *Electron Markets* **30,** 99–106. https://doi.org/10.1007/s12525-019-00365-8

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hollstein, T., Peng, Z., Ubar, R., & Glesner, M. (2002). *Challenges for Future System-on-Chip Design Challenges for Future System-on-Chip Design*. (September).

Hornik, K. (1991). Approximation Capabilities of Multilayer Neural Network. *Neural Networks*, *4*(1989), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T

Husin, Z. B., Shakaff, A., Aziz, A., & Farook, R. (2012). Feasibility study on plant chili disease detection using image processing techniques. *Third Int. Conf. Intelligent Syst.. Modelling and Simulation (ISMS)*, 291-296.

Hossin-Sulaiman, (2015). A REVIEW ON EVALUATION METRICS FOR DATA CLASSIFICATION EVALUATIONS. Retrieve November 9, 2021, from: https://www.semanticscholar.org/paper/A-REVIEW-ON-EVALUATION-METRICS-FOR-DATA-EVALUATIONS-Valuations/6c30f90b0c65d75bc37e279a861e0f9ef8a06e77

Javier, A., & Pérez, J. (2013). *Indicadores De Calidad De Planta En Cuatro Viveros*. 56–62. Retrieved November 9, 2021, from: http://eprints.uanl.mx/15965/

Jeon, W.-S., & Rhee, S.-Y. (2017). Plant Leaf Recognition Using a Convolution Neural Network. *The International Journal of Fuzzy Logic and Intelligent Systems*, *17*(1), 26–34. https://doi.org/10.5391/ijfis.2017.17.1.26

Lecun, Y., Eon Bottou, L., Bengio, Y., & Haaner, P. (1998). Gradient-Based Learning Applied to Document Recognition RS-SVM Reduced-set support vector method. SDNN Space displacement neural network. SVM Support vector method. TDNN Time delay neural network. V-SVM Virtual support vector method. *Proc. of the Ieee*, (November), 1–46.

Lin, K., Si, H., Chen, J., & Wu, J. (2018). *Advances in Image and Graphics Technologies*. *757*, 1–7. Retrieved November 9, 2021, from: https://doi.org/10.1007/978-981-10-7389-2

Mochida, K., Koda, S., Inoue, K., Hirayama, T., Tanaka, S., Nishii, R., & Melgani, F. (2018). Computer vision-based phenotyping for improvement of plant productivity: A

machine learning perspective. *GigaScience*, *8*(1), 1–12. https://doi.org/10.1093/gigascience/giy153

Moncada G., G. (2011). Perú: Estimaciones y Proyecciones de población, 1950 - 2050. Recuperado: de https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib0466/Libro.pd

Mech, R., James, M., Hammel, M., Hanan, J., Prusinkiewicz, P. (2005, May 31). CPFG Version 4.0 User's Manual. Retrieve November 9, 2021, from http://algorithmicbotany.org/lstudio/CPFGman.pdf

Minervini, M., A. Fischbach, H.Scharr, and S.A. Tsaftaris. (2015). Finely-grained annotated datasets for image-based plant phenotyping. Pattern Recognition Letters, pages 1-10, doi:10.1016/j.patrec.2015.10.013

Machart, P., & Ralaivola, L. (2012, May 24). Confusion matrix stability bounds for multiclass classification. arXiv.org. Retrieved November 9, 2021, from https://arxiv.org/abs/1202.6221.

Niebel, D., Kopp, G., & Beerfeltz, H. (2013). Information and communications technology (ICT) Key technologies for sustainable development. *BMZ Strategy Paper*, *2*, 1–32. Retrieved November 9, 2021, from: http://article.sapub.org/10.5923.j.scit.20120205.06.html

Non-destructive growth measurement of selected vegetable seedlings using orthogonal images. (2005). *Transactions of the American Society of Agricultural Engineers*, *48*(5), 1953–1961. https://doi.org/10.13031/2013.19987

Noda, K., Ezaki, N., Takizawa, H., Mizuno, S., & Yamamoto, S. (2006). Detection of plant saplessness with image processing. *International Joint Conference SICE-ICASE*, 291-296

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016, May 9). You only look once: Unified, real-time object detection. arXiv.org. Retrieved November 8, 2021, from https://arxiv.org/abs/1506.02640v5.

Redmon, J. and Farhadi, A., 2016. YOLO9000: Better, Faster, Stronger. [online] arXiv.org. Available at: <https://arxiv.org/abs/1612.08242> [Accessed 8 November 2021].

Redmon, J., & Farhadi, A. (2018, April 8). Yolov3: An incremental improvement. 1ZarXiv.org. Retrieved November 8, 2021, from https://arxiv.org/abs/1804.02767.

Omnes, N., Bouillon, M., Fromentoux, G., & Grand, O. (2015). A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming challenges. *2015 18th International Conference on Intelligence in Next Generation Networks*, 64–69. https://doi.org/10.1109/ICIN.2015.7073808

ONU,(2015). WorldPopulation 2015 : Population Division. [en línea] Recuperado el 10 de Julio: https://www.un.org/es/sections/issues-depth/population/index.html

Perez-Sanz, F., Navarro, P. J., & Egea-Cortines, M. (2017). Plant phenomics: An overview of image acquisition technologies and image data analysis algorithms. *GigaScience*, *6*(11), 1–18. https://doi.org/10.1093/gigascience/gix092

Pound, M. P., Atkinson, J. A., Townsend, A. J., Wilson, M. H., Griffiths, M., Jackson, A. S., French, A. P. (2017). Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *GigaScience*, *6*(10), 1–10. https://doi.org/10.1093/gigascience/gix083

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing Explorations in the Microstructure of Cognition*, *1*(1), 194–281. Retrieved from http://portal.acm.org/citation.cfm?id=104279.104290

Story, D., Kacira, M., Kubota, C., Akoglu, A., & An, L. (2010). Lettuce calcium deficiency detection with machine vision computed plant features in controlled environments. *Computers and Electronics in Agriculture*, *74*(2), 238–243. https://doi.org/10.1016/j.compag.2010.08.010

Shelhamer, E., J. Long and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 640-651, 1 April 2017, doi: 10.1109/TPAMI.2016.2572683.

Shatnawi, A.M., Al-Bdour, G., Al-Qurran, R., & Al-Ayyoub, M. (2018). A comparative study of open source deep learning frameworks. 2018 9th International Conference on Information and Communication Systems (ICICS), 72-77. Retrieved November 9, 2021, from: https://www.semanticscholar.org/paper/A-comparative-study-of-open-source-deep-learning-Shatnawi-Al-Bdour/fb9f932b82a68cb3289beb9eed7094a41a8248b5

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556. Retrieved November 9, 2021, from: https://www.semanticscholar.org/paper/Very-Deep-Convolutional-Networks-for-Large-Scale-Simonyan-Zisserman/eb42cf88027de515750f230b23b1a057dc782108

Thaler, S., Menkovski, V., & Petkovic, M. (2018). *Deep Learning in Information Security*. Retrieved November 9, 2021, from http://arxiv.org/abs/1809.04332

Thompson, B. E. (1985). Seedling morphological. *Seedling Morphological Evaluation*, 59–71. Retrieved November 9, 2021, from: https://rngr.net/publications/evaluating/PDF.2003-10-27.0157

Tong, J. H., Li, J. B., & Jiang, H. Y. (2013). Machine vision techniques for the evaluation of seedling quality based on leaf area. *Biosystems Engineering*, *115*(3), 369–379. https://doi.org/10.1016/j.biosystemseng.2013.02.006

Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. (Jul 2002). An Efficient k-Means Clustering Algorithm:

Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 7, 881–892. DOI:https://doi.org/10.1109/TPAMI.2002.1017616

Turgut, K., Dutagaci, H., Galopin, G., Rousseau, D. (2020). Segmentation of Estructural Parts of Rosebush Plants with 3D Point–based Deep Learning Methods. arXiv. arXiv:2012.11489v1

Ubbens, J., Cieslak, M., Prusinkiewicz, P. *et al.* (2018) The use of plant models in deep learning: an application to leaf counting in rosette plants. *Plant Methods* **14,** 6. https://doi.org/10.1186/s13007-018-0273-z

Visa, S., Ramsay, B., Ralescu, A.L., & Knaap, E.V. (2011). Confusion Matrix-based Feature Selection. MAICS. Retrieve November 9, 2021, from: https://www.semanticscholar.org/paper/Confusion-Matrix-based-Feature-Selection-Visa-Ramsay/5ab58de53586bb52e17d3602439a879caa17ff03

Viera-Maza, G. (2017). *Procesamiento de Imágenes Usando OpenCV Aplicado en Raspeberry PI para la Clasificación del Cacao (Tesis de licenciatura en Ingeniería Mecánico-Eléctrica). Universidad de Piura, Facultad de Ingeniería.* Piura.

Kundu et. al., (2019). Understanding NFC-Net: a deep learning approach to word-level handwritten indic script recognition. Link: https://www.semanticscholar.org/paper/Understanding-NFC-Net%3A-a-deep-learning-approach-to-Kundu-Paul/19d43558ec1dd5a9cd6968eb8b37e384c7ab56b3

Klein, (2013). Coding the matrix – Linear Algebra through Computer Science Applications. Newtonian Press, Brown University.

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2021, February 22). Scaled-yolov4: Scaling cross stage partial network. arXiv.org. Retrieved November 9, 2021, from https://arxiv.org/abs/2011.08036.

Zhang, B., Mor, N., Kolb, J., Chan, D.S., Lutz, K., Allman, E., Wawrzynek, J., Lee, E.A., & Kubiatowicz, J.D. (2015). The Cloud is Not Enough: Saving IoT from the Cloud. HotStorage. Retrieved November 9, 2021, from:

https://www.semanticscholar.org/paper/The-Cloud-is-Not-Enough%3A-Saving-IoT-from-the-Cloud-Zhang-Mor/4bef7f35800919456555506ed112802c5680b4cd

96 Boards. (2019). 96Boards - Develop & Prototype on the Latest Arm Technology - 96Boards. Retrieved July 22, 2019, from https://www.96boards.org/

ANEXOS

Anexo I: Informe técnico: visita a vivero Agrogénesis

INFORME TECNICO: VISITA A VIVERO AGROGENESIS

Solicitante:	UNIVERSIDAD PRIVADA ANTENOR ORREGO
Domicilio Legal:	Av América Sur 3145, Trujillo 13008
Representante:	Sixto Ricardo Prado Gardini
Materia prima/Presentación :	Plantines Alcachofa
Servicio solicitado:	Construcción de línea base en marco del sub proyecto "Desarrollo e Implementación de un sistema robotizado para un control de calidad eficiente y continuo del crecimiento de los plantines en viveros industriales de la Región La Libertad Perú"
Fecha de asistencia técnica:	12/02/2019

ANTECEDENTES:

El día 31 de enero del presente en el laboratorio de automatización y robótica de la Universidad Particular Antenor Orrego se realizó una reunión de la primera visita de monitores de Concytec donde participaron Sr. Sixto Ricardo Prado Gardini investigador principal, representantes del CITEagroindustrial Chavimochic y los monitores de Concytec , reunión que tuvo como agenda: presentación del equipo técnico del sub proyecto y funciones que participarán, exposición del subproyecto, presentación del plan de adquisiciones, convenios de asociación, verificación de capacidades y firma de primera acta.

Luego de la reunión con los monitores, se vio necesario construir la línea base, en cual consistiría identificar todas las características en cuanto a la calidad de crecimiento de los plantínes (malformaciones al inicio de su crecimiento) así como identificar los controles de calidad visual durante la etapa de clasificación. Toda esta información recolectada servirá para que sea procesada por el sistema robótico – electrónico, en tal sentido el Sr. Sixto Prado Gardini investigador principal del sub proyecto, solicitó el apoyo del CITE para realizar una visita guiada por los especialistas del vivero Agrogénesis, teniendo como objetivo obtener información preliminar.

CVMG/JUB

ACTUADO:

El 12 de febrero a horas 9:30 am, en las instalaciones de la empresa Agronegocios Génesis se inició la reunión en la cual participaron: Sr. Lucio Olmos especialista del vivero, Sr. Sixto Prado Gardini investigador principal de la entidad solicitante (UPAO) y Jordan Ulloa Bello gestor tecnológico de la entidad asociada (CITEAGROINDUSTRIAL CHAVIMOCHIC), donde se presentó y explicó a la empresa el sub proyecto "Desarrollo e Implementación de un sistema robotizado para un control de calidad eficiente y continuo del crecimiento de los plantines en viveros industriales de la Región La Libertad Perú", teniendo respuesta afirmativa por parte de la empresa, mostrando interés y brindando la cooperación en la toma de información para la elaboración de línea base del sub proyecto en mención.

RESULTADOS:

a. Procedimiento Operacional:

En la figura N°1, se muestra el diagrama de flujo para la obtención de plantines de alcachofa de la empresa Agronegocios Génesis.

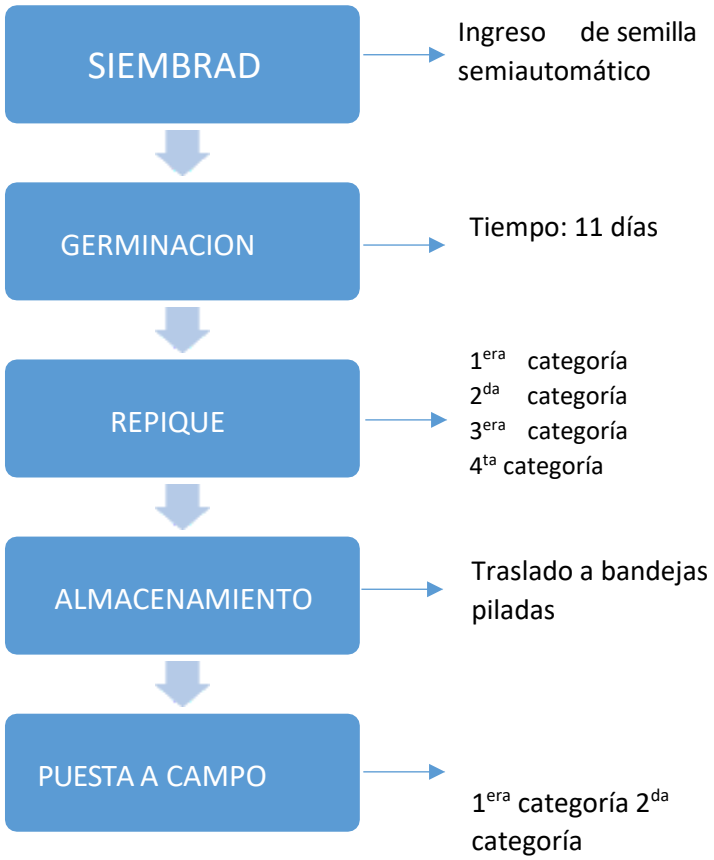


Figura 1: Diagrama de flujo para la obtención de plantines de alcachofa

CVMG/JUB

A continuación se describe la toma de información en cada una de las operaciones del diagrama de flujo presentada en la figura 1, identificando los problemas en el proceso de obtención de plantines de alcachofa.

Sembrado: Es realizado en un ambiente semiautomatizado, colocándose una semilla en cada pozo con sustrato en la bandeja.

Germinación: Durante los primeros 11 días dentro del proceso de germinación, los plantines de alcachofa son cubiertos por mantas de polipropileno, evitando la radiación excesiva o lluvias. El regado de los plantines se realiza generalmente de 1 a 2 veces al día (10 am – 3 pm) durante un tiempo de 15 min aproximadamente, siendo el regado por nebulización a un caudal promedio desde 37-40 litros/hora y a la vez se le hace un cruce de camas para que el riego sea homogéneo.

Figura 2: Riego por nebulización

Durante la germinación no es imprescindible que la testa (cubierta parte de la semilla) no se desprenda de la hoja, sin embargo, dependerá de que tan adherido este a la hoja, ya que puede producir cortes en esta. Mucho va a depender del vigor que tiene la semilla.

Figura 3: Testa en germinación

www.itp.gob.pe
Carretera Panamericana Norte Km 513
Provincia Virú, La Libertad, Perú
T: (51) (44) 525029

Las temperaturas adecuadas en el proceso de germinación son de 18-23°C con una humedad de 70%. La humedad en los plantines es categorizada según el peso de cada bandeja. Los plantines tienen un tiempo máximo de 30 días en el vivero.

En el cuadro N°1 se presentan los pesos de las bandejas vacias, bandejas con sustrato, bandeja con sustrato regada y el peso final de la bandeja con el plantin a los 30 días.

Cuadro N° 1 Pesos de las bandejas

OBJETO	PESO
BANDEJA VACIA	1000gr
BANDEJA CON SUSTRATO	1300gr
BANDEJA CON SUSTRATO REGADA	2800gr
PESO FINAL (bandeja con sustrato y plantin de alcachofa a los 30 días)	3500gr

El repique: Proceso que consiste en seleccionar, categorizar y ordenar los plantines en 1ra, 2da y 3ra categoría; desechando los plantines que no lograron su germinación (4ta categoría) o no cumplen con los parámetros adecuados para su siembra. Este proceso demanda una gran cantidad de jornales, siendo estos un promedio de 100 personas en época de pique, conllevando un costo alto para su realización, en la cual se muestra en la figura 5.

En un proceso productivo normal de plantines se tiene que el rendimiento de las categorías de primera es del 70%, de segunda 20%, de tercera se tiene un 5% y cuarta igual 5%, los cuales se muestran en la figura 4. Siendo los de tercera y cuarta categoría llega a la suma de un 10% que no cumplen con los requisitos de germinación.

Figura 4: Porcentaje de las categorías de plantines de alcachofas

CVMG/JUB

1era Categoría 2da Categoría 3era Categoría

Figura 5: Categorías de los plantines de alcachofa

En el repique, un plantin bueno tiene las siguientes características:

a. En el correcto desarrollo de un plantin se toma en cuenta dos medidas: FOLIAR (Color, forma de la hoja) comúnmente llamada "Orejita de conejo" y RADICULAR (radio máximo del plantin), la cual se muestra en la figura 6.

Figura 6: Correcto desarrollo de un plantin

En el cuadro 2, se presentan las medidas de un plantin con un correcto desarrollo (Categoría: primera y segunda)

Cuadro N° 2 Medición de con vernier digital a plantines de alcachofa

DESCRIPCION	1era CATEGORIA	2da CATEGORIA
Tallo – hoja	49.51	33.98
Ancho de hoja	23.41	23.09
Altura	53.47	34.01
Ancho de hoja superior	20.19	13.96

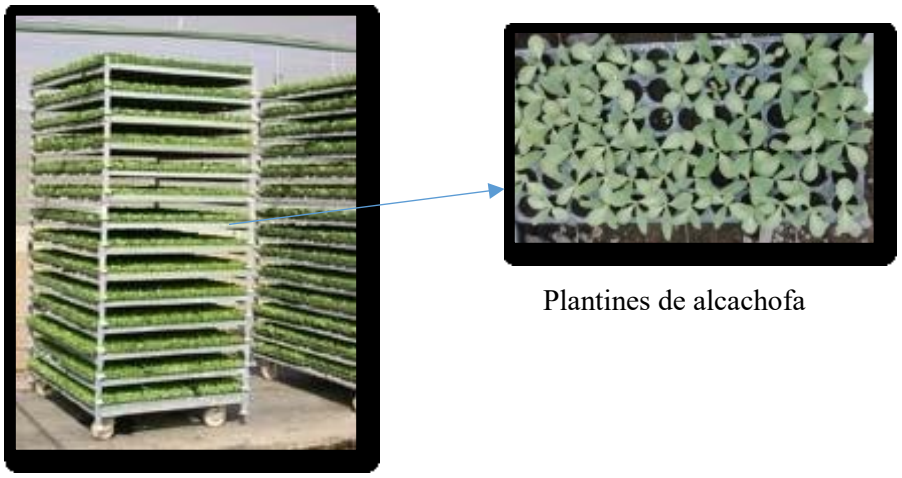
b. Las raíces del plantin deben cubrir por lo menos entre 60% y 70% el cono de sustrato.

www.itp.gob.pe
Carretera Panamericana Norte Km 513
Provincia Virú, La Libertad, Perú
T: (51) (44) 525029

c. El cono de sustrato debe mantener su forma cónica, donde se adhieren las raíces a este de forma fuerte. Un mal plantin presentara un cono de sustrato débil.

Figura 7: Medición de plantin de alcachofa

Almacenamiento: Proceso que consiste en acumular los plantines según su categoría (1^{era} y 2da) en las bandejas hasta el tiempo de puesta en campo, la cual se muestra en la figura 8.



Plantines de alcachofa

Figura 8: Soporte apilables con los plantines

Puesta a campo: los plantines de 1^{era} y 2da categoría son trasladados a campo, la cual se muestra en la figura 9.

www.itp.gob.pe
Carretera Panamericana Norte Km 513
Provincia Virú, La Libertad, Perú
T: (51) (44) 525029

Figura 9: Traspaso de plantin a campo

CONCLUSIONES:

- Se obtuvo el diagrama productivo de los plantines de alcachofa.
- Se pudo determinar los aspectos visuales de calidad según su categoría como forma de hoja, raíz y color del plantin.
- Se pudo determinar parámetros productivos según su categoría como tamaño, tiempo, datos biométricos y categorías de plantines.

Atentamente,

Ing. Jordan Ulloa Bello
Auxiliar I+D+i
CITEagroindustrial Chavimochic

CVMG/JUB

Anexos II: Codificación en C+L del modelo del plantín

Codificación en C+L de la versión 6 del modelo del plantín sintético de alcachofa. El resto de los archivos necesarios para renderizar el modelo en el LStudio, se encuentra en el CD con el nombre de "Plantin – Sistema L".

```
#define STEPS    100       /*Age will modify seedling development*/
#define rnd_n   3703

#define dt 0.05
#define l_i 1 /*leaf increment per decomposition*/
#define f_l 4 /*first leaves*/
#define PL 1.20000   /*1.2*/
#define MAX_INT_LEN 0.50000
#define t_max_cotiledon 3.4

Lsystem: 1
derivation length: STEPS
Axiom: -(5)#(0.15)A(0,0)

A(t,l) --> A(t+dt,l+l_i)
I(t)   --> I(t+dt)
L(t)   --> L(t+dt) /*first-stage leaf*/
T(t)   --> T(t+dt) /*second&third-stage leaf*/

decomposition
A(t,l) : {t=t-PL;} t>0 && l<26  {srand(rnd_n);
                                 fsfang = 130 + ran(90);
                                 fssang = 60 + ran(40);}
                                -->  I(t)[L(t)]/(fsfang)[L(t)]/(fssang)A(t,l)
/*(noQ)*/

A(t,l) : {t=t-PL;} t>0 && l>79 && l<81       {ssfang = 120 + ran(120);
                                 sssang = ran(360);}
                                --> I(t)[T(t)]/(ssfang)[T(t)]/(sssang)A(t,l)

A(t,l) : {t=t-PL;} t>0 && l>900              {tsfang = ran(360);}
                                --> I(t)[T(t)]/(tsfang)A(t,l)
```

```
homomorphism
I(t) --> F(MAX_INT_LEN*func(1,t/25))

L(t) : 1 {if(t>t_max_cotiledon){t=t_max_cotiledon;}
         len = func(2,t/10); /*10*/
         wid = func(3,t/12); /*12*/
         ang = -ran(45) + 26*func(4,t/30);
         col = 32+floor(21*func(5,t/20)); /*20*/
         plhspin1 = ran(35);}
       --> &(ang);(col)!(0.025)[#(0.02)F(0.01)&(60)/(plhspin1)~l(wid,len,len)]

T(t) : 1 {len = func(6,t/13); /*10*/
         wid = func(7,t/16); /*14*/
         ang = 6*func(4,t/33); /*30*/
         col = 32+floor(21*func(5,t/20)); /*20*/
         lspinoffs = ran(20) - 20;
         inc = lspinoffs+28+36*func(8,t/5); /*5*/
         plhspin2 = ran(20);}
       --> &(ang);(col)!(0.025)[#(0.02)F(0.01)&(inc)/(plhspin2)~s(wid,len,len)]
endlsystem
```

Anexos III: Código de la síntesis de plantines con Robotic Process Automation

Código en Python para la síntesis de plantines v6 haciendo uso del RPA. Para que corra este programa se debe tener cuidado en asegurarse de que la ventana donde se ubica el código en C+L en el LStudio esté detrás de la ventana del programa que se use para ejecutar el programa con nombre rpa_v5.py (includído también en el CD carpeta "síntesis_plantin_rpa").

```
# -*- coding: utf-8 -*-
"""

Seedling generating RPA

for seedling model v6

Created on Mon Feb 15 16:51:32 2021

@author: Erick Fiestas
"""

import pyautogui
import time
import numpy as np
import shutil
import os
from random import randint
from random import seed
#import cv2

original_path           =           r'C:\Users\hjara\OneDrive\Maestría\thesis
works\tesis\lsystems-tests\artichocke6\plant.bmp'
target_path = r'C:\Users\hjara\OneDrive\Maestría\thesis works\tesis\lsystems-
tests\artichocke6\images_db' #it needs namefile yet

steps_coord = np.array([225,145])
rnd_d_coord = np.array([264,165])

steps_inter_gq = np.array([105,118])
steps_inter_aq = np.array([85,95])
steps_inter_bq = np.array([40,80])
rnd_d_inter = np.array([0,9999])


def updating_Ls_parameters(steps,rnd_d):
    #this procedure is for updating L-system parameters from artichoke model
    time.sleep(0.2)
    pyautogui.click(steps_coord[0], steps_coord[1])
    pyautogui.press(['backspace','backspace','backspace'])
    if(len(str(steps))==2):
        pyautogui.press(['0',str(steps)[0],str(steps)[1]])
    else:
        pyautogui.press([str(steps)[0],str(steps)[1],str(steps)[2]])

    pyautogui.click(rnd_d_coord[0], rnd_d_coord[1])
```

```
    pyautogui.press(['backspace','backspace','backspace','backspace'])

    if(rnd_d<10): #three zeros
        pyautogui.press(['0','0','0',str(rnd_d)])
    if(rnd_d>9 and rnd_d<100): #two zeros
        pyautogui.press(['0','0',str(rnd_d)[0],str(rnd_d)[1]])
    if(rnd_d>99 and rnd_d<1000): #one zero
        pyautogui.press(['0',str(rnd_d)[0],str(rnd_d)[1],str(rnd_d)[2]])
    if(rnd_d>999): #without zeros

pyautogui.press([str(rnd_d)[0],str(rnd_d)[1],str(rnd_d)[2],str(rnd_d)[3]])

def generate_random_number(min_n,max_n):
    num = randint(min_n, max_n)
    return num

def generate_seedling_parameters(parameter,seedling_type):
    print(["Generating seedling parameters"])
    if(parameter == "steps"):
        print(parameter)
        if(seedling_type==1): #good quality
            n = generate_random_number(steps_inter_gq[0],steps_inter_gq[1])
            print(n)
            return n
        if(seedling_type==2): #average quality
            n = generate_random_number(steps_inter_aq[0],steps_inter_aq[1])
            print(n)
            return n
        if(seedling_type==3): #bad quality
            n = generate_random_number(steps_inter_bq[0],steps_inter_bq[1])
            print(n)
            return n

    if(parameter == "rnd_d"):
        print(parameter)
        n = generate_random_number(rnd_d_inter[0],rnd_d_inter[1])
        print(n)
        return n

def qual(q):
    if(q=="good"):
        return 1
    if(q=="average"):
        return 2
    if(q=="bad"):
        return 3

pyautogui.hotkey('alt', 'tab') # Press the alt+tab hotkey combination.
time.sleep(0.2)
seed(1104)

#have to generate three different kinds of seelings on specific folders
qualities = ["bad"] #"good","average","bad"]
for q in qualities:
    for i in range(100):#10 good quality images
        seedling_type = qual(q) #main features of seedlings
        steps = generate_seedling_parameters("steps",seedling_type)
        rnd_d = generate_seedling_parameters("rnd_d",seedling_type)
```

```
        updating_Ls_parameters(steps,rnd_d)

        pyautogui.hotkey('ctrl', 'g') # Press the Ctrl-g hotkey combination.
        time.sleep(1.1)

        #Sequence for generating plant image
        ixcoord = 204
        iycoord = 505
        pyautogui.moveTo(ixcoord, iycoord) #548, 389) # Move the mouse to XY
coordinates.
        pyautogui.click(button='right')
        time.sleep(0.5)
        pyautogui.click(312,595)
        time.sleep(0.5)
        pyautogui.click(548,598)
        time.sleep(0.5)
        pyautogui.click(752,652)
        time.sleep(0.5)
        pyautogui.click(707,340)
        time.sleep(0.5)
        pyautogui.hotkey('ctrl', 's') # Press the Ctrl-g hotkey combination.

        time.sleep(0.5)

        #new_tp = target_path + "\" + "lala.bmp"
        fl_name = str(q) + "_" + str(i) + ".bmp"
        new_tp = os.path.join(target_path,fl_name)
        #image = cv2.imread(original_path)
        #cv2.imwrite(new_tp,image)
        shutil.copyfile(original_path, new_tp)
```

Anexos IV: Código para la creación de grupos de imágenes de plantines

Código en Python del programa "créate_data_v3.py" para la creación de imágenes de grupos de plantines, así mismo para el formateo de éstas según lo establecido en la Sección 4.1.3.

```
# -*- coding: utf-8 -*-
"""
CREATE DATA v2
---------------
Create files for training yolov3 with Darknet

Update:
    produce rotated seedling images
    adaptation to darknet dataset format

Created on Tue Feb 23 16:06:14 2021

@author: Erick Fiestas
"""

import numpy as np
import cv2
import os
import imageio
from PIL import Image
from random import randint
from random import seed
from datetime import datetime
import math

"""

              txt file
                  image.jpg class tx,ty,tw,th

              classes.txt
                  name1
                  name2
                  ...
                  namen
"""

img_ds = r'D:\databases\artichoke seedlings\artichokev6_ds\3kseedlings'
seedgrou_fpath                    =                    r'D:\databases\artichoke
seedlings\artichokev6_ds\seedling_groups_datasets\1k-rot_bueno_regular-
seedlings'
h_offset = 0
w_offset = 0
debug_mode = False
rotation_angles = ([0,90,180,270])

def rgb2binary(img,background_color):
    if debug_mode: print("RGB2BINARY function")
    gsimage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    if debug_mode:
        cv2.imshow("Gray image",gsimage)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    if       background_color       ==       0:       ret,binaryimage       =
cv2.threshold(gsimage,40,255,cv2.THRESH_BINARY) #black background
    if       background_color       ==       1:       ret,binaryimage       =
cv2.threshold(gsimage,170,255,cv2.THRESH_BINARY_INV) #different from black

    if debug_mode:
        cv2.imshow("binary image",binaryimage)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    return np.array(binaryimage)

def cropimage(image,x_min,x_max,y_min,y_max):
    #image = imageio.imread(path)
    #image = np.array(image)

    # crop the image using array slices -- it's a NumPy array
    # after all!
    cropped = image[y_min:y_max,x_min:x_max] #443:720, 403:1082]
    if debug_mode:
        cv2.imshow("cropped", cropped)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return np.array(cropped)

def applymask(simg,bimg):

    #simg = imageio.imread(spath)
    #mimg   =   dilateimage(imageio.imread(mpath))   #dilation   for   avoiding
segmentation and leaf pixels errors
    #mimg = imageio.imread(mpath)
    #bimg = cv2.cvtColor(bimg, cv2.COLOR_GRAY2RGB)
    simg, mimg = np.array(simg), np.array(bimg)
    return cv2.bitwise_and(simg,simg,mask=mimg)

def getwh(bimg):
    if debug_mode: print("GETTING WIDTH and WEIGHT")
    height,width = bimg.shape #2dimensions

    for h in range(height):
        for w in range(width):
            if bimg[h,w] == 255:
                hmax = h + h_offset

    for w in range(width):
        for h in range(height):
            if bimg[h,w] == 255:
                wmax = w + w_offset

    for h in range(height-1,-1,-1):
        for w in range(width-1,-1,-1):
            if bimg[h,w] == 255:
                hmin = h - h_offset
```

```
    for w in range(width-1,-1,-1):
        for h in range(height-1,-1,-1):
            if bimg[h,w] == 255:
                wmin = w - w_offset

    if debug_mode:
        print("hmin: " + str(hmin))
        print("wmin: " + str(wmin))
        print("hmax: " + str(hmax))
        print("wmax: " + str(wmax))

    ih = hmax - hmin
    iw = wmax - wmin

    if debug_mode:
        print("witdh = " + str(iw))
        print("height = " + str(ih))

    return iw,ih,wmin,wmax,hmin,hmax

def imageremix(base_img,mask_img,rv,cv): #row and column values
    if debug_mode: print("Mixing 2 images")
    brv = rv #base image row value
    bcv = cv
    bimg = np.array(base_img)
    mimg = np.array(mask_img)
    mimg_r, mimg_c, _ = mimg.shape
    bimg_r, bimg_c, _ = bimg.shape
    limitflag_c = True
    #limitflag_r = True

    if debug_mode:
        print("mask image row and column: " + str(mimg_r) + ", " + str(mimg_c))
        print("base image row and column: " + str(bimg_r) + ", " + str(bimg_c))

    for r in range(mimg_r):
        for c in range(mimg_c):
            if mimg[r,c,0] > 0: #different from zero
                if limitflag_c:
                    bimg[brv,bcv,:] = mimg[r,c,:]
            bcv += 1
            if bcv >= bimg_c: limitflag_c = False
            #else: limitflag_c = True
        brv += 1
        bcv = cv
        if brv >= bimg_r: break

    return bimg

def qual(q,m):
    if m == 0:
        if(q=="2"):
            return "good"
        if(q=="1"):
            return "average"
        if(q=="0"):
            return "bad"
    if m == 1:
        if(q=="good"):
```

```
            return "2"
        if(q=="average"):
            return "1"
        if(q=="bad"):
            return "0"

def generate_random_number(min_n,max_n):
    num = randint(min_n, max_n)
    return num

def generate_seedling_name():
    #type of seedling
    quality = qual(str(generate_random_number(1,2)),0)
    #number of seedling: between 0-1000
    sn = generate_random_number(0,999)
    #seedling name
    sname = quality + "_" + str(sn) + ".png"
    return sname

def get_position_reference(sp):
    #0:98,102 1:201,102 2:303,102
    #3:98,195 4:201,195 5:302,195
    #6:98,302 7:201,302 8:302,302
    if sp==0:
        return 98,102
    if sp==1:
        return 201,102
    if sp==2:
        return 302,102
    if sp==3:
        return 98,195
    if sp==4:
        return 201,195
    if sp==5:
        return 302,195
    if sp==6:
        return 98,302
    if sp==7:
        return 201,302
    if sp==8:
        return 302,302

def get_seedling_coordinates(img,sp): #seedling position as the input argument
    if debug_mode: print("Getting seedling coordinates")
    r,c = get_position_reference(sp)
    imgr,imgc,_ = img.shape

    src = round(r-imgr/2) #seedling row coordinate
    if src < 0: src = 0
    scc = round(c-imgc/2) #seedling column coordinate
    if scc < 0: scc = 0

    if debug_mode: print("src: " + str(src) + ", scc: " + str(scc))
    return src,scc

def generate_random_seedling_position(sepo_collection):
    if debug_mode: print("Generation random seedling positions")
    p_was_found = False
    while (p_was_found == False):
```

```
        p_is_busy = False
        pp = generate_random_number(0,8) #posible position
        if debug_mode: print(pp)
        #print(sepo_collection)
        for p in range(sepo_collection.shape[0]):
            if sepo_collection[p] == pp:
                p_is_busy = True
                if debug_mode: print(str(sepo_collection[p])+", "+str(pp))
        if p_is_busy == False:
            p_was_found = True

    for p in range(sepo_collection.shape[0]):
        if sepo_collection[p] == -1:
            sepo_collection[p] = pp
            break
    if debug_mode: print("New position: " + str(pp))
    return pp,sepo_collection

def save_image(final_img, seedgrou_fpath):
    if debug_mode: print("Saving image")
    #generate file name
    fname  =  "seedling_group_"  +  datetime.now().strftime("%d-%m-%Y_%H-%M-
%S.%f") + ".jpg"
    fnamepath = os.path.join(seedgrou_fpath,fname)
    #save image
    final_img.save(fnamepath)
    return fname

def generate_dataset_txt_files(siname,annotations_vec,seedgrou_fpath):
    if debug_mode: print("Generating txt files for seedling groups dataset")

    content = ""
    for p in range(annotations_vec.shape[0]):
        #content      +=      str(annotations_vec[p,0])      +      ","      +
str(annotations_vec[p,1])  +  ","  +  str(annotations_vec[p,2])  +  ","  +
str(annotations_vec[p,3]) + "," + str(annotations_vec[p,4]) + " "

        #content      +=      str(annotations_vec[p,0])      +      ","      +
str(annotations_vec[p,1])  +  ","  +  str(annotations_vec[p,2])  +  ","  +
str(annotations_vec[p,3]) + "," + str(annotations_vec[p,4]) + " "
        center_x   =   annotations_vec[p,0]   +   (annotations_vec[p,2]   -
annotations_vec[p,0])/2
        center_x /= 416 #normalization
        center_y   =   annotations_vec[p,1]   +   (annotations_vec[p,3]   -
annotations_vec[p,1])/2
        center_y /= 416
        hi = (annotations_vec[p,2] - annotations_vec[p,0])/416
        wi = (annotations_vec[p,3] - annotations_vec[p,1])/416
        content += str(annotations_vec[p,4]-1) + " " + str(center_x) + " " +
str(center_y) + " " + str(wi) + str(" ") + str(hi) + "\n"

    filepath = os.path.join(seedgrou_fpath, siname[:-4]+str(".txt")) #-4 in
order to get .png away
    f=open(filepath,"w")
    f.write(content)
    f.close()

def create_img_fmk(img):
    h,w = img.shape[:2]
```

```
    hinc = 30 #up and down sides height increment
    nif = np.zeros([h+hinc,w,3]).astype(np.uint8) #black framework
    return nif

def rotateimg(img,fmk,wmin,wmax,hmin,hmax,ang):
    #img is seedling image, ang in radiants

    #row = round(fmk.shape[0]/2 - img.shape[0]/2)
    #col = 0
    imapixpos = []
    #imgres = []
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            #r = ((r+row)**2+(c+col)**2)**0.5
            imapixpos.append(complex(r,c)) #image pixel positions
            #imgres.append(r * complex(math.cos(0),math.sin(0)))

    #Center the image at the origin of cartesian corrdinates
    imapixpos = np.array(imapixpos) #Complex space array >>>>> x1 + yi1, x2 +
yi2...
    imapixposs = imapixpos - complex(img.shape[0]/2,img.shape[1]/2)
    #imgres = np.array(imgres)
    #rotate image
    imgres = imapixposs * complex(math.cos(ang),math.sin(ang))
    if debug_mode:
        print("img: " + str(img.shape))
        print("fmk: " + str(fmk.shape))
    #transalate back to the center or 4th quadrant
    imgres += complex(fmk.shape[0]/2,fmk.shape[1]/2)

    #imgres += complex(350,270) #offset in order to see rotates image
    if debug_mode: print(imgres)

    #Convert into pixel coordinates
    for pos in range(imapixpos.shape[0]):
        if   imgres[pos].real   >   0   and   imgres[pos].imag   >   0   and
int(imgres[pos].real) < fmk.shape[0]:

#print(round(imgres[pos].real),round(imgres[pos].imag),round(imapixpos[pos].r
eal) - row,round(imapixpos[pos].imag) - col)
            fmk[int(imgres[pos].real),int(imgres[pos].imag),:]              =
img[int(imapixpos[pos].real),int(imapixpos[pos].imag),:]
            #fmk[int(imgres[pos].real),int(imgres[pos].imag),1]             =
int(img[int(imapixpos[pos].real),int(imapixpos[pos].imag),1])
            #fmk[int(imgres[pos].real),int(imgres[pos].imag),2]             =
int(img[int(imapixpos[pos].real),int(imapixpos[pos].imag),2])
            #fmk[int(imgres[pos].real),int(imgres[pos].imag),0]             =
int(fmk[int(imgres[pos].real),int(imgres[pos].imag),0])
            #fmk[int(imgres[pos].real),int(imgres[pos].imag),1]             =
int(fmk[int(imgres[pos].real),int(imgres[pos].imag),1])
            #fmk[int(imgres[pos].real),int(imgres[pos].imag),2]             =
int(fmk[int(imgres[pos].real),int(imgres[pos].imag),2])

            #print(img[int(imapixpos[pos].real),int(imapixpos[pos].imag),:])
            #print(fmk[int(imgres[pos].real),int(imgres[pos].imag),:])

    fmk.astype(np.uint8)

    return fmk, imgres
```

```
def rotateseedling(image,ang,wmin,wmax,hmin,hmax):
    #crop seedling out and paste it onto bigger framework
    cimg = cropimage(image,wmin,wmax,hmin,hmax)
    fmk = create_img_fmk(image)
    #row = fmk.shape(0)/2 - cimg.shape(0)/2
    #col = 0
    if debug_mode: print(fmk)
    irotated,imgres = rotateimg(cimg,fmk,wmin,wmax,hmin,hmax,ang*math.pi/180)

    """
    fimg = imageremix(fmk,cimg,row,col) #to avoid losing tip leaves wehn
rotating seedling image


    # grab the dimensions of the image and calculate the center of the image
    (h, w) = image.shape[:2]
    (cX, cY) = (w // 2, h // 2)
    # rotate our image by ang degrees around the center of the image
    M = cv2.getRotationMatrix2D((cX, cY), ang, 1.0)
    irotated = cv2.warpAffine(image, M, (w, h))
    #cv2.imshow("Rotated by 45 Degrees", irotated)
    """

    return irotated, imgres

num_seedling_groups = 800
seed(1577)

#siname_vec = []

for t in range(num_seedling_groups):
    seedling_num = generate_random_number(1,9)
    wmin_vec = []
    wmax_vec = []
    hmin_vec = []
    hmax_vec = []
    seco_r = []
    seco_c = []
    qualities = []
    tw_vec = []
    th_vec = []
    sepo_collection = np.array([-1,-1,-1,-1,-1,-1,-1,-1,-1]) #To know what
seedling position is occupied. "-1" means free position

    #read the reference image
    final_img = Image.open("size_ref.jpg")

    for seedl_n in range(seedling_num):

        #for seedling in os.listdir(img_ds):
        seedling = generate_seedling_name() #"good_2.png"
        if debug_mode: print(seedling)
        seedlingpath = os.path.join(img_ds, seedling)
        image = imageio.imread(seedlingpath)
        #image = Image.open(seedlingpath)
        #image size
        target_width = 400
```

```
        size=(target_width,round(target_width*0.51)) #image size proportion
720/1424 = 0.51
        #resize image
        rsimg = cv2.resize(image,size) #this is really important to facilitate
seedling manipulation

        #rsimg = rotateimg(rsimg,randint(0, 360))

        bimage = rgb2binary(rsimg,1)
        tw,th,wmin,wmax,hmin,hmax = getwh(bimage)

        mimg = applymask(rsimg,bimage)
        if debug_mode:
            cv2.imshow("masked image", mimg)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        mmimg,imgres                                                    =
rotateseedling(mimg,rotation_angles[randint(0,3)],wmin,wmax,hmin,hmax)
        if debug_mode:
            cv2.imshow("rotated image", mmimg)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        #It is necessary to binarize the image again and to apply getwh in
order to crop ratated image
        bimage = rgb2binary(mmimg,0)
        tw,th,wmin,wmax,hmin,hmax = getwh(bimage)
        wmin_vec.append(wmin)
        wmax_vec.append(wmax)
        hmin_vec.append(hmin)
        hmax_vec.append(hmax)
        qualities.append(int(qual(seedling.split("_")[0],1)))

        #Apply masking to rotated image
        mimg = applymask(mmimg,bimage)
        if debug_mode:
            cv2.imshow("masked imageeeeeeeeeeeeee", mimg)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        #Crop masked image
        cimg = cropimage(mimg,wmin,wmax,hmin,hmax)

        #image size
        #size=(224,224)
        #resize image
        #rsimg = simg.resize(size) #this is really important to facilitate
seedling manipulation

        #nimg = rotateimg(nimg,randint(0, 360))

        simg = Image.fromarray(cimg)

        seedling_position,sepo_collection                               =
generate_random_seedling_position(sepo_collection)
        secor, secoc = get_seedling_coordinates(cimg,seedling_position)
        tw_vec.append(tw+secoc) #seedling image coordinates in final + width
and height
        th_vec.append(th+secor)
```

```
        seco_r.append(secor)
        seco_c.append(secoc)
        final_img = imageremix(final_img,simg,secor,secoc) #rsimg above refim
applying sort of masking
        #final2_img = imageremix(final_img,rsimg,20,20)
        final_img = Image.fromarray(final_img)

    if debug_mode: final_img.show()
    wmin_vec = np.array(wmin_vec)
    wmax_vec = np.array(wmax_vec)
    hmin_vec = np.array(hmin_vec)
    hmax_vec = np.array(hmax_vec)
    seco_r = np.array(seco_r)
    seco_c = np.array(seco_c)
    qualities = np.array(qualities)
    tw_vec = np.array(tw_vec)
    th_vec = np.array(th_vec)

    annotations_vec = np.column_stack((seco_c,seco_r))
    annotations_vec = np.column_stack((annotations_vec,tw_vec))
    annotations_vec = np.column_stack((annotations_vec,th_vec))
    annotations_vec = np.column_stack((annotations_vec,qualities))


    #Save image: and get seedling image name
    siname = save_image(final_img, seedgrou_fpath)

    #siname_vec.append(siname[:-4])

    if debug_mode: print("file generated: " + siname + " on " + seedgrou_fpath)

    #Generate txt files
    generate_dataset_txt_files(siname,annotations_vec,seedgrou_fpath)
```

Anexos V: Código para graficar PC1 vs PC2

Se presenta el programa llamado “distribution_v4.py” que permite graficar PC1 *vs* PC2.

Se puede encontrar también en la carpeta PROGRAMAS en el CD adjunto.

```
# -*- coding: utf-8 -*-
"""
Analyzing phenotyping distribution for virtual seedlings

+ This program if for analyzing seedling image dataset from image folder that
is as a result of synthetizing artichoke seedlings
+ In this way it is expected to catch images out according to the seedling
number
matched with its name

Created on Tue Feb  9 10:37:10 2021

@author: Erick Fiestas
"""

from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import os
import imageio
import cv2

#bipath  =  r'C:\Users\hjara\OneDrive\Maestría\thesis  works\tesis\lsystems-
tests\artichocke6\images_db'
bipath = r'D:\databases\artichoke seedlings\artichokev6_ds\3kseedlings'
#D:\databases\artichoke seedlings\3kseednling'

def seedling_ql(sn):
    if sn[0] == 'a':
        return 1
    if sn[0] == 'g':
        return 2
    if sn[0] == 'b':
        return 0

def load_training_imgs(path,ll,rl,q): #left limit and rigth limit: [ll:rl]

    width, height = 224, 224
    dictionary = [] #[index][name][quality]
    tr_ds = []
    label_ds = []
    c = 0
    sc = 0
    for seedling in os.listdir(bipath):
        if seedling[0] == q:
            if c >= ll and c <= rl:
                ip = os.path.join(bipath,seedling)
                dim = (width, height)
                image                                                    =
cv2.resize(imageio.imread(ip),dim,interpolation=cv2.INTER_AREA)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = np.array(image).reshape(3,width,height)
```

```
                tr_ds.append(image)
                label_ds.append(seedling_ql(seedling))
                dic = ([c],[seedling],[seedling[0]])
                dictionary.append(dic)
                sc += 1
            c += 1

    label_ds = np.stack(label_ds)
    tr_ds = np.stack(tr_ds)
    tr_ds                                                                    =
tr_ds.reshape(tr_ds.shape[0],tr_ds.shape[2],tr_ds.shape[3],tr_ds.shape[1])
    dictionary = np.stack(dictionary)
    print("Seedling generation counter: " + str(sc) + " units")

    return [tr_ds, label_ds, dictionary]

qualities = ["b","a","g"]
for n in range(1): #6
    data = []
    lbl = []
    dicti = []
    for q in qualities:
        ll = n*1000
        rl = ll + 1000 - 1
        dats,label,dic = load_training_imgs(bipath,ll,rl,q)
        data.append(dats)
        lbl.append(label)
        dicti.append(dic)
    data = np.array(data)
    lbl = np.array(lbl).flatten()
    dicti = np.array(dicti).reshape(3000,3,1)
    print(data.shape)

    #taking a sample image to view
    #Remember image is in the form of numpy array.
    data = data.reshape(3000,150528) #224,224,3)
    #plt.imshow(data[0,:])

    pca = PCA(2) # we need 2 principal components.
    converted_data = pca.fit_transform(data)
    print(converted_data.shape)

    consider_seedl = []
    f,c = converted_data.shape
    for i in range(f):
        if converted_data[i,0] > 0 and lbl[i] == 1:
            consider_seedl.append(converted_data[i,0])
            print(converted_data[i,0])
            print("lbl[i], i = " + str(i) + ". Name file: " + dicti[i,1,0])

    consider_seedl = np.array(consider_seedl)

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize = (10,6))
    c_map = plt.cm.get_cmap('jet', 3)
    plt.scatter(converted_data[:, 0], converted_data[:, 1], s = 15, cmap =
c_map , c = lbl)
    plt.colorbar()
    plt.xlabel('PC-1') , plt.ylabel('PC-2')
```

```
plt.show()
```

Anexos VI: Registro de 48 plantines reales

48 plantines reales como *dataset* de validación para los modelos propuestos. Estos plantines muestreados en el laboratorio LABIMN están clasificados según las medidas oficiales de la agroindustrial Agrogénesis mostradas en la Tabla 1, que corresponden a las medidas de Altura, Largo y Ancho de la hoja de plantín de alcachofa. PALA significa: Promedio de Altura, Largo y Ancho. La casilla del plantín 38 (correspondiente al número del plantín, ver carpeta PLANTINET) no fue detectado por el Yolov3.

ID	NumFoto	Altura	Largo hoja	Ancho hoja	Largo cotiledona	ncho cotiledor	Calidad	PALA
1	1	46.51	42.85	15.2	29.3	16.66	2	34.8533333
2	2	33.95	34.47	15	46.84	23.23	1	27.8066667
3	3	20.3	8.9	3.08	34.72	21.75	0	10.76
4	5	25.01	18.69	5.73	35.78	22.98	0	16.4766667
5	6	24.85	20.1	19.2	40.05	23.68	0	21.3833333
6	7	22.22	18.83	3.95	39.13	20.89	0	15
7	8	36.82	27.33	11.31	36.95	20.97	1	25.1533333
8	9	32.92	22.51	13.4	35.11	22.27	1	22.9433333
9	12	19.77	10.41	4.9	29.85	21.23	0	11.6933333
10	13	46.69	40.05	16.21	45.28	25.82	2	34.3166667
11	15	35.06	24.77	10.12	36.08	23.05	1	23.3166667
12	17	38.46	32.57	11.1	39.09	25.1	1	27.3766667
13	18	34.76	23.92	8.82	32.45	23.86	1	22.5
14	19	26.7	26.3	14.2	15.67	21.42	1	22.4
15	20	55.34	46.79	14.43	40.77	23.14	2	38.8533333
16	22	15.68	11.15	6.5	31.07	19.42	0	11.11
17	23	1	1	1	1	1	0	1
18	24	21.27	13.41	6.32	28.13	21.38	0	13.6666667
19	25	14.66	11.73	4.2	24.79	18.02	0	10.1966667
20	26	24.7	16.07	4.72	33.78	22.75	0	15.1633333
21	27	49.85	46.86	17.35	47.16	24.57	2	38.02
22	28	49.1	46.85	17.67	39.16	24.08	2	37.8733333
23	30	52.54	44.06	15.86	46.23	15.39	2	37.4866667
24	31	48.9	41.09	17.8	43.8	29.53	2	35.93
25	34	49.28	42.82	16.1	36.5	45.3	2	36.0666667
26	35	31.59	22.06	12.76	35.09	24.76	1	22.1366667
27	38	31.03	24.5	12.7	37	21.93	1	22.7433333
28	39	59.6	50.22	18.71	44.35	26.99	2	42.8433333
29	41	51.97	35.73	14.95	41.37	24.67	2	34.2166667
30	43	43.9	37.13	15.32	40	23.14	2	32.1166667
31	44	46.37	25.75	6.8	44.09	23.1	1	26.3066667
32	45	47.1	37.23	13.56	38.54	26.45	2	32.63
33	46	54.82	38.2	17.34	46.45	27.92	2	36.7866667
34	48	49.98	46.44	15.17	41.37	25.08	2	37.1966667
35	49	54.64	48.03	15.24	40.57	25.51	2	39.3033333
36	50	42.54	37.32	17.1	39.98	23.83	2	32.32
37	51	49.81	49.92	17.6	51.68	25.71	2	39.11
38	55	1	1	1	15.47	9.62	0	1
39	56	1.8	1.5	1	21.73	18.78	0	1.43333333
40	58	38.9	34.3	10.3	30.3	26.96	1	27.8333333
41	59	51.32	43.8	13.83	37.85	27.5	2	36.3166667
42	61	52	42	18	15	14.1	2	37.3333333
43	62	45	50.2	17.56	50.04	27.93	2	37.5866667
44	64	32.4	25.72	13.5	32.4	13.1	1	23.8733333
45	65	22.1	20.3	12.4	33.9	23.32	0	18.2666667
46	66	32.16	21.65	15.7	28.4	22.9	1	23.17
47	71	54.8	42.1	15.8	35.4	27.2	2	37.5666667
48	72	33.7	31.13	12.42	37.6	22.1	1	25.75

Anexos VII: Código para entrenamiento, testeo y validación del PCA-Kmeans

A continuación, el programa "pcakmeans_model_v2.1.py" usado para el entrenamiento, testeo y validación del modelo basado en PCA y Kmeans. Programa encontrado en la carpeta PROGRAMAS en el CD adjunto.

```
# -*- coding: utf-8 -*-
"""
PCA-kmeans model v2.1
-------------------

includes all testing images for kmeans fitting

This is the model based on pca and kmeans for classifying a seedling image dataset
The results are basically:

confusion matrix
precision
sensitivity
recall
...

Created on Wed Jun 23 18:13:45 2021

@author: Erick Fiestas
"""


from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import os
import imageio
import cv2
import pickle
import scipy.stats
from datetime import datetime

############## TRAINING DATA ##############
#bipath         =         r'C:\Users\hjara\OneDrive\Maestría\thesis         works\tesis\lsystems-
tests\artichocke6\images_db\train'
#bipath = r'D:\databases\artichoke seedlings\300seedlingv2\train'
bipath = r'D:\databases\artichoke seedlings\artichokev6_ds\3kseedlings\train'
#bipath = r'D:\databases\artichoke seedlings\3kseednling\train'

############## TESTING DATA ##############
#newdatapath       =       r'C:\Users\hjara\OneDrive\Maestría\thesis       works\tesis\lsystems-
tests\artichocke6\images_db\test'
#newdatapath = r'D:\databases\artichoke seedlings\300seedlingv2\test'
#newdatapath = r'D:\databases\artichoke seedlings\artichokev6_ds\3kseedlings\test'
#newdatapath = r'D:\databases\artichoke seedlings\3kseednling\test'

############## VALIDATION DATA ##############
newdatapath = r'D:\databases\artichoke seedlings\plantinescorrelacion\images\plantines_dataset -
copia - fondo verde\malo_regular_bueno'

################# MODEL ####################
pcakmeansmodelpath                          =                          r'C:\Users\hjara\OneDrive\IA
apps\artichoke_seedling_classification\pcakmeansmodel'
modelname = "300sv2_pcakmeans.pkl"


#D:\databases\artichoke seedlings\3kseednling'
```

```
operation = 0 #0:needs trainng/ 1:testing or deploying

def seedling_ql(sn):
    if sn[0] == 'a':
        return 1
    if sn[0] == 'g':
        return 2
    if sn[0] == 'b':
        return 0

def load_training_imgs(path): #left limit and rigth limit: [ll:rl]

    width, height = 224, 224
    dictionary = [] #[index][name][quality]
    tr_ds = []
    label_ds = []
    c = 0
    sc = 0
    for seedling in os.listdir(path):

        ip = os.path.join(path,seedling)
        dim = (width, height)
        image = cv2.resize(imageio.imread(ip),dim,interpolation=cv2.INTER_AREA)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = np.array(image).reshape(3,width,height)/255 #and normalizing
        tr_ds.append(image)
        label_ds.append(seedling_ql(seedling))
        dic = ([c],[seedling],[seedling[0]])
        dictionary.append(dic)
        sc += 1
        c += 1

    #tesst
    #print("testing")
    #timpath                                =                                r'C:\Users\hjara\Google
Drive\Colab_Notebooks\artichoke_seedling\data\testing_images_2\r_2.png'
    #image = cv2.resize(imageio.imread(timpath),(224,224),interpolation=cv2.INTER_AREA)
    #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    #image = np.array(image).reshape(3,224,224)/255 #and normalizing
    #tr_ds.append(image)

    label_ds = np.stack(label_ds)
    tr_ds = np.stack(tr_ds)
    print(tr_ds.shape)
    tr_ds = tr_ds.reshape(tr_ds.shape[0],tr_ds.shape[2],tr_ds.shape[3],tr_ds.shape[1])
    print(tr_ds.shape)
    dictionary = np.stack(dictionary)
    print("Seedling generation counter: " + str(sc) + " units")

    return tr_ds, label_ds

def make_kmean_regions(reduced_data, h, kmeans):
    np.random.seed(42)
    # Plot the decision boundary. For that, we will assign a color to each
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Obtain labels for each point in mesh. Use last trained model.
    Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    return x_min, x_max, y_min, y_max, xx, yy, Z

def plot_kmean_result(Z,xx,yy,npred,reduced_data,kmeans,x_min,x_max,y_min,y_max,samples,spath):
    #plt.figure(1)
    plt.figure(figsize = (17,11))
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=plt.cm.Paired,
               aspect='auto', origin='lower')
```

```
    plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)



    pos = 0
    for seedling in os.listdir(spath):
        #plt.text(reduced_data[pos-samples,0],      reduced_data[pos-samples,1],      seedling[:-
4]+"/"+str(kmeans.labels_[pos-samples]), fontsize=12, color='black')
        plt.text(reduced_data[pos-samples,0],      reduced_data[pos-samples,1],      seedling[:-
4]+"/"+str(npred[pos]), fontsize=12, color='black')
        #plt.text(reduced_data[-1,0],     reduced_data[-1,1],     seedling[:-4],     fontsize=12,
color='black')
        pos += 1
    #plt.text(60, 4.1, 'No continuous trend observed', fontsize=15,  color='red')

    #plt.plot(reduced_data[300,0], reduced_data[300,1], 'ko', markersize=12)

    # Plot the centroids as a white X
    centroids = kmeans.cluster_centers_
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=169, linewidths=3,
                color='w', zorder=10)
    plt.title('K-means clustering on the digits dataset (PCA-reduced data)\n'
              'Centroids are marked with white cross')

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()

def get_group_class(npred,kmeans,c,samples):

    centroids = kmeans.cluster_centers_
    cxmin = centroids[:,0].min()
    temp = []
    co = 0
    for cx,cy in centroids:
        if cxmin == cx:
            min1idx = co
        else:
            temp.append(cx)
        co += 1
    #print(min1idx)
    cxmin = np.array(temp)[:].min()
    co = 0
    for cx,cy in centroids:
        if cxmin == cx:
            min2idx = co
        co += 1
    #print(min2idx)
    """
    if kmeans.labels_[c-samples] == min1idx:
        return 0
    if kmeans.labels_[c-samples] == min2idx:
        return 1
    else:
        return 2
    """
    if npred[c] == min1idx:
        return 0
    if npred[c] == min2idx: #plus 1 cos tem is a -1-element array
        return 1
    else:
        return 2


def get_kmeans_pred(samples,spath,npred,kmeans):
    print("getting predictions")
    #if kmeans.labels_[c] == 0:  #regular 0
    #        shutil.copy(image_path, medium_path)
    c = 0
    pred = []
```

```
    for seedling in os.listdir(spath):
        clss = get_group_class(npred,kmeans,c,samples)
        pred.append(clss)
        c += 1

        """
        if clss == 0:  #regular 0
            pred.append(0)
        if clss == 1:  #regular 0
            pred.append(1)
        if clss == 2:  #regular 0
            pred.append(2)
        """


    """
    for seedling in os.listdir(spath):
        if kmeans.labels_[c-samples] == 4:  #regular 0
            pred.append(0)
        if kmeans.labels_[c-samples] == 1:  #regular 0
            pred.append(1)
        if kmeans.labels_[c-samples] == 0:  #regular 0
            pred.append(2)
        if kmeans.labels_[c-samples] == 2:  #regular 0
            pred.append(2)
        if kmeans.labels_[c-samples] == 3:  #regular 0
            pred.append(2)

        c += 1
    """
    pred = np.array(pred)
    return pred

if operation == 0:
    print("Starting with pca-kmeans creating model process")
    data, _ = load_training_imgs(bipath)
    data = np.array(data)
    print(data.shape)

    elem_num = data.shape[0]
    #taking a sample image to view
    #Remember image is in the form of numpy array.
    data = data.reshape(elem_num,150528) #224,224,3)

    #load new data and concatenate with training data
    newdata, ndlabels = load_training_imgs(newdatapath)
    newdata = np.array(newdata)
    newdata = newdata.reshape(newdata.shape[0],150528)


    tdata = np.vstack([data, newdata])

    now1 = datetime.now()
    tbt = now1.strftime("%d/%m/%Y %H:%M:%S")

    pca = PCA(2) # we need 2 principal components.
    converted_data = pca.fit_transform(tdata)


    #start training kmeans model
    #Obtein k-mean values
    n_digits = 5 #groups quantity
    kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)
    #kmeans.fit(converted_data)
    kmeans.fit(converted_data[:converted_data.shape[0]-newdata.shape[0],:])

    # Step size of the mesh. Decrease to increase the quality of the VQ.
    h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max, y_min, y_max, xx, yy, Z = make_kmean_regions(converted_data, h, kmeans)

    #Get predictions
    samples = newdata.shape[0]
    npred = kmeans.predict(converted_data[-samples:])
```

```
    now2 = datetime.now()
    tet = now2.strftime("%d/%m/%Y %H:%M:%S")

    #Plot results

plot_kmean_result(Z,xx,yy,npred,converted_data,kmeans,x_min,x_max,y_min,y_max,samples,newdatapat
h)
    #plot_kmean_result(Z,xx,yy,converted_data,kmeans,x_min,x_max,y_min,y_max,1,i,newdatapath)

    #Get predictions
    pred = get_kmeans_pred(newdata.shape[0],newdatapath,npred,kmeans)

    confusion = confusion_matrix(ndlabels, pred)
    print('Confusion Matrix\n')
    print(confusion)

    print(classification_report(ndlabels, pred, target_names=['Bad', 'Average', 'Good']))


    #It is time for pearson's correlation
    x = ndlabels
    y = pred
    r, p = scipy.stats.pearsonr(x, y)
    print("R: " + str(r) + " and p = " + str(p))
    plt.style.use('ggplot')
    slope, intercept, r, p, stderr = scipy.stats.linregress(x, y)
    line = f'Regression line: y={intercept:.2f}+{slope:.2f}x, r={r:.2f}'
    fig, ax = plt.subplots()
    ax.plot(x, y, linewidth=0, marker='s', label='Data points')
    ax.plot(x, intercept + slope * x, label=line)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.legend(facecolor='white')
    #plt.show()

    td = (now2 - now1).microseconds
    tp = td/48
    print("((((()Prediction time((()))")
    print("TValidation begining at: " + str(tbt))
    print("Validation ending at: " + str(tet))
    print("Total time: " + str(td))
    print("time/pred: " + str(tp) + " microseconds")
```

Anexos VIII: Código para entrenamiento, testeo y validación del VGG16

Programa "artichoke_seedling_training_models.py" para el entrenamiento, testeo y validación del modelo VGG16. También se puede encontrar en la carpeta PROGRAMAS en el CD adjunto.

```
# -*- coding: utf-8 -*-
"""
Models for seedling classification task
---------------------------------------

This program stands for providing all functions needed for training models
based on the generated artificial seedlings database
6 models were taken into account:
    * VGG16
    * ResNet152V2
    * InceptionV3
    * Xception
    * InceptionResNetV2
    * NASNetLarge

Created on Mon Dec 28 11:34:33 2020
@author: Erick Fiestas
"""

from keras.utils.vis_utils import plot_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.applications import NASNetLarge
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
#from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import model_from_json
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import imageio
#from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
#import argparse
import cv2
import os
from datetime import datetime
import scipy.stats

# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-3
EPOCHS = 3
BS = 50

####################################PATHS#####################################
model_path = "/content/drive/My Drive/Colab_Notebooks/artichoke_seedling/weights/"
model_json                                  =                                  "/content/drive/My
Drive/Colab_Notebooks/artichoke_seedling/weights/artichoke_seedling_model.json"
```

```
#dsfilepath = "/content/drive/My Drive/Colab_Notebooks/artichoke_seedling/data/3kseedlingv2"
dsfilepath   =   "/content/drive/My   Drive/Colab_Notebooks/artichoke_seedling/data/3kseedlingv6"
#2ndwaveseedlings/8kseedlings_wth_mutants" #"/data/reg_tra_images"
tdsfilepath                                =                                "/content/drive/My
Drive/Colab_Notebooks/artichoke_seedling/vgg16_prediction_tests/malo_regular_bueno"
#C:\Users\hjara\Google
Drive\Colab_Notebooks\artichoke_seedling\vgg16_prediction_tests\malo_regular_bueno
################################################################################


def create_model(modelname):
    print("CREATING MODEL: " + modelname)

    if modelname == "VGG16":
        width, height = 224, 224
        print("Assigning image dimensions for " + modelname + " as " + str(width) + ", " +
str(height))
        # load the VGG16 network, ensuring the head FC layer sets are left off
        baseModel = VGG16(weights="imagenet", include_top=False, input_tensor=Input(shape=(width,
height, 3)))

    if modelname == "ResNet152V2":
        width, height = 224, 224
        print("Assigning image dimensions for " + modelname + " as " + str(width) + ", " +
str(height))
        baseModel          =          ResNet152V2(weights="imagenet",          include_top=False,
input_tensor=Input(shape=(width, height, 3)))

    if modelname == "InceptionV3":
        width, height = 229, 229
        print("Assigning image dimensions for " + modelname + " as " + str(width) + ", " +
str(height))
        baseModel          =          InceptionV3(weights="imagenet",          include_top=False,
input_tensor=Input(shape=(width, height, 3)))

    if modelname == "Xception":
        width, height = 229, 229
        print("Assigning image dimensions for " + modelname + " as " + str(width) + ", " +
str(height))
        baseModel           =           Xception(weights="imagenet",           include_top=False,
input_tensor=Input(shape=(width, height, 3)))

    if modelname == "InceptionResNetV2":
        width, height = 229, 229
        print("Assigning image dimensions for " + modelname + " as " + str(width) + ", " +
str(height))
        baseModel        =        InceptionResNetV2(weights="imagenet",        include_top=False,
input_tensor=Input(shape=(width, height, 3)))

    if modelname == "NASNetLarge":
        width, height = 331, 331
        print("Assigning image dimensions for " + modelname + " as " + str(width) + ", " +
str(height))
        baseModel          =          NASNetLarge(weights="imagenet",          include_top=False,
input_tensor=Input(shape=(width, height, 3)))

    # loop over all layers in the base model and freeze them so they will
    # *not* be updated during the first training process
    for layer in baseModel.layers:
        layer.trainable = False

    return baseModel

def model_heading(baseModel):
    # construct the head of the model that will be placed on top of the
    # the base model
    headModel = baseModel.output
    headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
    headModel = Flatten(name="flatten")(headModel)
    headModel = Dense(64, activation="relu")(headModel)
    headModel = Dropout(0.5)(headModel)
    headModel = Dense(3, activation="softmax")(headModel)

    # place the head FC model on top of the base model (this will become
```

```
    # the actual model we will train)
    nmodel = Model(inputs=baseModel.input, outputs=headModel)

    return nmodel

def load_training_imgs(path):
    datafilepath = os.path.join(path,"3kseedlngimages.npy")
    laelfilepath = os.path.join(path,"3kseedlnglabels.npy")
    with open(datafilepath,'rb') as f:
        tr_ds = np.load(f)
    with open(laelfilepath,'rb') as f:
        label_ds = np.load(f)

    tr_ds = tr_ds.reshape(tr_ds.shape[0],tr_ds.shape[2],tr_ds.shape[3],tr_ds.shape[1])

    return [tr_ds, label_ds]

def load_testing_imgs(path,modelname):

    if modelname == "VGG16":
        width, height = 224, 224
    if modelname == "ResNet152V2":
        width, height = 224, 224
    if modelname == "InceptionV3":
        width, height = 229, 229
    if modelname == "Xception":
        width, height = 229, 229
    if modelname == "InceptionResNetV2":
        width, height = 229, 229
    if modelname == "NASNetLarge":
        width, height = 331, 331

    X=[]
    y = []
    category_images=[]
    c = 0

    print("Loading seedling images from testing dataset")
    curr_y = 0
    for tseedling in os.listdir(path):
        image_path = os.path.join(path,tseedling)
        image = imageio.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        dim = (width, height)
        image = cv2.resize(image,dim,interpolation=cv2.INTER_AREA)
        if tseedling[0]=='b': curr_y = 0
        if tseedling[0]=='a': curr_y = 1
        if tseedling[0]=='g': curr_y = 2
        if tseedling[0]=='m': curr_y = 3
        print("tseedling[0]: "+str(tseedling[0]))
        print("curr_y: " + str(curr_y))
        image = np.array(image)
        category_images.append(image.reshape(width, height, 3))
        print(image.shape)
        print("-")
        y.append(curr_y)
        c += 1
    y = np.stack(y)
    X = np.stack(category_images)
    X = X.reshape(c,width,height,3)

    return X,y

def train_seedling_vgg16(model,mn):

    #Necessary for test images, real seedlings
    #te_data, te_labels, sqdict = loadimgs(train_folder)

    #Training images, synthetic seedlings
    npath = os.path.join(dsfilepath,mn)
    tr_data, tr_labels = load_training_imgs(npath)
    ts_data, ts_labels = load_testing_imgs(tdsfilepath,mn)

    print("labels:" + str(tr_labels.shape))
```

```
    print("data:" + str(tr_data.shape))
    print("la0")
    #Normalizing data
    ts_data = np.array(ts_data) / 255.0
    print("la1")
    tr_data = np.array(tr_data) / 255.0

    # perform one-hot encoding on the labels
    tr_lb = LabelBinarizer()
    print("la2")
    tr_labels = tr_lb.fit_transform(tr_labels)
    print("la3")
    ts_lb = LabelBinarizer()
    print("la4")
    ts_labels = ts_lb.fit_transform(ts_labels)

    #labels = to_categorical(labels,3)  ####OJOOOOO!!!
    print("one-hot encoding on labels for training: " + str(tr_labels.shape))
    print("one-hot encoding on labels for testing: " + str(ts_labels.shape))

    # partition the data into training and testing splits using 80% of
    # the data for training and the remaining 20% for testing
    tp = 0.2
    (trainX,  testX,  trainY,  testY)  =  train_test_split(tr_data,  tr_labels,  test_size=tp,
stratify=tr_labels, random_state=42)
    print("trainX shape:" + str(trainX.shape))
    print("trainY shape:" + str(trainY.shape))

    #getting testing data:
    #(trX,  rtestX,  trY,  rtestY)  =  train_test_split(te_data,  te_labels,  test_size=tp,
stratify=te_labels, random_state=42)
    #print("Real test data shape:" + str(rtestX.shape))

    #INITIALIZE THE TRAINING DATA AUGMENTATION OBJECT
    trainAug = ImageDataGenerator(
        featurewise_center=True,
        featurewise_std_normalization=True,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True)

    trainAug.fit(trainX)

    # train the head of the network
    #print("[INFO] training head whit fit...")
    H = model.fit(
        trainAug.flow(trainX, trainY, batch_size=BS),
                        steps_per_epoch=len(trainX) // BS,
                        validation_data=(testX, testY),
                        validation_steps=len(testX) // BS,
                        epochs=EPOCHS)

    #to visualize the accuracy history
    print(H.history.keys())
    # summarize history for accuracy
    plt.plot(H.history['accuracy'])
    plt.plot(H.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('/content/drive/My Drive/Colab_Notebooks/artichoke_seedling/acc.png')
    plt.show()
    # summarize history for loss
    plt.plot(H.history['loss'])
    plt.plot(H.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('/content/drive/My Drive/Colab_Notebooks/artichoke_seedling/loss.png')
    plt.show()
```

```
    ############################## TESTING MODEL ################################
    now1 = datetime.now()
    tbt = now1.strftime("%d/%m/%Y %H:%M:%S")

    y_pred = model.predict(ts_data)

    now2 = datetime.now()
    tet = now2.strftime("%d/%m/%Y %H:%M:%S")

    y_pred = np.argmax(y_pred, axis=1)
    ts_labels = ts_labels.argmax(axis=1)
    #importing confusion matrix
    confusion = confusion_matrix(ts_labels, y_pred)
    print('Confusion Matrix\n')
    print(confusion)

    #importing accuracy_score, precision_score, recall_score, f1_score
    print('\nAccuracy: {:.2f}\n'.format(accuracy_score(ts_labels, y_pred)))

    print('Micro Precision: {:.2f}'.format(precision_score(ts_labels, y_pred, average='micro')))
    print('Micro Recall: {:.2f}'.format(recall_score(ts_labels, y_pred, average='micro')))
    print('Micro F1-score: {:.2f}\n'.format(f1_score(ts_labels, y_pred, average='micro')))

    print('Macro Precision: {:.2f}'.format(precision_score(ts_labels, y_pred, average='macro')))
    print('Macro Recall: {:.2f}'.format(recall_score(ts_labels, y_pred, average='macro')))
    print('Macro F1-score: {:.2f}\n'.format(f1_score(ts_labels, y_pred, average='macro')))

    print('Weighted      Precision:      {:.2f}'.format(precision_score(ts_labels,      y_pred,
average='weighted')))
    print('Weighted Recall: {:.2f}'.format(recall_score(ts_labels, y_pred, average='weighted')))
    print('Weighted F1-score: {:.2f}'.format(f1_score(ts_labels, y_pred, average='weighted')))

    print('\nClassification Report\n')
    print(classification_report(ts_labels, y_pred, target_names=['Bad', 'Average', 'Good']))

    #It is time for pearson's correlation
    x = ts_labels
    y = y_pred
    r, p = scipy.stats.pearsonr(x, y)
    print("R: " + str(r) + " and p = " + str(p))
    plt.style.use('ggplot')
    slope, intercept, r, p, stderr = scipy.stats.linregress(x, y)
    line = f'Regression line: y={intercept:.2f}+{slope:.2f}x, r={r:.2f}'
    fig, ax = plt.subplots()
    ax.plot(x, y, linewidth=0, marker='s', label='Data points')
    ax.plot(x, intercept + slope * x, label=line)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.legend(facecolor='white')
    #plt.show()
    plt.savefig('/content/drive/My
Drive/Colab_Notebooks/artichoke_seedling/regresion_wth_correlation_fig.png')

    td = (now2 - now1).seconds
    print("(((()))Prediction time((()))")
    print("Training begining at: " + str(tbt))
    print("Training ending at: " + str(tet))
    print("Elapsed time: " + str(td))

def save_model(model):
    # serialize the model to disk
    print("[INFO] saving COVID-19 detector model...")
    #m_path = os.path.join(model_path, 'artichoke_seedling_model.h5')
    model.save(model_path)

def load_model():
    # load json and create model
    json_file = open(model_json, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    wn_path = os.path.join(model_path, 'artichoke_seedling_model_w.h5')
    loaded_model.load_weights(wn_path)
```

```
    print("Loaded model from disk and weights updated")

    return loaded_model

def show_feature_maps(model):
    print("Extracting featuring maps from vgg16seedlingmodel")
    # redefine model to output right after the first hidden layer
    ixs = [2, 5, 9, 13, 17]
    outputs = [model.layers[i].output for i in ixs]
    model = Model(inputs=model.inputs, outputs=outputs)

    #get testing image
    img   =   cv2.imread("/content/drive/My   Drive/Colab_Notebooks/artichoke_seedling/data/image
ds/g_7.bmp")
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224, 224), interpolation = cv2.INTER_AREA)
    img = np.array(img)
    img = img.reshape(1,224,224,3)#img.shape[1],img.shape[2],img.shape[0])
    #tr_ds = tr_ds.reshape(tr_ds.shape[0],tr_ds.shape[2],tr_ds.shape[3],tr_ds.shape[1])

    # get feature map for first hidden layer
    feature_maps = model.predict(img)
    # plot the output from each block
    square = 8
    cc = 0
    for fmap in feature_maps:
        # plot all 64 maps in an 8x8 squares
        ix = 1
        for _ in range(square):
            plt.figure(figsize=(64,64))
            for _ in range(square):
                # specify subplot and turn of axis
                ax = plt.subplot(square, square, ix)
                ax.set_xticks([])
                ax.set_yticks([])
                # plot filter channel in grayscale
                plt.imshow(fmap[0, :, :, ix-1], cmap='viridis')
                ix += 1
                        # show the figure
        plt.savefig('/content/drive/My    Drive/Colab_Notebooks/artichoke_seedling/dreamm_'    +
str(cc) + '.png')
        cc += 1

    plt.show()
    #plt.savefig('/content/drive/My Drive/Colab_Notebooks/artichoke_seedling/dream.png')

    return

#Create the model encarnation
modelname = ["VGG16","ResNet152V2","InceptionV3","Xception","InceptionResNetV2","NASNetLarge"]
mn = modelname[0]
print("]=]=]=]=]=]=]=]Creando el modelo[=[=[=[=[=[=[=")
base_model = create_model(mn)
#set head model for transfer learning
model = model_heading(base_model)

#model compilation
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

now1 = datetime.now()
tbt = now1.strftime("%d/%m/%Y %H:%M:%S")

#Modify googlenet in order to retrain on artichoke seedling images
#model training
train_seedling_vgg16(model,mn)


#Save model for future testings
save_model(model)
plot_model(model, to_file="/content/drive/My Drive/Colab_Notebooks/artichoke_seedling/model.png",
show_shapes=True, show_layer_names=True)

#save vgg16 dream as an image
```

```
show_feature_maps(model)

now2 = datetime.now()
tet = now2.strftime("%d/%m/%Y %H:%M:%S")

td = (now2 - now1).microseconds #/ 3600  #from seconds to hours
tp = td/48 #num of seedling images
print("#-#-#-#-#-#-#-#- The process has been finished for " + mn + " model training -#-#-#-#-#-#-
#-#-")
print("(((()))Maquina entrenada((()))")
print("Training begining at: " + str(tbt))
print("Training ending at: " + str(tet))
print("Elapsed time: " + str(td))
print("time/pred: " + str(tp) + " microseconds")
```

Anexos IX: Código para entrenamiento, testeo y validación del Yolo y umbrales

Programa "test_yolov3_v1.py" para el entrenamiento, testeo y validación del modelo Yolov3. También se puede encontrar en la carpeta PROGRAMAS en el CD adjunto.

```
# -*- coding: utf-8 -*-
"""
Test Yolov3 based on seedling images
-----------------


Created on Tue Jun 22 21:31:21 2021

@author: Erick Fiestas
"""

###########LIBRERIAS/APIs/FRAMEWORKS##############
from time import time,sleep
from datetime import datetime
from PIL import Image
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import numpy as np
import cv2
import csv
import os
import imageio
import scipy.stats
import matplotlib.pyplot as plt



##########VARIABLES#############
data = []
labels = []

plantpath  =  r'D:\databases\artichoke  seedlings\plantinescorrelacion\images\images_18_06_2021'
#'/home/plantinator/plantinator/images/test_images'
seedldestpath                              =                              r'D:\databases\artichoke
seedlings\plantinescorrelacion\images\plantines_segmentados_18_06_21' #'/home/root/vision'
modelpath  =  r'C:\Users\hjara\OneDrive\IA  apps\artichoke_seedling_classification\yolov3_testes'
#"/home/erickmfs/ai_apps/seedling_groups/darknet-yolov3/" #'/home/plantinator/plantinator'


#simg_path                      =                      r'C:\Users\hjara\OneDrive\Maestría\thesis
works\tesis\imagenes\plantinescorrelacion\images\images_18_06_2021'
#mask_path       =       r'C:\Users\hjara\OneDrive\Maestría\thesis       works\tesis\imagenes\4ta
ola\paulos\images_06_03_2021\seedling_0_mask.jpg'
base_img_path                              =                              r'D:\databases\artichoke
seedlings\artichokev6_ds\seedling_groups_datasets\size_ref.jpg'
greenbkgd_path                             =                              r'D:\databases\artichoke
seedlings\artichokev6_ds\seedling_groups_datasets\green_size_ref.png'
destination_path                           =                              r'D:\databases\artichoke
seedlings\artichokev6_ds\seedling_groups_datasets\real_seedling_images\correlation_images'
seedling_dic_path = r'D:\databases\artichoke seedlings\plantinescorrelacion\dic.csv'
xyareadatainfofilepath                         =                          r'D:\databases\artichoke
seedlings\plantinescorrelacion\xyareadatainfo.csv'
w_h_centroids_filepath                 =                 r'C:\Users\hjara\OneDrive\Maestría\thesis
works\tesis\imagenes\plantinescorrelacion\ancho-alto-centroides-plantines-reales.csv'
IoUfilepath                        =                       r'C:\Users\hjara\OneDrive\Maestría\thesis
works\tesis\imagenes\plantinescorrelacion\Yolov3_IoU.csv'
seedling_quality_dataset                       =                          r'D:\databases\artichoke
seedlings\plantinescorrelacion\images\plantines_dataset'

mixdataset_path                            =                              r'D:\databases\artichoke
seedlings\plantinescorrelacion\images\plantines_dataset - copia - fondo verde\malo_regular_bueno'

#limits for every group of seedling predictions
g3min = 60
g3max = 175
```

```
g2min = 175
g2max = 275
g1min = 275
g1max = 400
hmin = 175 #194
hmax = 232
score_threshold = 1.0
left_threshold = 2780 #2614
right_threshold = 4600 #4676

DIAG_CMD = "DIAG"
user = "cts"
password = "123456789cts"

modServerIp, modServerPort = "192.168.1.103", 502

#to save or not images from yolo analysis
saveimages = False
#to show or not images according processing process is running
showimages = False

###########FUNCTIONS###########
def class_pred_in_letters(pred): #from number-based seedling classification to letters
    if pred == 3:
        return "A"
    if pred == 2:
        return "B"
    if pred == 1:
        return "C"

def classify_pred(box, score): #classifications images according to calibrated thresholds
    print("Classifying predictions done by yolov3")
    x, y, w, h = box
    area = w * h

    if score < score_threshold: #It is suspected that it is not average seedling
        #then it is necessary to know if it is bad or good seedling
        if area < left_threshold:
            #bad seedling
            return 1#0
        if area > right_threshold:
            #good seeling
            return 3#2
        else:
            return 2#1  #TO BE CONSIDERED!
    else:
        #at least that it has a big area
        if area <= right_threshold: #area > left_threshold and area < right_threshold: #OJOOOO!!
take it into account
            #average seedling
            return 2#1
        if area > right_threshold:
            #good seedling
            return 3#2
        else:
            return 1#0
    print("no pasa")

def cropimage(image,x_min,x_max,y_min,y_max,res_per):  #resizing images
    #per_res: resizing percentage
    #image = imageio.imread(path)
    #image = np.array(image)

    # crop the image using array slices -- it's a NumPy array
    # after all!
    cropped = image[y_min:y_max,x_min:x_max] #443:720, 403:1082]
    size=(int(cropped.shape[1]*res_per),int(cropped.shape[0]*res_per))  #image  size  proportion
720/1424 = 0.51
    #resize image
    #rsimg = cv2.resize(image,size) #this is really important to facilitate seedling manipulation
    cropped = cv2.resize(cropped,size)

    return np.array(cropped)
```

```
def applymask(spath,mpath):

    simg = imageio.imread(spath)
    mask = imageio.imread(mpath)
    #simg = cv2.imread(spath)
    #print(spath)
    #print(simg)
    #simg = cv2.cvtColor(simg, cv2.COLOR_BGR2RGB)

    #cv2.imshow("simg", simg)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    #mask = cv2.imread(mpath)
    #mask = cv2.cvtColor(mask, cv2.COLOR_RGB2GRAY)
    #print(" lalalalalal")
    #print(mask)
    #mimg = imageio.imread(mpath)
    #mimg = cv2.cvtColor(mimg, cv2.COLOR_GRAY2RGB)

    simg, mask = np.array(simg), np.array(mask)
    return cv2.bitwise_and(simg,simg,mask=mask)

def imageremix(base_img,mask_img,rv,cv): #Put two images together
    print("Mixing 2 images")
    brv = rv #base image row value
    bcv = cv
    bimg = np.array(base_img)
    mimg = np.array(mask_img)
    mimg_r, mimg_c, _ = mimg.shape
    bimg_r, bimg_c, _ = bimg.shape
    limitflag_c = True
    #limitflag_r = True

    #if debug_mode:
    #    print("mask image row and column: " + str(mimg_r) + ", " + str(mimg_c))
    #    print("base image row and column: " + str(bimg_r) + ", " + str(bimg_c))

    for r in range(mimg_r):
        for c in range(mimg_c):
            if mimg[r,c,0] > 0: #different from zero
                if limitflag_c:
                    bimg[brv,bcv,:] = mimg[r,c,:]
            bcv += 1
            if bcv >= bimg_c: limitflag_c = False
            else: limitflag_c = True
        brv += 1
        bcv = cv
        if brv >= bimg_r: break

    return bimg

def get_prediction(modelpath,image_input,seedling_counter,gcou,acou,bcou):

    print("begin PREDICTION PROCESS")
    weight_path = os.path.join(modelpath, 'yolov3_training_last.weights')
    cfg_path = os.path.join(modelpath, 'yolov3_testing.cfg')
    net = cv2.dnn.readNet(weight_path, cfg_path)
    print("yolov3 loaded")
    classes = []
    classpath = os.path.join(modelpath, 'classes.txt')
    with open(classpath, "r") as f:
        classes = f.read().splitlines()

    #cap = cv2.VideoCapture('video4.mp4')
    #cap = 'test_images/<your_test_image>.jpg'
    font = cv2.FONT_HERSHEY_PLAIN
    colors = np.random.uniform(0, 255, size=(100, 3))

    ##########################################################
    img = cv2.imread(image_input) #"test_images/img19.png")
    ##########################################################

    height, width, _ = img.shape
```

```
    if width > 416:
        img = cropimage(img,403,1082,443,720,0.5625) #according to 416x146pixel image size input
for yolov3 model
        img   =   imageremix(imageio.imread(os.path.join(modelpath,"size_ref.jpg")),img,int(679/2-
416/2),int(416/2-277/2))
        height, width, _ = img.shape

    print("height: " + str(height) + ", width: " + str(width))

    blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), (0,0,0), swapRB=True, crop=False)
    net.setInput(blob)
    output_layers_names = net.getUnconnectedOutLayersNames()
    layerOutputs = net.forward(output_layers_names)
    print("aquiiiiii")

    boxes = [(0,0,0,0),(0,0,0,0),(0,0,0,0)]
    confidences = [0,0,0]
    class_ids = [0,0,0]

    PM = np.ones([10,3,6]) #*-1 #Position matrix
    idx = [0,0,0] #np.zeros([1,3]) #with zeros considering 0 position at the beginning

    for output in layerOutputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.2:
                center_x = int(detection[0]*width)
                center_y = int(detection[1]*height)
                w = int(detection[2]*width)
                h = int(detection[3]*height)
                x = int(center_x - w/2)
                y = int(center_y - h/2)

                print("Seedling detected: " + str(center_x) + ", " + str(center_y))

                if center_x > g1min and center_x <= g1max and center_y >= hmin and center_y <=
hmax:
                    #1st group
                    PM[idx[0],0,:] = [classify_pred([x, y, w, h],confidence),x,y,w,h,confidence]
                    idx[0] += 1
                if center_x > g2min and center_x <= g2max and center_y >= hmin and center_y <=
hmax:
                    #2nd group
                    PM[idx[1],1,:] = [classify_pred([x, y, w, h],confidence),x,y,w,h,confidence]
                    idx[1] += 1
                if center_x > g3min and center_x <= g3max and center_y >= hmin and center_y <=
hmax:
                    #3th group
                    PM[idx[2],2,:] = [classify_pred([x, y, w, h],confidence),x,y,w,h,confidence]
                    idx[2] += 1
        print("por aquiiiii")

        #here it is necessary to treat boxes, confidences and class_ids as stablished array size
variables
        #in order to know if there is a cell with no seedling detection

        for pidx in range(3):
            val = -1
            print(idx[pidx])
            if idx[pidx] > 0: #if there is 0 value at position it must be recognized as no seedling
detection situation
                for i in range(idx[pidx]):
                    print("i: ", i)
                    c = PM[i,pidx,0] #this stage goes for discrimination in case more than one
seedling was detected and classified at the same cell
                    if c > val: #the greatest class is our guy!
                        indx = i
                        val = c

                ##################those     variables     could     be     initialized     as     -1
arrays#################3
                print("PM: ",PM[indx,pidx,1:5])
```

```
                boxes[pidx] = PM[indx,pidx,1:5]
                confidences[pidx] = float(PM[indx,pidx,5])
                class_ids[pidx] = int(PM[indx,pidx,0])
                #boxes.append(PM[indx,pidx,1:5])
                #confidences.append(float(PM[indx,pidx,5]))
                #class_ids.append(PM[indx,pidx,0])

            #else:
            #    boxes.append((0,0,0,0))
            #    confidences.append(0)
            #    class_ids.append(0)

            print("por aquiiiii 2")

        #boxes.append([x, y, w, h])
        #confidences.append((float(confidence)))
        #class_ids.append(class_id)

    print("saliooo")
    print("shape boxes: ", len(boxes))
    print(boxes[0])
    print(boxes[1])
    print(boxes[2])

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.2, 0.4)
    print("indexes: ", indexes)
    seedlfnames = [" "," "," "]

    #This is just for saving seedling images each one in different file
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            print("paso aquiiii")

            #label = str(classes[class_ids[i]])

            #confidence = str(round(confidences[i],2))
            #color = colors[i]
            #cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)
            #cv2.putText(img, label + " " + confidence, (x, y+20), font, 1, (255,255,255), 2)

            crop_img = img[int(y):int(y+h+40), int(x):int(x+w+40)]
            now = datetime.now()
            dt_str = now.strftime("%d-%m-%Y_%H-%M-%S.%f")
            sfn = 'seedling_' + "Q" + str(class_ids[i]) + "_c-" + str(x) + "-" + str(y) + "-" +
str(w) + "-" + str(h) + "_" + dt_str + '.png'

            dpath = os.path.join(seedldestpath,sfn)
            if saveimages: cv2.imwrite(dpath,crop_img)

            seedlfnames[i] = sfn
            print("seedling: " + sfn + " wth quality: " + str(class_ids[i]))
            #seedlfnames.append(sfn)

            gcou,                       acou,                       bcou                       =
tag_seedlings_detected(x,y,w,h,crop_img,class_ids[i],i,seedling_counter,gcou,acou,bcou)

            #area and quality
            #s_cen_qua_array.append([class_ids[i],x,y])
            #s_area_array.append([class_ids[i],w*h])

    #seedlfnames = np.array(seedlfnames)
    return seedlfnames, class_ids, gcou, acou, bcou

def generating_csv_file_centroides_area(quality,pred_quality,x,y,area):
    #generating csv file
    with open(xyareadatainfofilepath,mode='a',newline='') as file:
        writer = csv.writer(file,delimiter=';')
        #writer.writerow(["ID","X","Y","Area"])
        #for i in range(s_cen_qua_array.shape[0]):
        writer.writerow([quality,pred_quality,x,y,area])

def bb_intersection_over_union(boxA, boxB):
                        # determine the (x, y)-coordinates of the intersection rectangle
```

```
                            xA = max(boxA[0], boxB[0])
                            yA = max(boxA[1], boxB[1])
                            xB = min(boxA[2], boxB[2])
                            yB = min(boxA[3], boxB[3])
                            # compute the area of intersection rectangle
                            interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
                            # compute the area of both the prediction and ground-truth
                            # rectangles
                            boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
                            boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
                            # compute the intersection over union by taking the intersection
                            # area and dividing it by the sum of prediction + ground-truth
                            # areas - the interesection area
                            iou = interArea / float(boxAArea + boxBArea - interArea)
                            # return the intersection over union value
                            return iou

    def generating_csv_file_IoU(sp,x,y,w,h):
        boxA = np.array([0,0,0,0])
        boxB = np.array([0,0,0,0])
        with open(w_h_centroids_filepath,mode='r') as rfile:
            csv_reader = csv.reader(rfile, delimiter=';')
            with open(IoUfilepath, mode='a', newline='') as wfile:
                csv_writer = csv.writer(wfile, delimiter=';')
                for row in csv_reader:
                    if row[0].isdigit():
                        if sp == int(row[0]):
                            boxA[0] = int(row[1]) - int(row[3])/2 #real coordinates
                            boxA[1] = int(row[2]) - int(row[4])/2
                            boxA[2] = int(row[1]) + int(row[3])/2
                            boxA[3] = int(row[2]) + int(row[4])/2
                            boxB[0] = x #predicted coordinates
                            boxB[1] = y
                            boxB[2] = x + w
                            boxB[3] = y + h

    csv_writer.writerow([sp,row[1],row[2],row[3],row[4],x,y,w,h,bb_intersection_over_union(boxA,
    boxB)])

    def tag_seedlings_detected(x,y,w,h,img,seedling_class,pos,seedling_counter,gcou,acou,bcou):

        sp = seedling_counter + pos  #seedling position for retrieving seedling quality in dic.csv
        #then it must ask for seedling validity

        #quality due to dic.csv file
        #seedling_counter allows me to follow up seedling processing
        with open(seedling_dic_path) as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=';')
            for row in csv_reader:
                if row[0].isdigit():
                    if sp == int(row[0]):
                        if row[2] == 's':
                            print("Seedling is valid")
                            #seedling_dic_path
                            qa = int(row[1]) + 1
                            if qa == 1:
                                name = "b_"+str(sp)+"_"+str(bcou)+".png"
                                #seedln                                                                    =
    os.path.join(os.path.join(seedling_quality_dataset,'malo'),name)
                                seedln = os.path.join(mixdataset_path,name)
                                if saveimages: cv2.imwrite(seedln,greebkgd(img))
                                bcou += 1
                            if qa == 2:
                                name = "a_"+str(sp)+"_"+str(acou)+".png"
                                #seedln                                                                    =
    os.path.join(os.path.join(seedling_quality_dataset,'regular'),name)
                                seedln = os.path.join(mixdataset_path,name)
                                if saveimages: cv2.imwrite(seedln,greebkgd(img))
                                acou += 1
                            if qa == 3:
                                name = "g_"+str(sp)+"_"+str(gcou)+".png"
                                #seedln                                                                    =
    os.path.join(os.path.join(seedling_quality_dataset,'bueno'),name)
                                seedln = os.path.join(mixdataset_path,name)
```

```
                            if saveimages: cv2.imwrite(seedln,greebkgd(img))
                            gcou += 1
                        if showimages:
                            cv2.imshow("New seedling detected: " + name, img)
                            cv2.waitKey(0)
                            cv2.destroyAllWindows()
                        #saving centroids and area in a csv file
                        generating_csv_file_centroides_area(qa,seedling_class,x,y,w*h)
                        generating_csv_file_IoU(sp,x,y,w,h)
                    if row[2] == 'n':
                        if showimages:
                            cv2.imshow("Bad seedling detected", img)
                            cv2.waitKey(0)
                            cv2.destroyAllWindows()


    return gcou, acou, bcou

def greebkgd(img):
    grbkgd = cv2.imread(greenbkgd_path)
    img    =    imageremix(grbkgd,img,int(grbkgd.shape[1]/2-img.shape[1]/2),int(grbkgd.shape[0]/2-
img.shape[0]/2))

    return img

def start_processing_image(img,mpath,seedlings_counter,gcou,acou,bcou):
    print("Processing image")
    sn,sc,gcou,acou,bcou = get_prediction(mpath,img,seedlings_counter,gcou,acou,bcou)
    return sn, sc, gcou, acou, bcou

def get_seedling_img_num(seedlingname):
    if seedlingname[-3:] == "jpg":
        num = seedlingname.split("QaPeven.jpg")[0].split("DepthrgbSample_num")[1]
    if seedlingname[-3:] == "png":
        num = seedlingname.split("QaPeven_mask.png")[0].split("DepthrgbSample_num")[1]
    return int(num)

def search_img_and_mask(num,path):
    nimgd = 0 #number of images detected
    print("num:"+str(num))
    for sn in os.listdir(path):
        #print("get_seedling_img_num(sn):" + str(get_seedling_img_num(sn)))
        if get_seedling_img_num(sn) == num and sn[-3:]=='jpg':
            imgn = sn
            nimgd += 1
        if get_seedling_img_num(sn) == num and sn[-3:]=='png':
            maskn = sn
            nimgd += 1
    if nimgd >= 2:
        return imgn, maskn

#######INICIO DEL PROGRAMA#########
numimages = 24
#img_names = [] #image names considering mask as well

#how many goods are
gcou = 0
#how many averages are
acou = 0
#how many bads are
bcou = 0

seedlings_counter = 1

#time
totaltime = 0

for i in range(numimages):
    imgn,maskn = search_img_and_mask(i+1,plantpath)

    #Apply mask
    print("imgn: " + os.path.join(plantpath, imgn))
    print("maskn: " + os.path.join(plantpath, maskn))
    img = applymask(os.path.join(plantpath, imgn),os.path.join(plantpath, maskn))
```

```
    #resize, crop and show resulting image
    cimg = cropimage(img,403,1082,443,720,0.55)
    #remix images
    img1 = imageremix(imageio.imread(base_img_path),cimg,int(679/2-416/2),int(416/2-277/2))
    #saving image
    imgf1 = Image.fromarray(img1)
    imgf1p = os.path.join(destination_path,"img" + str(i) + ".png")
    imgf1.save(imgf1p)

    if showimages:
        cv2.imshow("seedlings", img1)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    #read a folder with images and apply masking, process image later

    now1 = datetime.now()
    tbt = now1.strftime("%d/%m/%Y %H:%M:%S")

    [seedlpath,          seedlquality,          gcou,          acou,          bcou]          =
start_processing_image(imgf1p,modelpath,seedlings_counter,gcou,acou,bcou)

    now2 = datetime.now()
    totaltime += (now2-now1).microseconds/3
    tet = now2.strftime("%d/%m/%Y %H:%M:%S")


    seedlings_counter += 3 #seedlings at least cos of robot seedling disposition


"""
#when finishing turn lists into arrays with numpy
s_cen_qua_array = np.array(s_cen_qua_array)
s_area_array = np.array(s_area_array)
#And show them in charts
plt.figure(figsize=(17,11))
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Seedling centroids")
plt.plot(s_cen_qua_array[:,1],s_cen_qua_array[:,2])

plt.figure(figsize = (17,11))
plt.xlabel("seedling quality")
plt.ylabel("Area")
plt.title("Grouping seeedling areas")
plt.plot(s_area_array[:,0],s_area_array[:,1])
"""

realq = []
predq = []
with open(xyareadatainfofilepath, mode='r') as file:
    csv_reader = csv.reader(file, delimiter=';')
    for row in csv_reader:
        realq.append(int(row[0]))
        predq.append(int(row[1]))

realq = np.array(realq)
predq = np.array(predq)
confusion = confusion_matrix(realq, predq)
print('Confusion Matrix\n')
print(confusion)

print(classification_report(realq, predq, target_names=['Bad', 'Average', 'Good']))

#It is time for pearson's correlation
r, p = scipy.stats.pearsonr(realq, predq)
print("R: " + str(r) + " and p = " + str(p))
plt.style.use('ggplot')
slope, intercept, r, p, stderr = scipy.stats.linregress(realq, predq)
line = f'Regression line: y={intercept:.2f}+{slope:.2f}x, r={r:.2f}'
fig, ax = plt.subplots()
x = realq
y = predq
```

```
ax.plot(x, y, linewidth=0, marker='s', label='Data points')
ax.plot(x, intercept + slope * x, label=line)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend(facecolor='white')
plt.show()

td = totaltime
tp  = td/numimages
print("(((()))Prediction time((()))")
print("Elapsed time: " + str(td) + " microseconds")
print("time/pred: " + str(tp) + " microseconds")
```

Anexos X: Código para el tratamiento de centroides dadas por el Yolov3

Con el siguiente programa llamado “Object_Detection.py” se puede extraer las centroides de los plantines detectados por el Yolov3 entrenado en el presente trabajo.

```
import cv2
import numpy as np

print("begin")
net = cv2.dnn.readNet('yolov3_training_last.weights', 'yolov3_testing.cfg')
print("yolov3 loaded")
classes = []
with open("classes.txt", "r") as f:
    classes = f.read().splitlines()

#cap = cv2.VideoCapture('video4.mp4')
#cap = 'test_images/<your_test_image>.jpg'
font = cv2.FONT_HERSHEY_PLAIN
colors = np.random.uniform(0, 255, size=(100, 3))

while True:
    #_, img = cap.read()
    img = cv2.imread("test_images/img1.png")
    height, width, _ = img.shape

    blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), (0,0,0), swapRB=True, crop=False)
    net.setInput(blob)
    output_layers_names = net.getUnconnectedOutLayersNames()
    layerOutputs = net.forward(output_layers_names)

    boxes = []
    confidences = []
    class_ids = []

    for output in layerOutputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.2:
                center_x = int(detection[0]*width)
                center_y = int(detection[1]*height)
                w = int(detection[2]*width)
                h = int(detection[3]*height)

                x = int(center_x - w/2)
                y = int(center_y - h/2)

                boxes.append([x, y, w, h])
                confidences.append((float(confidence)))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.2, 0.4)

    if len(indexes)>0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = str(round(confidences[i],2))
            color = colors[i]
            cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)
            cv2.putText(img, label + " " + confidence, (x, y+20), font, 1, (255,255,255), 2)

    cv2.imshow('Image', img)
    #key = cv2.waitKey(1)
    #if key==27:
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    break
```

Anexos XI: Registro de 30 plantines reales

A contiuación se muestra la tabla de 30 plantines muestreados en el laboratorio a fin de poder usarlos como insumo para la visualización de las centroides de los mismos en la Fig. 38, que permiten establecer las regiones de interés para la identifiación de la posición de los plantines según las agujas del robot selector de plantines. Las imágnes se encuentran en la carpeta PLANTINET en el CD adjunto.

ID	Numfoto	Altura	Largo hoja	Ancho hoja	Largo cotiledona	Ancho cotiledona	Calidad	PALA
1	1	3	4	2	7	3	1	30
2	1	5	5	2	7	3	2	40
3	2	3	3	2	7	2	**1**	26.6666667
4	2	3	1	0.5	3	2	**0**	15
5	2	3	1	0.5	7	2.5	**0**	15
6	3	4	4	2	6	2	**2**	33.3333333
7	3	4	5	0.7	8	2.3	**2**	32.3333333
8	3	4	3.5	2	7	2	**1**	31.6666667
9	4	6.5	5.5	2	8	1.7	**2**	46.6666667
10	4	4.5	4.5	1.7	6.5	2	**2**	35.6666667
11	4	5	5.5	2	7	2.5	**2**	41.6666667
12	5	5	5.2	2.5	8	2	**2**	42.3333333
13	5	2	2	1.7	6	2	**0**	19
14	6	5	4.3	2	7	2	**2**	37.6666667
15	6	4	5	1.7	7	2	**2**	35.6666667
16	6	3.5	4	1.7	7	1.8	**1**	30.6666667
17	7	4	4	1.8	7	2	**2**	32.6666667
18	7	2	3	1.7	7	2	**1**	22.3333333
19	7	3.2	3.5	2	7	2	**1**	29
20	8	2	1.7	0.7	5.5	1.8	**0**	14.6666667
21	8	4	3.5	2.8	6.5	1.8	**2**	34.3333333
22	8	6	6	2	7	2	**2**	46.6666667
23	9	3.5	4	1.2	6	2	**1**	29
24	9	4	4	2	7	2	**2**	33.3333333
25	10	3.5	4.5	2.8	6.5	2	**2**	36
26	10	3	2.4	1	6.2	2	0	21.3333333
27	10	3	5	2	7	2	2	33.3333333

Anexos XII: Tabla para el área de 72 plantines reales

Tabla de 72 áreas de cajas de predicción de la detección de objetos por el Yolov3 que se ajusta a los plantines reales medidos en laboratorio.

TABLA CON MUESTRAS COMPLETAS						
	Malo	**Area**	**Regular**	**Area**	**Bueno**	**Area**
	1855	47389.9408	4230	384885.371	4060	6353760.44
	2622	301738.941	2720	791403.631	6642	3761.77778
	1890	33376.4793	3300	95857.5444	4970	2594247.11
	2652	335597.402	3000	371622.762	3036	12564661.8
	2537	215581.633	4042	186962.24	2940	13254453.8
	1856	46955.5562	2656	909369.544	6776	38155.1111
	2236	26669.4024	2585	1049822.98	7614	1067777.78
	672	1961938.94	1904	2909101.02	8330	3060167.11
	2320	61161.0947	7192	12833527.5	6916	112448.444
	936	1292069.4	2622	975370.936	9130	6499100.44
	1440	400299.556	3465	20911.6749	9632	9310635.11
	2775	493236.095	4248	407543.457	4071	6298426.78
	3154	1169226.33	3311	89167.1531	10452	14987221.8
			4807	1433745.94	5742	703361.778
			2417	1422315.5	9384	7858677.78
			2364	1551541.02	8526	3784321.78
			4060	202852.327	9506	8557575.11
			3759	22317.7618	14544	63414677.8
			4187	333380.718	6840	67253.7778
			4265	429537.762	3819	7626802.78
			2917	479706.805	8250	2786673.78
			3673	4018.45747	5767	662053.444
			5297	2847289.41	6439	20069.4444
					6026	307655.111
					5024	2423211.11
					4813	3124645.44
					3762	7944881.78
					7975	1944165.44
					6252	108021.778
					8220	2687413.78
					4044	6434677.78
					5066	2294215.11
					4676	3627755.11
					6292	83328.4444
					4483	4400205.44
					6885	92618.7778
	18%		32%		50%	
TOTAL	13		23		36	72
PROMEDIO	2072.69231		3609.6087		6580.66667	
MEDIANA	2236		3465		6365.5	
sd	700.836191		1137.35445		1658.20896	
	MIN	MAX				
MALO	1371.85612	2773.5285		<2780		
REGULAR	2472.25424	4746.96315		>2780<4922		
BUENO	4922.4577	8238.87563		>4922		

Anexos XIII: Programa gestor de visión por computadora del SVC

Programa “plantinator_ai_v2.1.py” el cual en la Fig. 43 funge de programa gestor de visión por computadora. También se puede encontrar el programa en la carpeta PROGRAMAS del CD adjunto.

```
# -*- coding utf-8 -*-
"""
Plantinator_AI_v2
-----------------

IA seedling detection and classification service
with modbus plc communication

plantinator_ai version 2 works verywell without nodered image performance
but this version is a depurated one of v2


Created on Mon Apr 12 18:01:17 2021

@author: Erick Fiestas / Paulo Linarez
"""

# -*- coding: utf-8 -*-


###########LIBRERIAS/APIs/FRAMEWORKS##############
#from tensorflow.keras.models import load_model
import numpy as np
import cv2
#import time
from time import time,sleep

from datetime import datetime
#import shutil
import json
import paho.mqtt.client as mqtt
import os
import imageio

import pyrealsense2 as rs
import pickle
from sklearn.cluster import KMeans

from libseedlingmodbus import SeedlingModbusClient
import libseedlingmodbus as lsmodb
import sys
from threading import Timer,Thread




##########VARIABLES#############
data = []
labels = []

plantpath      =      '/home/erickmfs/ai_apps/seedling_groups/darknet-yolov3/images/test_images/'
#'/home/plantinator/plantinator/images/test_images'
seedldestpath     =     '/home/erickmfs/ai_apps/seedling_groups/darknet-yolov3/images/seedlings/'
#'/home/root/vision'
modelpath                =               "/home/erickmfs/ai_apps/seedling_groups/darknet-yolov3/"
#'/home/plantinator/plantinator'


MQTT_SERVER = "localhost" #"192.168.0.8" #"localhost"

MQTT_PATH = "cts/#"
NOMBRE_ESCLAVO = "cts"
```

```
COSTO_KWH = 0.6
working_mode = 0 #if 0: control mode, if 1: manual mode

topic4listenig = "robot/vision/ia/#"
topic_prueba = "robot/vision/ia/prueba"
topic_parametros = "robot/vision/ia/parametros"
topic_ima1_ruta = "robot/vision/imagen1/ruta"
topic_ima1_cali = "robot/vision/imagen1/calidad"
topic_ima2_ruta = "robot/vision/imagen2/ruta"
topic_ima2_cali = "robot/vision/imagen2/calidad"
topic_ima3_ruta = "robot/vision/imagen3/ruta"
topic_ima3_cali = "robot/vision/imagen3/calidad"

#limits for every group of seedling predictions
g3min = 60
g3max = 175
g2min = 175
g2max = 275
g1min = 275
g1max = 400
hmin = 194
hmax = 232
score_threshold = 0.98
left_threshold = 2614
right_threshold = 4676

DIAG_CMD = "DIAG"
user = "cts"
password = "123456789cts"

modServerIp, modServerPort = "192.168.1.103", 502


###########FUNCTIONS###########
def colorize(depth_image):  #Paulos' function 1
    _min=0.35
    _max=0.5
    normalized=255.0*(depth_image-_min)/(_max-_min)
    normalized=np.clip(normalized,0,255).astype(np.uint8)
    colorized=cv2.applyColorMap(normalized,cv2.COLORMAP_JET)
    return colorized

def get_images():    #Paulos' function 2
    align=rs.align(rs.stream.color)
    pipeline = rs.pipeline()
    config = rs.config()
    config.enable_stream(rs.stream.depth, 1280, 720, rs.format.z16, 30)
    config.enable_stream(rs.stream.color, 1280, 720, rs.format.bgr8, 30)

    pipeline_profile = pipeline.start(config)
    depth_sensor = pipeline_profile.get_device().first_depth_sensor()
    depth_sensor.set_option(rs.option.emitter_enabled, 1)
    depth_sensor.set_option(rs.option.laser_power, 250)
    depth_sensor.set_option(rs.option.depth_units, 0.0001)  # changed 0.0001
    temp_filter = rs.temporal_filter()
    temp_filter.set_option(rs.option.filter_smooth_alpha, 0.5)
    temp_filter.set_option(rs.option.filter_smooth_delta, 10)
    temp_filter.set_option(rs.option.holes_fill, 1.0)
    spatial_filter = rs.spatial_filter()
    spatial_filter.set_option(rs.option.holes_fill, 3)
    sleep(0.05)
    pipeline.stop()
    sleep(0.05)

    pipeline_profile = pipeline.start(config)
    depth_sensor = pipeline_profile.get_device().first_depth_sensor()
    depth_sensor.set_option(rs.option.emitter_enabled, 1)
    depth_sensor.set_option(rs.option.laser_power, 250)
    depth_sensor.set_option(rs.option.depth_units, 0.0001)  # changed 0.0001
    temp_filter = rs.temporal_filter()
    temp_filter.set_option(rs.option.filter_smooth_alpha, 0.5)
    temp_filter.set_option(rs.option.filter_smooth_delta, 10)
    temp_filter.set_option(rs.option.holes_fill, 1.0)
    spatial_filter = rs.spatial_filter()
```

```
    spatial_filter.set_option(rs.option.holes_fill, 3)

    sleep(1)

    frames = pipeline.wait_for_frames(timeout_ms=2000)
    aligned_frames = align.process(frames) #NEW
    for i in range(30):
        depth_frame = aligned_frames.get_depth_frame() #From frames.get_depth_frame()
        depth_frame = spatial_filter.process(depth_frame)
        depth_frame = temp_filter.process(depth_frame)

    depth_frame = temp_filter.process(depth_frame)
    color_frame = aligned_frames.get_color_frame() #From frames.get_color_frame()
    depth_image = np.asanyarray(depth_frame.get_data())
    depth_image = depth_image*0.0001
    color_image= np.asanyarray(color_frame.get_data())
    return depth_image,color_image

def capture_and_segment_im():  #Paulos' function 3
    file = open("kmeans_model.pkl","rb")
    print(file)
    kmeans_model = pickle.load(file)
    print("cargooou")
    depthimage,colorimage = get_images()
    mask = np.zeros(depthimage.shape[0:2],dtype=np.uint8)

    #SEGMENTATION
    ###Use the ROI
    # Define the ROI
    row_roi=450
    col_roi=[360,1200]

    #depth= np.ones(depth_orig.shape)

    depth_roi = depthimage[row_roi:,col_roi[0]:col_roi[1]]
    rgb_roi = colorimage[row_roi:,col_roi[0]:col_roi[1]]

    mask_depth_roi = np.where((depth_roi<0.465)&(depth_roi>0.28),255,0).astype(np.uint8)# pixels
between 3cm and 33 cm
    preseg_rgb_roi = cv2.bitwise_and(rgb_roi,rgb_roi,mask=mask_depth_roi)

    preseg_hsv_roi = cv2.cvtColor(preseg_rgb_roi,cv2.COLOR_BGR2HSV) #Convert image to HSV
    reshaped_hsv_roi                                                                    =
np.reshape(preseg_hsv_roi,(preseg_hsv_roi.shape[0]*preseg_hsv_roi.shape[1],3))# Reshape image to
be used by Kmeans
    labeled = kmeans_model.predict(reshaped_hsv_roi)

    mask_roi                                                                            =
np.where((labeled==1)|(labeled==3)|(labeled==5)|(labeled==6),255,0).astype(np.uint8)
    mask_roi = np.reshape(mask_roi,(preseg_hsv_roi.shape[0],preseg_hsv_roi.shape[1]))
    mask[row_roi:,col_roi[0]:col_roi[1]] = mask_roi

    segmented_rgb = cv2.bitwise_and(colorimage,colorimage,mask=mask)

    now = datetime.now()
    dt_str = now.strftime("%d-%m-%Y_%H-%M-%S.%f")
    sfn = 'seedlings_' + dt_str + '.png'
    dpath = os.path.join(plantpath,sfn)
    cv2.imwrite(dpath,segmented_rgb)
    sleep(0.5)
    print(sfn)
    return sfn

def class_pred_in_letters(pred): #from number-based seedling classification to letters
    if pred == 3:
        return "A"
    if pred == 2:
        return "B"
    if pred == 1:
        return "C"

def classify_pred(box, score): #classifications images according to calibrated thresholds
    print("Classifying predictions done by yolov3")
    x, y, w, h = box
```

```
    area = w * h

    if score < score_threshold: #It is suspected that it is not average seedling
        #then it is necessary to know if it is bad or good seedling
        if area < left_threshold:
            #bad seedling
            return 1#0
        if area > right_threshold:
            #good seeling
            return 3#2
        else:
            return 2#1  #TO BE CONSIDERED!
    else:
        #at least that it has a big area
        if area <= right_threshold: #area > left_threshold and area < right_threshold: #OJOOOO!!
take it into account
            #average seedling
            return 2#1
        if area > right_threshold:
            #good seedling
            return 3#2
        #else:
        #    return 1#0
    print("no pasa")

def cropimage(image,x_min,x_max,y_min,y_max,res_per):  #resizing images
    #per_res: resizing percentage
    #image = imageio.imread(path)
    #image = np.array(image)

    # crop the image using array slices -- it's a NumPy array
    # after all!
    cropped = image[y_min:y_max,x_min:x_max] #443:720, 403:1082]
    size=(int(cropped.shape[1]*res_per),int(cropped.shape[0]*res_per))  #image  size  proportion
720/1424 = 0.51
    #resize image
    #rsimg = cv2.resize(image,size) #this is really important to facilitate seedling manipulation
    cropped = cv2.resize(cropped,size)

    return np.array(cropped)

def imageremix(base_img,mask_img,rv,cv): #Put two images together
    print("Mixing 2 images")
    brv = rv #base image row value
    bcv = cv
    bimg = np.array(base_img)
    mimg = np.array(mask_img)
    mimg_r, mimg_c, _ = mimg.shape
    bimg_r, bimg_c, _ = bimg.shape
    limitflag_c = True
    #limitflag_r = True

    #if debug_mode:
    #    print("mask image row and column: " + str(mimg_r) + ", " + str(mimg_c))
    #    print("base image row and column: " + str(bimg_r) + ", " + str(bimg_c))

    for r in range(mimg_r):
        for c in range(mimg_c):
            if mimg[r,c,0] > 0: #different from zero
                if limitflag_c:
                    bimg[brv,bcv,:] = mimg[r,c,:]
            bcv += 1
            if bcv >= bimg_c: limitflag_c = False
            else: limitflag_c = True
        brv += 1
        bcv = cv
        if brv >= bimg_r: break

    return bimg

def get_prediction(modelpath, image_input):

    print("begin PREDICTION PROCESS")
    weight_path = os.path.join(modelpath, 'yolov3_training_last.weights')
```

```
    cfg_path = os.path.join(modelpath, 'yolov3_testing.cfg')
    net = cv2.dnn.readNet(weight_path, cfg_path)
    print("yolov3 loaded")
    classes = []
    with open("classes.txt", "r") as f:
        classes = f.read().splitlines()

    #cap = cv2.VideoCapture('video4.mp4')
    #cap = 'test_images/<your_test_image>.jpg'
    font = cv2.FONT_HERSHEY_PLAIN
    colors = np.random.uniform(0, 255, size=(100, 3))

    ############################################################
    img = cv2.imread(image_input) #"test_images/img19.png")
    ############################################################

    height, width, _ = img.shape
    if width > 416:
        img = cropimage(img,403,1082,443,720,0.5625) #according to 416x146pixel image size input
for yolov3 model
        img   =  imageremix(imageio.imread(os.path.join(modelpath,"size_ref.jpg")),img,int(679/2-
416/2),int(416/2-277/2))
        height, width, _ = img.shape

    print("height: " + str(height) + ", width: " + str(width))

    blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), (0,0,0), swapRB=True, crop=False)
    net.setInput(blob)
    output_layers_names = net.getUnconnectedOutLayersNames()
    layerOutputs = net.forward(output_layers_names)
    print("aquiiiiii")

    boxes = [(0,0,0,0),(0,0,0,0),(0,0,0,0)]
    confidences = [0,0,0]
    class_ids = [0,0,0]

    PM = np.ones([10,3,6]) #*-1 #Position matrix
    idx = [0,0,0] #np.zeros([1,3]) #with zeros considering 0 position at the beginning

    for output in layerOutputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.2:
                center_x = int(detection[0]*width)
                center_y = int(detection[1]*height)
                w = int(detection[2]*width)
                h = int(detection[3]*height)
                x = int(center_x - w/2)
                y = int(center_y - h/2)

                if center_x > g1min and center_x <= g1max and center_y >= hmin and center_y <=
hmax:
                    #1st group
                    PM[idx[0],0,:] = [classify_pred([x, y, w, h],confidence),x,y,w,h,confidence]
                    idx[0] += 1
                if center_x > g2min and center_x <= g2max and center_y >= hmin and center_y <=
hmax:
                    #2nd group
                    PM[idx[1],1,:] = [classify_pred([x, y, w, h],confidence),x,y,w,h,confidence]
                    idx[1] += 1
                if center_x > g3min and center_x <= g3max and center_y >= hmin and center_y <=
hmax:
                    #3th group
                    PM[idx[2],2,:] = [classify_pred([x, y, w, h],confidence),x,y,w,h,confidence]
                    idx[2] += 1
        print("por aquiiiii")

        #here it is necessary to treat boxes, confidences and class_ids as stablished array size
variables
        #in order to know if there is a cell with no seedling detection

        for pidx in range(3):
```

```
            val = -1
            print(idx[pidx])
            if idx[pidx] > 0: #if there is -1 value at position it must be recognized as no
seedling detection situation
                for i in range(idx[pidx]):
                    print("i: ", i)
                    c = PM[i,pidx,0] #this stage goes for discrimination in case more than one
seedling was detected and classified at the same cell
                    if c > val: #the greatest class is our guy!
                        indx = i
                        val = c

                ##################those     variables     could     be     initialized     as     -1
arrays#################3
                print("PM: ",PM[indx,pidx,1:5])
                boxes[pidx] = PM[indx,pidx,1:5]
                confidences[pidx] = float(PM[indx,pidx,5])
                class_ids[pidx] = PM[indx,pidx,0]
                #boxes.append(PM[indx,pidx,1:5])
                #confidences.append(float(PM[indx,pidx,5]))
                #class_ids.append(PM[indx,pidx,0])

            #else:
            #    boxes.append((0,0,0,0))
            #    confidences.append(0)
            #    class_ids.append(0)

            print("por aquiiiii 2")

        #boxes.append([x, y, w, h])
        #confidences.append((float(confidence)))
        #class_ids.append(class_id)

    print("saliooo")
    print("shape boxes: ", len(boxes))
    print(boxes[0])
    print(boxes[1])
    print(boxes[2])

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.2, 0.4)
    print("indexes: ", indexes)
    seedlfnames = [" "," "," "]

    #This is just for saving seedling images each one in different file
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            print("paso aquiiii")
            #label = str(classes[class_ids[i]])
            confidence = str(round(confidences[i],2))
            #color = colors[i]
            #cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)
            #cv2.putText(img, label + " " + confidence, (x, y+20), font, 1, (255,255,255), 2)

            crop_img = img[int(y):int(y+h+40), int(x):int(x+w+40)]
            now = datetime.now()
            dt_str = now.strftime("%d-%m-%Y_%H-%M-%S.%f")
            sfn = 'seedling_' + dt_str + '.png'
            dpath = os.path.join(seedldestpath,sfn)
            cv2.imwrite(dpath,crop_img)
            seedlfnames[i] = sfn
            #seedlfnames.append(sfn)

            #cv2.imshow("cropped", crop_img)
            #cv2.waitKey(0)

    #seedlfnames = np.array(seedlfnames)
    return seedlfnames, class_ids

def start_processing_image(img):
    print("Processing image")
    sn,sc = get_prediction(modelpath, img)
    return sn, sc
```

```
################################## CLASSES ###################################
class mqttModbusClient(mqtt.Client):
    def __init__(self,MODBUSSERVER_IP,MODBUSSERVER_PORT):
        mqtt.Client.__init__(self)
        self.topic = None
        self.payload = None
        self.modbusclient = SeedlingModbusClient(MODBUSSERVER_IP,MODBUSSERVER_PORT)
        self.g3min = 60
        self.g3max = 175
        self.g2min = 175
        self.g2max = 275
        self.g1min = 275
        self.g1max = 400
        self.hmin = 194
        self.hmax = 232
        self.score_threshold = 0.98
        self.left_threshold = 2614
        self.right_threshold = 4676
        self.topic4listenig = "robot/vision/ia/#"
        self.topic_prueba = "robot/vision/ia/prueba"
        self.topic_parametros = "robot/vision/ia/parametros"
        self.topic_ima1_ruta = "robot/vision/imagen1/ruta"
        self.topic_ima1_cali = "robot/vision/imagen1/calidad"
        self.topic_ima2_ruta = "robot/vision/imagen2/ruta"
        self.topic_ima2_cali = "robot/vision/imagen2/calidad"
        self.topic_ima3_ruta = "robot/vision/imagen3/ruta"
        self.topic_ima3_cali = "robot/vision/imagen3/calidad"
        self.MQTT_PATH = "cts/#"
        self.plantpath            =            '/home/erickmfs/ai_apps/seedling_groups/darknet-
yolov3/images/test_images/' #'/home/plantinator/plantinator/images/test_images'
        self.seedldestpath          =          '/home/erickmfs/ai_apps/seedling_groups/darknet-
yolov3/images/seedlings/' #'/home/root/vision'
        self.modelpath        =        "/home/erickmfs/ai_apps/seedling_groups/darknet-yolov3/"
#'/home/plantinator/plantinator'
        self.modbusclient.writeCvStatus(lsmodb.CV_WAITING_STAT)
        self.thread = Timer(0.2,self.check_plc)
        self.thread.start()
    def check_plc(self):
        #print(self.modbusclient.getPLCInstruction())
        #plcinstruction        =        lsmodb.PLC_PROCEVEN_INST        #plcinstruction        =
self.modbusclient.getPLCInstruction()
        if     (self.modbusclient.getPLCInstruction()     ==     lsmodb.PLC_PROCEVEN_INST)     |
(self.modbusclient.getPLCInstruction() == lsmodb.PLC_PROCODD_INST):
        #if(plcinstruction == lsmodb.PLC_PROCEVEN_INST)
            ### run classification
            self.modbusclient.writeCvStatus(lsmodb.CV_PROCESSING_STAT)  #  Tell  the  PLC  you've
received the instruction and you're processing.
            imgname    =    capture_and_segment_im()    #"img53.png"    #capture_and_segment_im()
#"img34.png"
            try:
                [seedlpath,                              seedlquality]                              =
start_processing_image(os.path.join(self.plantpath,imgname))
                self.publish(topic_ima1_ruta,seedlpath[0]) #"plantin1.png")
                self.publish(topic_ima1_cali,class_pred_in_letters(seedlquality[0]))    #"Calidad
A")
                self.publish(topic_ima2_ruta,seedlpath[1])  #"plantin2.png")
                self.publish(topic_ima2_cali,class_pred_in_letters(seedlquality[1]))    #"Calidad
B")
                self.publish(topic_ima3_ruta,seedlpath[2]) #"plantin3.png")
                self.publish(topic_ima3_cali,class_pred_in_letters(seedlquality[2]))    #"Calidad
A")
                print("qualities: {}".format(seedlquality))
                self.modbusclient.writeSeedling1Quality(int(seedlquality[0]))              #+1))
#lsmodb.QTY_A) # Define seedling 1 as an A-QUALITY one
                self.modbusclient.writeSeedling2Quality(int(seedlquality[1]))              #+1))
#lsmodb.QTY_A)  # Define seedling 2 as an A-QUALITY one
                self.modbusclient.writeSeedling3Quality(int(seedlquality[2]))              #+1))
#lsmodb.QTY_A)  # Define seedling 3 as an A-QUALITY one
                self.modbusclient.cvFinishProcessing() #Tell the PLC the processing has finished
            except:
                print(imgname)
                raise Exception("NAO DEU CERTINHO")
        #print("Thread OK")
        self.thread = Timer(0.2,self.check_plc)
```

```
        self.thread.start()
    def on_connect_(self, userdata, flags, rc):
        print("Connected MQTT with result code "+str(rc))
        print("()()()()()()---PLANTINATOR AI PROCESSING UNIT---()()()()()()")
        #print(client + "/" + userdata + "/")
        #client.publish(topic_es,CMD_SLV_ON)
        #print("ON SENT")
        # Subscribing in on_connect() means that if we lose the connection and
        # reconnect then subscriptions will be renewed.
        self.subscribe(self.topic4listenig)

    def on_message_(self, userdata, msg):
        self.topic = str(msg.topic)
        self.payload = str(msg.payload.decode('utf-8'))
        print("payload: " + self.payload)
        print("topic: " + self.topic)

        if self.topic == self.topic_parametros:
            #then parameters are going to be updated
            print("Updating parameters")
            self.updating_parameters(json.loads(self.payload))

    def updating_parameters(self,json_parameters_payload):
        self.g3min = json_parameters_payload["g3min"]
        self.g3max = json_parameters_payload["g3max"]
        self.g2min = json_parameters_payload["g2min"]
        self.g2max = json_parameters_payload["g2max"]
        self.g1min = json_parameters_payload["g1min"]
        self.g1max = json_parameters_payload["g1max"]
        self.hmin = json_parameters_payload["hmin"]
        self.hmax = json_parameters_payload["hmax"]
        self.score_threshold = json_parameters_payload["score_threshold"]
        self.left_threshold = json_parameters_payload["left_threshold"]
        self.right_threshold = json_parameters_payload["right_threshold"]

        print("Parameters updated:")
        print("g3min: ", self.g3min)
        print("g3max: ", self.g3max)
        print("g2min: ", self.g2min)
        print("g2max: ", self.g2max)
        print("g1min: ", self.g1min)
        print("g1max: ", self.g1max)
        print("hmin: ", self.hmin)
        print("hmax: ", self.hmax)
        print("score_threshold: ", self.score_threshold)
        print("left_threshold: ", self.left_threshold)
        print("right_threshold: ", self.right_threshold)



#######INICIO DEL PROGRAMA#########
#mqtt connection
client = mqttModbusClient(modServerIp,modServerPort)
#client = paho.Client()
client.username_pw_set(user, password)
#client.connect(“broker.mqttdashboard.com”)
client.on_connect = client.on_connect_
client.on_message = client.on_message_
client.connect(MQTT_SERVER, 1883, 60)
client.modbusclient.writeCvStatus(lsmodb.CV_WAITING_STAT)

######PLC MODBUS CONNECTION#####f
#cv_Client = SeedlingModbusClient(modServerIp,modServerPort)


#test of conection
client.publish(topic_prueba,"PLANTINATOR SERVER RUNNING MDFK!!")

print("SMITH,SHUT UP YOUR FUCKING MOUTH !!!!!!")


# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and a
```

```
# manual interface.
client.loop_forever()
```

Anexos XIV: Programa de la Interfaz de Usuario en Node-Red

Programa de Node-Red exportado en formato json para la ventana de parámetros y envío de parámetros: “PARAMETROS IA”


```
[{"id":"c8c5263f.88d628","type":"tab","label":"Parametros
IA","disabled":false,"info":""},{"id":"7c9ae1f8.5a1d4","type":"mqtt
out","z":"c8c5263f.88d628","name":"","topic":"robot/vision/ia/prueba","qos":"1","retain":"false"
,"broker":"503bfeb1.e6ce","x":600,"y":300,"wires":[]},{"id":"44418ed3.deb97","type":"ui_button",
"z":"c8c5263f.88d628","name":"","group":"a0064009.ae3c8","order":4,"width":0,"height":0,"passthr
u":false,"label":"Iniciar
prueba","tooltip":"","color":"","bgcolor":"","icon":"","payload":"iniciar","payloadType":"str","
topic":"","x":160,"y":300,"wires":[["7c9ae1f8.5a1d4","601a138.163f1ec"]]},{"id":"564635db.60f97c
","type":"comment","z":"c8c5263f.88d628","name":"Boton
prueba","info":"","x":110,"y":240,"wires":[]},{"id":"1b862b07.a2a875","type":"ui_form","z":"c8c5
263f.88d628","name":"","label":"","group":"7383834c.e9f1dc","order":1,"width":"0","height":"0","
options":[{"label":"G3min","value":"g3min","type":"number","required":true,"rows":null},{"label"
:"G3max","value":"g3max","type":"number","required":true,"rows":null},{"label":"G2min","value":"
g2min","type":"number","required":true,"rows":null},{"label":"G2max","value":"g2max","type":"num
ber","required":true,"rows":null},{"label":"G1min","value":"g1min","type":"number","required":tr
ue,"rows":null},{"label":"G1max","value":"g1max","type":"number","required":true,"rows":null},{"
label":"Score
threshold","value":"score_threshold","type":"number","required":true,"rows":null},{"label":"Left
threshold","value":"left_threshold","type":"number","required":true,"rows":null},{"label":"Right
threshold","value":"right_threshold","type":"number","required":true,"rows":null}],"formValue":{
"g3min":"","g3max":"","g2min":"","g2max":"","g1min":"","g1max":"","score_threshold":"","left_thr
eshold":"","right_threshold":""},"payload":"","submit":"submit","cancel":"","topic":"","x":390,"
y":100,"wires":[["a2377d49.441e3"]]},{"id":"1a9ebe84.311511","type":"inject","z":"c8c5263f.88d62
8","name":"Valores por
defecto","props":[{"p":"payload"}],"repeat":"","crontab":"","once":true,"onceDelay":"0.2","topic
":"","payload":"{\"g3min\":60,\"g3max\":175,\"g2min\":175,\"g2max\":275,\"g1min\":275,\"g1max\":
400,\"hmin\":194,\"hmax\":232,\"score_threshold\":0.98,\"left_threshold\":2614,\"right_threshold
\":4676}","payloadType":"json","x":190,"y":140,"wires":[["1b862b07.a2a875"]]},{"id":"a2377d49.44
1e3","type":"mqtt
out","z":"c8c5263f.88d628","name":"","topic":"robot/vision/ia/parametros","qos":"1","retain":"fa
lse","broker":"503bfeb1.e6ce","x":590,"y":100,"wires":[]},{"id":"bfac4785.414c58","type":"mqtt
in","z":"c8c5263f.88d628","name":"","topic":"robot/vision/ia/parametros","qos":"1","datatype":"a
uto","broker":"503bfeb1.e6ce","x":190,"y":100,"wires":[["1b862b07.a2a875"]]},{"id":"6a2bdfcb.41a
17","type":"comment","z":"c8c5263f.88d628","name":"Formulario","info":"","x":100,"y":40,"wires":
[]},{"id":"aebc82bb.7654e","type":"mqtt
in","z":"c8c5263f.88d628","name":"","topic":"robot/#","qos":"1","datatype":"auto","broker":"503b
feb1.e6ce","x":130,"y":420,"wires":[["9be1628f.a391c"]]},{"id":"601a138.163f1ec","type":"debug",
"z":"c8c5263f.88d628","name":"","active":true,"tosidebar":true,"console":false,"tostatus":false,
"complete":"false","statusVal":"","statusType":"auto","x":580,"y":260,"wires":[]},{"id":"13380e0
e.b883b2","type":"comment","z":"c8c5263f.88d628","name":"Lista de mensajes
recibidos","info":"","x":160,"y":380,"wires":[]},{"id":"9be1628f.a391c","type":"function","z":"c
8c5263f.88d628","name":"","func":"function addZero(i) {\n  if (i < 10) {\n    i = \"0\" + i;\n
}\n    return i;\n}\n\nvar t = new Date();\nt.setHours(t.getHours()-5);\nvar h =
addZero(t.getHours());\nvar m = addZero(t.getMinutes());\nvar s = addZero(t.getSeconds());\nvar
previousmessage = context.get('previousmsg');\nvar result = h + \":\" + m + \":\" + s + \": \" +
msg.topic + \": \" + msg.payload;\nif (previousmessage !== undefined) {\n    result = result +
\"\\n\" + previousmessage\n}\ncontext.set('previousmsg', result);\nmsg.payload = result\nreturn
msg;\n","outputs":1,"noerr":0,"initialize":"","finalize":"","x":360,"y":420,"wires":[["a88dedf1.
5e781"]]},{"id":"a88dedf1.5e781","type":"ui_template","z":"c8c5263f.88d628","group":"86bed859.39
0f38","name":"","order":0,"width":0,"height":0,"format":"<div ng-
value=\"\nrecibido=msg.payload;\n\">\nMensajes recibidos al topic \"robot/\"\n<textarea
style=\"height: 300px; width:
100%\">\n{{recibido}}\n</textarea>\n\n</div>","storeOutMessages":true,"fwdInMessages":true,"rese
ndOnRefresh":true,"templateScope":"local","x":640,"y":420,"wires":[[]]},{"id":"41a4930e.7b4734",
"type":"mqtt
in","z":"c8c5263f.88d628","name":"","topic":"robot/vision/ia/prueba","qos":"2","datatype":"auto"
,"broker":"14fc7b83.5e4234","x":320,"y":220,"wires":[["601a138.163f1ec"]]},{"id":"66234ec7.e0808
","type":"ui_template","z":"c8c5263f.88d628","group":"7383834c.e9f1dc","name":"Hoja estilos
formulario","order":1,"width":0,"height":0,"format":"<style>\n\n\n</style>","storeOutMessages":t
rue,"fwdInMessages":true,"resendOnRefresh":true,"templateScope":"local","x":460,"y":40,"wires":[
[]]},{"id":"503bfeb1.e6ce","type":"mqtt-
broker","name":"Local","broker":"localhost","port":"1883","clientid":"","usetls":false,"compatmo
de":false,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload":"",
"closeTopic":"","closeQos":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},
{"id":"a0064009.ae3c8","type":"ui_group","name":"Iniciar
```

```
prueba","tab":"1cdb442e.8e1c2c","order":6,"disp":true,"width":"6","collapse":false},{"id":"73838
34c.e9f1dc","type":"ui_group","name":"Parametros","tab":"1cdb442e.8e1c2c","order":5,"disp":true,
"width":"6","collapse":false},{"id":"86bed859.390f38","type":"ui_group","name":"Datos
recibidos","tab":"1cdb442e.8e1c2c","order":3,"disp":true,"width":"12","collapse":false},{"id":"1
4fc7b83.5e4234","type":"mqtt-
broker","name":"Local","broker":"localhost","port":"1883","clientid":"","usetls":false,"compatmo
de":false,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload":"",
"closeTopic":"","closeQos":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},
{"id":"1cdb442e.8e1c2c","type":"ui_tab","name":"Vision          artificial","icon":"fa-lightbulb-
o","order":2,"disabled":false,"hidden":false}]
```


Y el programa de la ventana “Dashboard / Bandejas”:


```
[{"id":"f8358b8d.f848d8","type":"tab","label":"Dashboard                                    /
Bandejas","disabled":false,"info":""},{"id":"4e8f76ad.435de8","type":"ui_template","z":"f8358b8d
.f848d8","group":"5a0d5658.57dff8","name":"bandeja","order":2,"width":0,"height":0,"format":"<di
v           ng-value=\"\nnumero=msg.payload;\nfilas1=(numero-numero%6)/6;\nfilas0=(72-numero-(72-
numero)%6)/6;\nfilasX=12-filas1-filas0;\nx1=numero%6;\nx0=6-x1\">\n<table>\n    <tr ng-repeat=\"r
in [].constructor(filas1) track by $index\">\n       <td ng-repeat=\"c in [].constructor(6) track
by  $index\"><span  class=\"dotg\"></span></td>\n         </tr>\n         <tr  ng-repeat=\"r  in
[].constructor(filasX) track by $index\">\n        <td ng-repeat=\"c in [].constructor(x1) track
by $index\"><span class=\"dotg\"></span></td>\n          <td ng-repeat=\"c in [].constructor(x0)
track  by  $index\"><span  class=\"dotw\"></td>\n         </tr>\n         <tr  ng-repeat=\"r  in
[].constructor(filas0) track by $index\">\n        <td ng-repeat=\"c in [].constructor(6) track
by                          $index\"><span                          class=\"dotw\"></td>\n
</tr>\n</table>\n</div>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"te
mplateScope":"local","x":480,"y":340,"wires":[[]],"icon":"font-awesome/fa-
columns"},{"id":"36546e97.9c4192","type":"ui_text","z":"f8358b8d.f848d8","group":"5a0d5658.57dff
8","order":3,"width":0,"height":0,"name":"","label":"Plantines          en          bandeja:
","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":380,"wires":[]},{"id":"2d7502ac.2ed8de","type":"ui_text","z":"f8358b8d.f848d
8","group":"5a0d5658.57dff8","order":4,"width":0,"height":0,"name":"","label":"Contador       de
bandejas:                                    ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":420,"wires":[]},{"id":"d7488598.fc7fe8","type":"ui_text","z":"f8358b8d.f848d
8","group":"5a0d5658.57dff8","order":1,"width":0,"height":0,"name":"","label":"Reposición
activada:                                    ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":300,"wires":[]},{"id":"dc21ee4c.5c06b","type":"ui_template","z":"f8358b8d.f8
48d8","group":"7192dba7.e31c04","name":"bandeja","order":2,"width":0,"height":0,"format":"<div
ng-value=\"\nnumero=msg.payload;\nfilas1=(numero-numero%6)/6;\nfilas0=(72-numero-(72-
numero)%6)/6;\nfilasX=12-filas1-filas0;\nx1=numero%6;\nx0=6-x1\">\n<table>\n    <tr ng-repeat=\"r
in [].constructor(filas1) track by $index\">\n       <td ng-repeat=\"c in [].constructor(6) track
by  $index\"><span  class=\"dotg\"></span></td>\n         </tr>\n         <tr  ng-repeat=\"r  in
[].constructor(filasX) track by $index\">\n        <td ng-repeat=\"c in [].constructor(x1) track
by $index\"><span class=\"dotg\"></span></td>\n          <td ng-repeat=\"c in [].constructor(x0)
track  by  $index\"><span  class=\"dotw\"></span></td>\n      </tr>\n       <tr  ng-repeat=\"r  in
[].constructor(filas0) track by $index\">\n        <td ng-repeat=\"c in [].constructor(6) track
by                          $index\"><span                          class=\"dotw\"></span></td>\n
</tr>\n</table>\n</div>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"te
mplateScope":"local","x":480,"y":540,"wires":[[]],"icon":"font-awesome/fa-
columns"},{"id":"1b82b622.bc2e2a","type":"ui_text","z":"f8358b8d.f848d8","group":"7192dba7.e31c0
4","order":3,"width":0,"height":0,"name":"","label":"Plantines          en          bandeja:
","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":580,"wires":[]},{"id":"3ba7cff8.ce40b","type":"ui_text","z":"f8358b8d.f848d8
","group":"7192dba7.e31c04","order":4,"width":0,"height":0,"name":"","label":"Contador       de
bandejas:                                    ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":620,"wires":[]},{"id":"97014d6e.b148","type":"ui_text","z":"f8358b8d.f848d8"
,"group":"7192dba7.e31c04","order":1,"width":0,"height":0,"name":"","label":"Reposición
activada:                                    ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":500,"wires":[]},{"id":"63efd59f.2e131c","type":"ui_template","z":"f8358b8d.f
848d8","group":"555aa5e.8f0f45c","name":"bandeja","order":3,"width":0,"height":0,"format":"<div
ng-value=\"\nnumero=msg.payload;\nfilas1=(numero-numero%6)/6;\nfilas0=(72-numero-(72-
numero)%6)/6;\nfilasX=12-filas1-filas0;\nx1=numero%6;\nx0=6-x1;\">\n<table>\n          <tr  ng-
repeat=\"r  in  [].constructor(filas0)  track  by  $index\">\n             <td  ng-repeat=\"c  in
[].constructor(6) track by $index\"><span class=\"dotw\"></span></td>\n     </tr>\n     <tr ng-
repeat=\"r  in  [].constructor(filasX)  track  by  $index\">\n             <td  ng-repeat=\"c  in
[].constructor(6) track by $index\">\n            <span ng-if=\"($index%2==1 && x1>$index/2) ||
($index%2==0 && x1-3>$index/2)\" class=\"dotgray\"></span>\n        </td>\n    </tr>\n    <tr ng-
repeat=\"r  in  [].constructor(filas1)  track  by  $index\">\n             <td  ng-repeat=\"c  in
[].constructor(6)      track      by      $index\"><span      class=\"dotgray\"></span></td>\n
</tr>\n</table>\n</div>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"te
mplateScope":"local","x":480,"y":160,"wires":[[]],"icon":"font-awesome/fa-
```

```
columns"},{"id":"e268da2d.2a0888","type":"ui_text","z":"f8358b8d.f848d8","group":"555aa5e.8f0f45
c","order":4,"width":0,"height":0,"name":"","label":"Plantines en bandeja:
","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":200,"wires":[]},{"id":"7568ffea.69f84","type":"ui_text","z":"f8358b8d.f848d8
","group":"555aa5e.8f0f45c","order":5,"width":0,"height":0,"name":"","label":"Contador de
bandejas: ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":240,"wires":[]},{"id":"7b12efd8.a9f01","type":"ui_text","z":"f8358b8d.f848d8
","group":"555aa5e.8f0f45c","order":2,"width":0,"height":0,"name":"","label":"Reposición
activada: ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":120,"wires":[]},{"id":"ac3b3f44.d6156","type":"ui_template","z":"f8358b8d.f8
48d8","group":"9f8f7dd6.f0dd1","name":"bandeja","order":2,"width":0,"height":0,"format":"<div
ng-value=\"\nnumero=msg.payload;\nfilas1=(numero-numero%6)/6;\nfilas0=(72-numero-(72-
numero)%6)/6;\nfilasX=12-filas1-filas0;\nx1=numero%6;\nx0=6-x1\">\n<table>\n <tr ng-repeat=\"r
in [].constructor(filas1) track by $index\">\n <td ng-repeat=\"c in [].constructor(6) track
by $index\"><span class=\"dotg\"></span></td>\n </tr>\n <tr ng-repeat=\"r in
[].constructor(filasX) track by $index\">\n <td ng-repeat=\"c in [].constructor(x1) track
by $index\"><span class=\"dotg\"></span></td>\n <td ng-repeat=\"c in [].constructor(x0)
track by $index\"><span class=\"dotw\"></span></td>\n </tr>\n <tr ng-repeat=\"r in
[].constructor(filas0) track by $index\">\n <td ng-repeat=\"c in [].constructor(6) track
by $index\"><span class=\"dotw\"></span></td>\n
</tr>\n</table>\n</div>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"te
mplateScope":"local","x":480,"y":740,"wires":[[]],"icon":"font-awesome/fa-
columns"},{"id":"37cc3dc2.934182","type":"ui_text","z":"f8358b8d.f848d8","group":"9f8f7dd6.f0dd1
","order":3,"width":0,"height":0,"name":"","label":"Plantines en bandeja:
","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":780,"wires":[]},{"id":"c7dc7ab8.8e8f18","type":"ui_text","z":"f8358b8d.f848d
8","group":"9f8f7dd6.f0dd1","order":4,"width":0,"height":0,"name":"","label":"Contador de
bandejas: ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":820,"wires":[]},{"id":"61eed324.9be9ec","type":"ui_text","z":"f8358b8d.f848d
8","group":"9f8f7dd6.f0dd1","order":1,"width":0,"height":0,"name":"","label":"Reposición
activada: ","format":"{{msg.payload}}","layout":"row-
center","x":520,"y":700,"wires":[]},{"id":"58931ffd.eb37a","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/alimentadora/porprocesar","qos":"1",
"datatype":"auto","broker":"a0cc122.a2aaff","x":200,"y":200,"wires":[["63efd59f.2e131c","e268da2
d.2a0888"]]},{"id":"b1fe2711.55f098","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/alimentadora/reposicion","qos":"1","
datatype":"auto","broker":"a0cc122.a2aaff","x":190,"y":120,"wires":[["7b12efd8.a9f01"]]},{"id":"
70ccc733.d249b8","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/alimentadora/bandejas","qos":"1","da
tatype":"auto","broker":"a0cc122.a2aaff","x":190,"y":240,"wires":[["7568ffea.69f84"]]},{"id":"16
3bcdae.fc5e42","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseA/reposicion","qos":"1","dataty
pe":"auto","broker":"a0cc122.a2aaff","x":210,"y":300,"wires":[["d7488598.fc7fe8"]]},{"id":"3bbfe
cb.95d2514","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseA/cantidad","qos":"1","datatype
":"auto","broker":"a0cc122.a2aaff","x":210,"y":380,"wires":[["4e8f76ad.435de8","36546e97.9c4192"
]]},{"id":"b44da543.e6ec18","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseA/bandejas","qos":"1","datatype
":"auto","broker":"a0cc122.a2aaff","x":210,"y":420,"wires":[["2d7502ac.2ed8de"]]},{"id":"c96578f
d.5d8bc8","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseB/reposicion","qos":"1","dataty
pe":"auto","broker":"a0cc122.a2aaff","x":210,"y":500,"wires":[["97014d6e.b148"]]},{"id":"4ef9ee6
8.5321","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseB/cantidad","qos":"1","datatype
":"auto","broker":"a0cc122.a2aaff","x":210,"y":580,"wires":[["dc21ee4c.5c06b","1b82b622.bc2e2a"]
]},{"id":"c6c88c82.e1eaf","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseB/bandejas","qos":"1","datatype
":"auto","broker":"a0cc122.a2aaff","x":210,"y":620,"wires":[["3ba7cff8.ce40b"]]},{"id":"fe1410ec
.4c583","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseC/reposicion","qos":"1","dataty
pe":"auto","broker":"a0cc122.a2aaff","x":210,"y":700,"wires":[["61eed324.9be9ec"]]},{"id":"3fbfe
20f.87047e","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseC/cantidad","qos":"1","datatype
":"auto","broker":"a0cc122.a2aaff","x":210,"y":780,"wires":[["ac3b3f44.d6156","37cc3dc2.934182"]
]},{"id":"e09b3442.80b1b8","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/bandejas/claseC/bandejas","qos":"1","datatype
":"auto","broker":"a0cc122.a2aaff","x":210,"y":820,"wires":[["c7dc7ab8.8e8f18"]]},{"id":"964fa29
3.3fe74","type":"ui_template","z":"f8358b8d.f848d8","group":"555aa5e.8f0f45c","name":"Hoja de
estilos","order":1,"width":0,"height":0,"format":"<style>\n.dotg {\n height: 14px;\n width:
14px;\n background-color: #44AA44;\n border-radius: 50%;\n display: block;\n}\n.dotgray {\n
height: 14px;\n width: 14px;\n background-color: #7C7C7C;\n border-radius: 50%;\n display:
block;\n}\n.dotw {\n height: 14px;\n width: 14px;\n background-color: #FFFFFF;\n border-radius:
50%;\n display: block;\n}\ntable, th, td {\n border: 1px solid #CCCCAA;\n border-collapse:
collapse;\n font-size: small;\n padding: 1px 8px 1px 8px;\n margin-left: auto;\n margin-
```

```
right: auto;\n    margin-top: 8px;\n}\n.nr-dashboard-cardtitle {\n    text-align: center;\n}\n.nr-
dashboard-cardtitle  {\n       margin-bottom:  -7px  !important;\n}\n.nr-dashboard-cardpanel  {\n
margin-bottom:
20px;\n}\n</style>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"templat
eScope":"global","x":500,"y":20,"wires":[[]]},{"id":"c9cf2e73.db8c7","type":"ui_template","z":"f
8358b8d.f848d8","group":"3f6b96dd.87d91a","name":"Imagen                                    plantin
1","order":1,"width":0,"height":0,"format":"<img src=\"/{{msg.payload}}\" \nstyle=\"width: 78px;
height:      78px;      margin-left:      auto;\n                        margin-right:      auto\"
/>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"templateScope":"local",
"x":660,"y":1000,"wires":[[]]},{"id":"aeea4c97.bc93e","type":"ui_template","z":"f8358b8d.f848d8"
,"group":"7436b4ab.081ecc","name":"Imagen                                    plantin
2","order":1,"width":0,"height":0,"format":"<img src=\"/{{msg.payload}}\" \nstyle=\"width: 78px;
height:      78px;      margin-left:      auto;\n                        margin-right:      auto\"
/>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"templateScope":"local",
"x":660,"y":1140,"wires":[[]]},{"id":"9f2abb98.267f88","type":"ui_template","z":"f8358b8d.f848d8
","group":"626a0af4.748ee4","name":"Imagen                                    plantin
3","order":1,"width":0,"height":0,"format":"<img src=\"/{{msg.payload}}\" \nstyle=\"width: 78px;
height:      78px;      margin-left:      auto;\n                        margin-right:      auto\"
/>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":true,"templateScope":"local",
"x":660,"y":1280,"wires":[[]]},{"id":"258c077e.dbf418","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/vision/imagen1/ruta","qos":"1","datatype":"au
to","broker":"a0cc122.a2aaff","x":170,"y":1000,"wires":[["8d2a41f.d6fc3c"]]},{"id":"8d2a41f.d6fc
3c","type":"fs-ops-
copy","z":"f8358b8d.f848d8","name":"","sourcePath":"/home/root/vision","sourcePathType":"str","s
ourceFilename":"payload","sourceFilenameType":"msg","destPath":"/home/root/nodered-
static","destPathType":"str","destFilename":"plantin1.jpg","destFilenameType":"str","link":false
,"overwrite":true,"x":360,"y":1000,"wires":[["c269ac1f.05196"]]},{"id":"d3d24eea.ed007","type":"
mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/vision/imagen2/ruta","qos":"1","datatype":"au
to","broker":"a0cc122.a2aaff","x":170,"y":1140,"wires":[["d96f1b4f.b29b78"]]},{"id":"d73e9939.7f
0af8","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/vision/imagen3/ruta","qos":"1","datatype":"au
to","broker":"a0cc122.a2aaff","x":170,"y":1280,"wires":[["57ff5882.328df8"]]},{"id":"57ff5882.32
8df8","type":"fs-ops-
copy","z":"f8358b8d.f848d8","name":"","sourcePath":"/home/root/vision","sourcePathType":"str","s
ourceFilename":"payload","sourceFilenameType":"msg","destPath":"/home/root/nodered-
static","destPathType":"str","destFilename":"plantin3.jpg","destFilenameType":"str","link":false
,"overwrite":true,"x":360,"y":1280,"wires":[["7637e9f9.912cf8"]]},{"id":"d96f1b4f.b29b78","type"
:"fs-ops-
copy","z":"f8358b8d.f848d8","name":"","sourcePath":"/home/root/vision","sourcePathType":"str","s
ourceFilename":"payload","sourceFilenameType":"msg","destPath":"/home/root/nodered-
static","destPathType":"str","destFilename":"plantin2.jpg","destFilenameType":"str","link":false
,"overwrite":true,"x":360,"y":1140,"wires":[["2554474b.6341c8"]]},{"id":"9cbf32e2.5a3ad","type":
"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/vision/imagen1/calidad","qos":"1","datatype":
"auto","broker":"a0cc122.a2aaff","x":180,"y":1060,"wires":[["3e6f3977.efe766"]]},{"id":"3e6f3977
.efe766","type":"ui_text","z":"f8358b8d.f848d8","group":"3f6b96dd.87d91a","order":2,"width":0,"h
eight":0,"name":"","label":"Calidad:                ","format":"{{msg.payload}}","layout":"row-
center","x":580,"y":1060,"wires":[]},{"id":"6502eb08.839f34","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/vision/imagen2/calidad","qos":"1","datatype":
"auto","broker":"a0cc122.a2aaff","x":180,"y":1200,"wires":[["2ad6745a.d9f49c"]]},{"id":"2ad6745a
.d9f49c","type":"ui_text","z":"f8358b8d.f848d8","group":"7436b4ab.081ecc","order":2,"width":0,"h
eight":0,"name":"","label":"Calidad:                ","format":"{{msg.payload}}","layout":"row-
center","x":580,"y":1200,"wires":[]},{"id":"5118efd5.67132","type":"mqtt
in","z":"f8358b8d.f848d8","name":"","topic":"robot/vision/imagen3/calidad","qos":"1","datatype":
"auto","broker":"a0cc122.a2aaff","x":180,"y":1340,"wires":[["97fbf771.4ea0a8"]]},{"id":"97fbf771
.4ea0a8","type":"ui_text","z":"f8358b8d.f848d8","group":"626a0af4.748ee4","order":2,"width":0,"h
eight":0,"name":"","label":"Calidad:                ","format":"{{msg.payload}}","layout":"row-
center","x":580,"y":1340,"wires":[]},{"id":"861daa3.589d058","type":"ui_text","z":"f8358b8d.f848
d8","group":"98830236.798aa","order":0,"width":0,"height":0,"name":"","label":"Valor    de    la
alarma","format":"{{msg.payload}}","layout":"row-
center","x":570,"y":1420,"wires":[]},{"id":"23b843a3.0ad3cc","type":"ui_template","z":"f8358b8d.
f848d8","group":"392a1b43.4e3504","name":"Logo                                    /
Clock","order":0,"width":0,"height":0,"format":"<script                        id=\"clockScript1\"
type=\"text/javascript\">\n    var clockInterval;\n    $(function () {\n        if (clockInterval)
return;\n\n        //add logo\n        var div1 = $('<div>LABINM ROBÓTICA </div>');\n        var
logo = new Image();\n\n            logo.src = '/imagen1.jpg'\n            logo.height = 45;\n
div1[0].style.margin = '10px auto';\n\n          //div1.append(logo);\n\n          //add clock\n
var div2 = $('<div/>');\n              var p = $('<p/>');\n\n              div2.append(p);\n
div2[0].style.margin = '5px';\n\n           function displayTime() {\n                p.text(new
Date().toLocaleString());\n        }\n        \n        clockInterval = setInterval(displayTime,
1000);\n\n       //add to toolbar when it's available\n       var addToToolbarTimer;\n       \n
function addToToolbar() {\n             var toolbar = $('.md-toolbar-tools');\n             \n
if(!toolbar.length)  return;\n                    \n                    toolbar.append(div1);\n
```

```
toolbar.append(div2);\n                    clearInterval(addToToolbarTimer);\n            }\n
addToToolbarTimer            =            setInterval(addToToolbar,            100);\n
});\n</script>","storeOutMessages":true,"fwdInMessages":true,"resendOnRefresh":false,"templateSc
ope":"global","x":330,"y":20,"wires":[[]]},{"id":"32d5f662.badf4a","type":"inject","z":"f8358b8d
.f848d8","name":"","props":[{"p":"payload"}],"repeat":"","crontab":"","once":true,"onceDelay":0.
1,"topic":"","payload":"sinplantin.jpg","payloadType":"str","x":400,"y":1180,"wires":[["aeea4c97
.bc93e"]]},{"id":"4b1dc9b7.c92d78","type":"inject","z":"f8358b8d.f848d8","name":"","props":[{"p"
:"payload"}],"repeat":"","crontab":"","once":true,"onceDelay":0.1,"topic":"","payload":"sinplant
in.jpg","payloadType":"str","x":400,"y":1320,"wires":[["9f2abb98.267f88"]]},{"id":"839524cf.a29d
18","type":"inject","z":"f8358b8d.f848d8","name":"","props":[{"p":"payload"}],"repeat":"","cront
ab":"","once":true,"onceDelay":0.1,"topic":"","payload":"0","payloadType":"num","x":290,"y":160,
"wires":[["63efd59f.2e131c"]]},{"id":"ad1c5ac.065e9a8","type":"inject","z":"f8358b8d.f848d8","na
me":"","props":[{"p":"payload"}],"repeat":"","crontab":"","once":true,"onceDelay":0.1,"topic":""
,"payload":"0","payloadType":"num","x":290,"y":340,"wires":[["4e8f76ad.435de8"]]},{"id":"57de520
c.7ffc7c","type":"inject","z":"f8358b8d.f848d8","name":"","props":[{"p":"payload"}],"repeat":"",
"crontab":"","once":true,"onceDelay":0.1,"topic":"","payload":"0","payloadType":"num","x":290,"y
":540,"wires":[["dc21ee4c.5c06b"]]},{"id":"a444d616.b0c7a8","type":"inject","z":"f8358b8d.f848d8
","name":"","props":[{"p":"payload"}],"repeat":"","crontab":"","once":true,"onceDelay":0.1,"topi
c":"","payload":"0","payloadType":"num","x":290,"y":740,"wires":[["ac3b3f44.d6156"]]},{"id":"6b7
40027.51c5b","type":"comment","z":"f8358b8d.f848d8","name":"Imagenes            de            los
plantines","info":"","x":110,"y":940,"wires":[]},{"id":"5723180f.6d4238","type":"comment","z":"f
8358b8d.f848d8","name":"Bandejas","info":"","x":60,"y":60,"wires":[]},{"id":"c269ac1f.05196","ty
pe":"function","z":"f8358b8d.f848d8","name":"","func":"msg.payload = \"plantin1.jpg?r=\" + (new
Date().getTime());\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":500,"y":1000,"wires":[["c9cf2e73.d
b8c7"]]},{"id":"17f4e905.85f3d7","type":"inject","z":"f8358b8d.f848d8","name":"","props":[{"p":"
payload"}],"repeat":"","crontab":"","once":true,"onceDelay":0.1,"topic":"","payload":"sinplantin
.jpg","payloadType":"str","x":420,"y":1040,"wires":[["c9cf2e73.db8c7"]]},{"id":"2554474b.6341c8"
,"type":"function","z":"f8358b8d.f848d8","name":"","func":"msg.payload = \"plantin2.jpg?r=\" +
(new                                                    Date().getTime());\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":500,"y":1140,"wires":[["aeea4c97.b
c93e"]]},{"id":"7637e9f9.912cf8","type":"function","z":"f8358b8d.f848d8","name":"","func":"msg.p
ayload        =        \"plantin3.jpg?r=\"        +        (new        Date().getTime());\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":500,"y":1280,"wires":[["9f2abb98.2
67f88"]]},{"id":"5a0d5658.57dff8","type":"ui_group","name":"Bandeja                                Clase
A","tab":"2b75e1c6.f29e9e","order":2,"disp":true,"width":"5","collapse":false},{"id":"7192dba7.e
31c04","type":"ui_group","name":"Bandeja                                                          Clase
B","tab":"2b75e1c6.f29e9e","order":3,"disp":true,"width":"5","collapse":false},{"id":"555aa5e.8f
0f45c","type":"ui_group","name":"Bandeja
Alimentadora","tab":"2b75e1c6.f29e9e","order":1,"disp":true,"width":"5","collapse":false},{"id":
"9f8f7dd6.f0dd1","type":"ui_group","name":"Bandeja                                                 Clase
C","tab":"2b75e1c6.f29e9e","order":4,"disp":true,"width":"5","collapse":false},{"id":"a0cc122.a2
aaff","type":"mqtt-
broker","name":"Local","broker":"localhost","port":"1883","clientid":"","usetls":false,"compatmo
de":false,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload":"",
"closeTopic":"","closeQos":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},
{"id":"3f6b96dd.87d91a","type":"ui_group","name":"Imagen                                          plantin
1","tab":"2b75e1c6.f29e9e","order":5,"disp":true,"width":5,"collapse":false},{"id":"7436b4ab.081
ecc","type":"ui_group","name":"Imagen                                                            plantin
2","tab":"2b75e1c6.f29e9e","order":6,"disp":true,"width":5,"collapse":false},{"id":"626a0af4.748
ee4","type":"ui_group","name":"Imagen                                                            plantin
3","tab":"2b75e1c6.f29e9e","order":7,"disp":true,"width":5,"collapse":false},{"id":"98830236.798
aa","type":"ui_group","name":"Alarmas","tab":"2b75e1c6.f29e9e","order":8,"disp":true,"width":5,"
collapse":false},{"id":"392a1b43.4e3504","type":"ui_group","name":"Default","tab":"cb85678b.0456
d8","order":1,"disp":false,"width":"24","collapse":false},{"id":"2b75e1c6.f29e9e","type":"ui_tab
","name":"Bandejas","icon":"filter_vintage","order":1,"disabled":false,"hidden":false},{"id":"cb
85678b.0456d8","type":"ui_tab","name":"Home","icon":"dashboard","order":5}]
```

Anexos XV: Tabla de desempeño de clasificación de 700 plantines

Tabla resultante de la clasificación de plantines en el último experimento realizado para la correlación de los modelos de predicción tomando como referencia el trabajo de clasificación hecho por los jornaleros de un vivero industrial de la Región La Libertad.

#Bandeja	#plantines	Calidad Buena	Calidad Regular	Calidad Mala	Aciertos	Desaciertos
1	54	32	13	9	38	16
2	60	39	15	6	45	15
3	65	48	10	7	39	26
4	51	35	11	5	46	5
5	48	27	10	11	40	8
6	67	47	12	8	42	25
7	62	43	9	10	41	21
8	53	38	9	6	34	19
9	55	35	15	5	36	19
10	64	36	16	12	50	14
11	40	21	11	8	35	5
TOTAL	619	401	131	87	446	173