EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE DURANGO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

INSTITUTO TECNOLÓGICO DEL VALLE DEL GUADIANA

Dispositivo de escaneo para el cálculo de masa de una estiba de costales

TESIS

Que como parte de los requisitos para obtener el grado de:

Maestría en Ingeniería

Presenta:

Nancy Vázquez Morales

Director de tesis:

Dr. Efraín Ibarra Jiménez

Co-Director:

Dr. Fernando Blanco Castañeda

Durango, Dgo., México, noviembre 2022

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Dispositivo de escaneo para el cálculo de masa de una estiba de costales

TESIS

Que como parte de los requisitos para obtener el grado de

Maestría en Ingeniería

Presenta:

Ing. Nancy Vázquez Morales

Dirigido Por

Dr. Efraín Ibarra Jiménez

COMITÉ TUTORIAL

Dr. Efraín Ibarra Jiménez Director	Efrain Ibarra J. Firma
Dr. Fernando Blanco Castañeda Codirector	Firma
Dr. Rubén Guerrero Rivera Asesor	Firma

M. C. Norma Alicia García Vidaña
Nombre y firma
Coordinador de la Maestría en Ingeniería

M.C. Adriana Erendira Murillo
Nombre y firma
Jefe(a) de la División de Estudios de Posgrado e Investigación

Durango, Dgo. México, noviembre, 2022

Autorización de tema de Tesis de Maestría

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Durango
División de Estudios de Posgrado e Investigación

Victoria de Durango, Dgo., a 04 / Noviembre / 2022.

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
DEPI / C / 457 / 22.
ASUNTO: Autorización de Tema de Tesis de Maestría.

C. NANCY VÁZQUEZ MORALES
No. DE CONTROL G15041110
P R E S E N T E .

Con base en el Reglamento en vigor y teniendo en cuenta el dictamen emitido por el Jurado que le fue asignado, se le autoriza a desarrollar el tema de tesis para obtener el **Grado de Maestra en Ingeniería** cuyo título es:

"Dispositivo de escaneo para el cálculo de masa de una estiba de costales"

CONTENIDO:

A T E N T A M E N T E.

Excelencia en Educación Tecnológica®
"La Técnica al Servicio de la Patria"

C. ADRIANA ERÉNDIRA MURILLO
JEFA DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE DURANGO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

AEM'ammc.

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Autorización de impresión de Tesis de Maestría

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Durango
División de Estudios de Posgrado e Investigación

Victoria de Durango, Dgo., a 04 / Noviembre / 2022.

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
DEPI / C / 458 / 22.
ASUNTO: Autorización de Impresión de Tesis de Maestría.

C. NANCY VÁZQUEZ MORALES
No. DE CONTROL G15041110
P R E S E N T E .

De acuerdo al reglamento en vigor y tomando en cuenta el dictamen emitido por el jurado que le fue asignado para la revisión de su trabajo de tesis para obtener el **Grado de Maestra en Ingeniería,** esta División de Estudios de Posgrado e Investigación le autoriza la impresión del mismo, cuyo título es:

"Dispositivo de escaneo para el cálculo de masa de una estiba de costales"

Sin otro particular de momento, quedo de Usted.

A T E N T A M E N T E.

Excelencia en Educación Tecnológica®
"La Técnica al Servicio de la Patria"

C. ADRIANA ERÉNDIRA MURILLO
JEFA DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE DURANGO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

AEM'ammc.

Av. Felipe Pescador #1830 Ote. Col. Nueva Vizcaya C.P.34080 Durango, Durango.
Tel. (618) 8290900 e-mail: dir_itdurango@tecnm.mx tecnm.mx | itdurango.edu.mx

2022 Ricardo Flores Magón
Año de

Agradecimientos

Este proyecto fue posible gracias al apoyo del Dr. Efraín Ibarra Jiménez y al Instituto Tecnológico de Durango, que me brindaron tanto la guía como equipo el para el desarrollo del dispositivo. De la misma forma, agradezco profundamente a mi familia por toda su ayuda y cariño que me sirvió para poder superar los obstáculos y dificultades que se iban presentado. También al Ing. Ricardo Zamudio Carbajal que con sus conocimientos, ideas y apoyo moral me dio la fuerza para persistir cuando no salían los resultados como se esperaban.

Asimismo, al Dr. Adrián Manzanilla Magallanes y al Mtro. Edgar López Díaz por sus invaluables consejos que sirvieron de sustento para abordar la problemática del proyecto con distintos puntos de vista, que me permitieron complementar la óptica que inicialmente tenía planteada en el proyecto, así como su apoyo para la realización de pruebas con la minicomputadora NUC. Además, al Dr. Rubén Guerrero Rivera que con su experiencia en redes neuronales me ayudo a incorporar diversas mejoras al proyecto. Finalmente, le doy un agradecimiento especial al Consejo Nacional de Ciencia y Tecnología de México (CONACyT) por el soporte financiero que me han otorgado para poder llevar a cabo mis estudios de posgrado, en forma de una beca, de acuerdo al número 1079739.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En esta tesis se desarrolla un sistema de detección y conteo de costales de granos para ayudar a la gestión logística del stock de un almacén. El objetivo final consiste en construir un dispositivo electrónico, que sea capaz de ejecutar el conteo de costales con gran precisión, todo esto, mediante las técnicas más avanzadas de visión artificial.

Dicho dispositivo tendrá la tarea de contabilizar el total de costales que conforman una estiba completa, asimismo, calculará el volumen y la cantidad de masa estimada, estos resultados pasarán a una hoja de cálculo de Excel para tener un monitoreo constante y de fácil manejo para los usuarios.

El modelo de detección es realizado con Python, PyTorch y YOLOv3, donde se obtuvo como resultado un mAP de 0.9235 y un f1 de 0.9555, además, se propuso un acomodo que permite que el resultado del volumen sea lo más exacto posible, igualmente se desarrolló un programa que lleva a cabo este cálculo.

CAPÍTULO I
INTRODUCCIÓN

Hoy en día las redes neuronales artificiales se encuentran en todos sectores económicos desde el entretenimiento, agricultura, medicina entre otros, estas redes están relacionadas con el Aprendizaje profundo, el cual se llama así debido a que entre mayor sea la cantidad de capas conectadas entre sí se obtiene un aprendizaje más fino y mejor. La inteligencia artificial ha resuelto innumerables problemas, por lo que en este documento se busca hacer uso de ella para mejorar el control de inventario de un almacén de granos en donde se guardan costales de frijoles acomodados en estibas. Algunos proyectos de detección y conteo comentan que una posible mejora es el uso de YOLO, ya que mejora la velocidad de procesamiento, asimismo mantiene una precisión similar a obtenida con otras técnicas de gran exactitud.

Actualmente la inteligencia artificial está en los diversos sectores económicos donde todas las personas interactúan constantemente con ella, de manera consciente o inconsciente. Como ejemplo particular, la IA es utilizada para mejorar el mercado de compra y venta, logrando que las empresas o personas físicas puedan obtener un mayor margen de ganancias al momento de ofrecer sus productos por internet, esto se debe a que IA es capaz de identificar patrones de gustos en todos los usuarios que navegan por la web, ya que puede monitorear en tiempo real el historial de búsqueda, identificando los sitios web más visitados o los videos de YouTube con más visualizaciones, aunque pudiera parecer una invasión a la privacidad, la IA de esta manera puede saber los gustos personales, y posteriormente enviar notificaciones de videos o de algún artículo a la venta que pudiera ser de interés. Así mismo la IA tiene aplicaciones en sectores tales como la medicina, entretenimiento, agricultura, entre otros.

1.1. Antecedentes

El concepto de inteligencia artificial (IA) nació en 1956 por John McCarthy, Minsky y Claude Shannon, así mismo la primera red neuronal fue creada en 1957 por Frank Rosenblatt, con esto en mente, los creadores del concepto de IA pensaron que no pasaría mucho tiempo para que esta tecnología estuviera conviviendo cotidianamente con las personas, no obstante, no fue hasta la década de los 80's que la inteligencia artificial comenzó a expandirse con el uso de un kit de herramientas algorítmicas y fondos más exclusivos.

John Hopefield y David Rumelhart introdujeron las técnicas de "Aprendizaje Profundo" conocidas en inglés como "Deep Learning" que permiten a las computadoras aprender mediante la experiencia, pero no fue hasta los años 90's que la inteligencia artificial empezó a tener un gran impulso tal y como se conoce hoy en día. Este impulso fue causado principalmente por el repentino crecimiento de la tecnología industrial, la cual se vería afectada y podría quedar obsoleta si la IA no se actualizaba a la par de sus necesidades. Cabe mencionar, que un hecho que realmente marcó esta época sucedió en el año 1997, cuando IBM mostró un ordenador con IA capaz de ganar una partida de ajedrez al campeón del mundo Gari Kasparov, esta super computadora se llamó Deep Blue.

1.1.1. Antecedentes de la IA en la agricultura

A continuación, se muestran algunos artículos de investigación que han utilizado las técnicas más exclusivas del Aprendizaje Profundo en la detección y conteo de objetos, para utilizarse en aplicaciones enfocadas a la agricultura, dando a conocer un resumen especifico de cada una de estas investigaciones.

Sistema de conteo de costales usando análisis de video

Se propone desarrollar un sistema de detección y conteo de objetos utilizando tecnología de visión por computadora. El articulo propone un sistema de detección de conteo de costales, con el objetivo de reducir el esfuerzo humano involucrado durante el seguimiento y conteo de costales para la gestión logística de una empresa.

Para lograr este objetivo se utilizaron algoritmos y métodos de procesamiento de imágenes en la transmisión de video de la cámara de un almacén. Cabe mencionar que la detección de los objetos fue hecha aplicando un algoritmo R-CNN (Region Based Convolutional Network), para determinar si un saco está o no presente en la imagen, la cual es extraída de la transmisión del video de una cámara CCTV. Asimismo, para el desarrollo del "Aprendizaje Profundo" se utilizó la plataforma OpenCV y la librería *tensorflow*. Todos los detalles del desarrollo de esta investigación pueden visualizarse en [1].

Detección y conteo automatizados de agave utilizando una red neuronal convolucional y sistemas aéreos no tripulados

Se utiliza un método de "Aprendizaje Profundo" para detectar y contar plantas de agave, lo cual da como resultado una validación del algoritmo que muestra un 96% de detección real y conteo individual, para esto se aplica un algoritmo *CNN* (Convolucional Neural Network), con el cual se obtiene un promedio de detección correcta del 98%, mismo que se compara con el método de *haar classifier* que obtiene solo un 42%, demostrando que este método de detección y conteo posee una precisión que permite extender el estudio de la planta de agave para también detectar la edad de esta, identificando si ya está madura y lista para ser cosechada. Se ve la opción de posible mejora al hacer uso de

YOLO, pues podría mejorar la velocidad de procesamiento con una precisión similar a la del método propuesto. Los detalles precisos de esta investigación pueden verse en [2].

Detección y conteo de frutos de tomate en invernaderos utilizando Aprendizaje Profundo

Para la detección de frutos de tomate se hace mediante el uso de "Aprendizaje Profundo" utilizando *MaskRCNN*, los resultados de los experimentos demuestran que este método en conjunto con unas cámaras sencillas y económicas, funcionan bien para la detección de frutas de tomate, tanto de clases maduras como inmaduras, superando un valor de predicción del 0.91, ver en [3].

Detección y conteo de panículas de sorgo utilizando imágenes de sistemas aéreos no tripulados y Aprendizaje Profundo

Un antecedente del "Aprendizaje Profundo" en la agricultura, se encuentra la detección y conteo de partículas de sorgo, utilizando imágenes obtenidas de un sistema aéreo no tripulado, en donde se hace uso de un algoritmo CNN para integrar el "Aprendizaje Profundo", el cual se implementa usando Python con la biblioteca Tensorflow para el procedimiento de "Aprendizaje Profundo" y la biblioteca OpenCV. Este algoritmo por medio del entrenamiento con 1000 imágenes obtiene una precisión del 95.5% y un error cuadrático medio de 2.5 en la detección y conteo de partículas de sorgo, con lo cual se observa que la integración de imágenes y el modelo U-Net CNN es un método preciso y robusto, que permite hacer una estimación precisa de las partículas sorgo. Para detalles específicos verse en [4].

Localización automática y recuento de plagas de cultivos agrícolas basados en una canalización de aprendizaje profundo mejorada

Las plagas de insectos son una de las principales causas de daños en los cultivos agrícolas, por lo que se hace uso del "Aprendizaje Profundo" para localizar y hacer un recuento de las plagas agrícolas en imágenes por medio de un algoritmo CNN del modelo de Zeiler y Fergus, así como de una red propuesta de región para eliminar las detecciones superpuestas, con esto se alcanza una precisión del 0.93 con una tasa de error de 0.10, además este método tiene una mAP (mean Average Precision) de 0.885, ver [5].

Técnicas de aprendizaje profundo para la estimación del rendimiento y el tamaño de los cítricos utilizando un VANT

Los frutos cítricos son de las más importantes producciones del área mediterránea, por lo cual se buscó desarrollar un método para estimar el rendimiento de los cultivos y el tamaño de la fruta antes de la cosecha, para que con esta información se pueda ayudar a los agricultores a tomar decisiones. Para hacer la estimación con el dispositivo de IA, se tiene que entrenar un modelo de "Aprendizaje Profundo", y por medio del VANT (Vehiculo Aereo No Tripulado) se adquirieron imágenes para hacer el conjunto de datos de aprendizaje además con OpenCV se hace una máscara para delimitar la región de interés, se selecciona un modelo Faster-R-CNN, debido a que esta red puede usar varias arquitecturas tales como ResNet, Inception y Atrous, las cuales aumentan la eficiencia y precisión de la detección de la fruta.

Se utiliza el modelo pre-entrenado Faster R-CNN Inception Resnet V2 Atrous Coco con una interfaz de programación de aplicaciones (API) de detección de objetos TensorFlow. Las estimaciones de rendimiento son muy precisas, pues

los resultados del modelo están más cerca del rendimiento real que las estimaciones visuales por un técnico profesional, ver[6].

Conteo y mapeo de frutas basado en cámaras monoculares con asociación de datos semánticos

Se propone un *pipeline* de conteo de frutas que sea liviana, rápida y económica, que se basa en una cámara monocular, teniendo un rendimiento de conteo comparable con un sistema de conteo de frutas de última generación sumamente costoso. La detección de frutos se basa en *Faster R-CNN*. El *frame R-CNN* más rápido consta de dos módulos. El primer módulo es una red de propuesta de región, que detecta regiones de interés. El segundo módulo es un módulo de clasificación, que clasifica regiones individuales y retrocede el cuadro delimitador para cada fruta simultáneamente. Finalmente, el umbral de probabilidad se aplica y se lleva a cabo una supresión no máxima para eliminar detecciones duplicadas. Todos los detalles mencionados pueden encontrarse en [7].

Contar manzanas y naranjas con Aprendizaje profundo: un enfoque basado en datos

El conteo de frutas es realmente importante para que los productores estimen el rendimiento y administren sus huertos. Se aplica “Aprendizaje Profundo” usando una canalización que consiste en una forma parecida a la red totalmente convolucional (FCN) para detectar manchas, para el conteo se usa una segunda red neuronal y una regresión lineal para contar el número de frutas dentro de cada *blob* detectado por la FCN. Utilizando Caffe (entorno de

trabajo del modelo “Aprendizaje Profundo”) para este modelo se entrenó en una GPU NVIDIA TitanX, ver [8].

Detección profunda de frutos en huertos

Sirve para la detección de frutas (mangos, manzanas y almendras) a gran escala utilizando “Aprendizaje Profundo”, haciendo uso de una Faster R-CNN la cual se compone de dos módulos: un RPN (Region Proposal Network) usando para la detección de Rols en la imagen y un módulo de clasificación, el cual clasifica las regiones individuales y retrocede un cuadro delimitador alrededor del objeto. Se realiza transferencia de aprendizaje (transfer learning) y así usando las funciones de CNN de ImageNet pre-entranado se obtienen resultados de última generación en una variedad de tareas de procesamientos de imágenes, desde la clasificación hasta el subtitulado de imágenes, ver [9]

Imágenes aéreas y red neuronal convolucional para la detección de floraciones de algodón

Se entrena una red convolucional para detectar flores de algodón en imágenes sin procesar, y sus ubicaciones 3D se calculan usando una nube de puntos densa, construida a partir de las imágenes aéreas con el método de estructura de movimiento. La CNN que se utiliza para clasificar las imágenes de flores potenciales tiene siete capas, una de entrada, dos capas de convolución, dos capas de *Max-Pooling* y dos capas densas. El conjunto de datos se realiza con imágenes de floración potencial, clasificándolas en floración y no floración, con un total de 14,000 y 60,000 imágenes respectivamente, además de ser etiquetado manualmente. Se hace un entrenamiento de 30 épocas con un lote de 256, ver [10].

1.2. Definición del problema

En el estado de Durango hay muchas empresas dedicadas a la distribución de granos, las cuales cuentan con máquinas modernas capaces de introducir de manera autónoma distintos tipos de granos dentro de costales, llenándolos con una masa promedio por unidad de costal, posteriormente estos costales se ponen uno encima de otro hasta formar una estiba. Esto siempre utilizando estrategias de acomodo para evitar algún accidente al trabajador, además facilita el conteo manual de los costales por el personal encargado. Cabe mencionar que es una labor tediosa y no suele hacerse con frecuencia, ya que se complica el conteo conforme la estiba aumenta de tamaño. De esta manera no se tiene el control en tiempo real del inventario existente en el almacén, impactando a la planeación y a la logística en la distribución del producto, por esto mismo, se dificulta el cálculo en las ganancias que se tendrían al vender el producto a corto o a largo plazo.

1.3. Justificación

Debido al constante incremento en el tamaño de la estiba, el conteo manual de los costales se convierte en un verdadero reto, razón por la cual los conteos son realizados en periodos muy largos, por lo tanto, no se cuenta con la información actualizada de la cantidad de costales, ni tampoco del volumen en inventario, lo cual impacta en el control del mismo. En consecuencia, en este proyecto de investigación se pretende desarrollar un sistema inteligente de detección y conteo de costales, que pueda realizar los conteos en tiempo real para saber en todo momento el inventario en volumen. Este sistema considera el diseño y desarrollo de un dispositivo electrónico conectado a una

computadora y a una cámara para que detecte y cuente los costales de la estiba, calculando el volumen total y la masa existente. Es importante señalar que dicha información se guarda en una hoja de cálculo de Excel.

1.4. Hipótesis

Es posible diseñar y desarrollar un dispositivo que por medio de inteligencia artificial pueda detectar y contar los costales que conforman una estiba para calcular el volumen total y la masa existente.

1.5. Objetivos

1.5.1. Objetivo General

Desarrollar un prototipo de escaneo para el cálculo estimado del volumen y la masa de una estiba de grano por medio del conteo de costales que la conforman, con la finalidad de llevar el control eficiente en el inventario de un almacén.

1.5.2. Objetivos específicos

- Elaborar el *software* que permita la detección y conteo de los costales.
- Contar el contenido de cada dimensión de la estiba, para posteriormente calcular el volumen y masa.
- Seleccionar la mejor minicomputadora para esta aplicación.

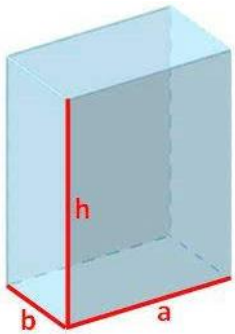
- Diseñar la carcasa del dispositivo.
- Elaborar un prototipo integral del dispositivo y ejecución de pruebas experimentales.

CAPÍTULO II
MARCO TEÓRICO

2.1. Estiba

Es una forma de distribuir y colocar de forma adecuada un lote determinado en un almacén o bodega, en este caso se trata de estibas conformadas por costales, éstas van formando un paralelepípedo, lo cual permite sacar su volumen basándose en la fórmula:

Tabla 1. Fórmula para cálculo de volumen

Fórmula de volumen:	**Donde:**	
$\mathbf{V = A * h}$	$\mathbf{V = Volumen}$ $\mathbf{A = Área\ de\ la\ base}$ $\mathbf{h = Altura}$	
Es decir:	$\mathbf{a = Largo}$ $\mathbf{b = Ancho}$	
$\mathbf{V = a * b * h}$		

A continuación, se muestran imágenes de estibas en construcción de un almacén de granos en el municipio de Villa Unión, Dgo., el cual se tomó de base para llevar a cabo esta investigación. Figura 1.

Figura 1. Imágenes de estibas en formación

2.2 Inteligencia artificial (IA)

La inteligencia artificial consiste en los intentos de replicar la inteligencia humana en sistemas artificiales, dentro de ella se tiene el "Aprendizaje Automático" y en ésta última está el "Aprendizaje Profundo". Más detalladamente, la IA es la capacidad de las máquinas para usar algoritmos, aprender de los datos y utilizar todo lo aprendido en la toma de decisiones, tal y como lo haría un ser humano, pero a diferencia de los humanos, los dispositivos basados en IA no tienen necesidad de descansar y pueden analizar grandes volúmenes de información, además, la proporción de errores utilizando las máquinas es mucho menor que en los humanos en la ejecución de dichas tareas, ver [11].

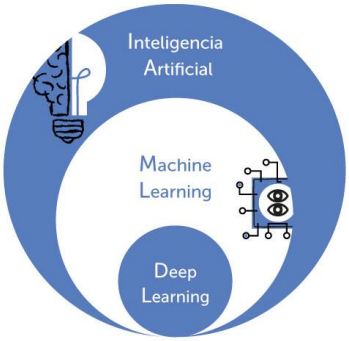


Figura 2. Inteligencia Artificial

Fuente: https://talentocorporativo.com/diferencias-entre-machine-learning-inteligencia-artificial-y-robotica/

2.2.1. Aprendizaje Automático (Machine Learning con las siglas ML)

Son las técnicas en las que el mismo sistema aprende a encontrar una respuesta sin que alguien lo esté programando; el "Aprendizaje Automático" usa algoritmos para aprender patrones de datos para utilizar el conocimiento adquirido en la toma de decisiones.

2.2.2. Aprendizaje Profundo (Deep Learning con las siglas DL)

Es un subcampo del "Aprendizaje Automático", que se usa para resolver problemas muy complejos y que normalmente implican grandes cantidades de datos. Se produce mediante la utilización de redes neuronales, que se organizan en capas para reconocer relaciones y patrones complejos en los datos, se llama "Aprendizaje Profundo", porque a mayor cantidad de capas conectadas entre sí, se obtiene un aprendizaje más fino.

Su aplicación requiere un enorme conjunto de información y una potente capacidad de procesamiento. Actualmente se usa en el reconocimiento de voz, el procesamiento del lenguaje natural, la visión artificial y la identificación de vehículos en los sistemas de asistencia al conductor, ver [12].

Existen dos problemas en el "Aprendizaje Profundo" los cuales son:

- **Overfiting**: Es cuando el algoritmo "memoriza" los datos y la red neuronal no sabe resolver problemas.
- **Caja negra:** En este caso se conocen las entradas a la red neuronal, sin embargo, no se conoce lo que pasa dentro de las capas intermedias de la red.

2.2.3. Red neuronal

La neurona funciona de la siguiente forma: se tienen las entradas de los datos y los pesos de la neurona, por dentro hace sumas ponderadas, pasa por una función de activación y entrega un resultado, en cada iteración (época) se cambian los pesos para resolver un problema.

Una red neuronal está conformada por capas y cada capa pasa información a la siguiente, y cada una avanza con respecto a la información de la capa anterior, es decir, entre más avance la información va a ser más específica.

2.2.4. Arquitectura de la red neuronal

La arquitectura de la red neuronal se describe de la siguiente manera:

1. Comienza con el almacenamiento de los datos dentro de las capas de entrada.

2. La salida de cada capa de entrada es a su vez la entrada de todas las capas ocultas, que son las encargadas de procesar toda la información y de realizar las operaciones matemáticas.
3. Por último, la capa de salida es la que realiza las predicciones.

Las capas de la red neuronal se visualizan en detalle en la figura 3.

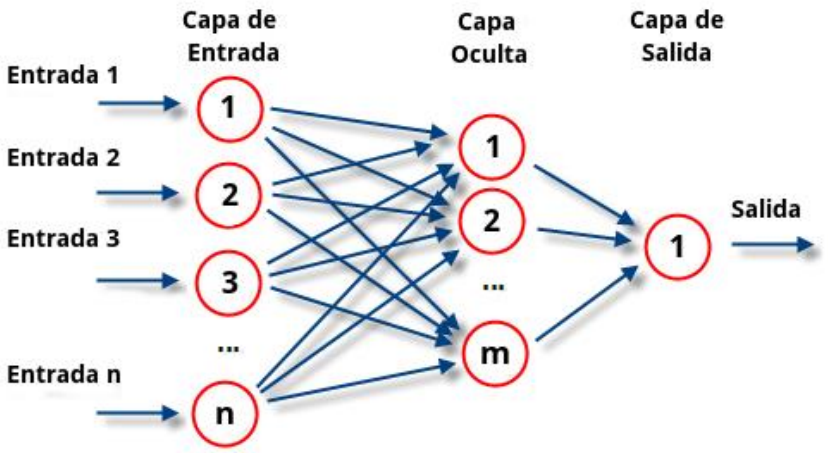


Figura 3. Capas de una red neuronal

Fuente: https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/

Prácticamente la red neuronal se encarga de realizar la predicción, ya sea de una regresión o de una clasificación. Para saber si el algoritmo está prediciendo de manera correcta, la red neuronal toma los valores reales y los compara con la predicción, de esta manera se obtiene un *score* o error entre lo real y lo que se predijo, por lo que a este score se le denomina como: la función de pérdida, la cual es un indicativo de que tan buena puede llegar a ser la predicción de la red. Por lo que tener valores pequeños en la función de pérdida, significa que

la predicción es muy buena y lo contrario sucede al tener valores altos lo cual será indicativo de una mala predicción. Para disminuir el error de la función de pérdida, el descenso del gradiente o un optimizador actualiza los pesos en la red, posteriormente, se hace una nueva predicción, comparándola nuevamente con el valor real para generar así una nueva función de pérdida, posteriormente se aplica el gradiente de forma recursiva para ir disminuyendo el error y encontrar el mínimo global y obtener así una buena predicción.

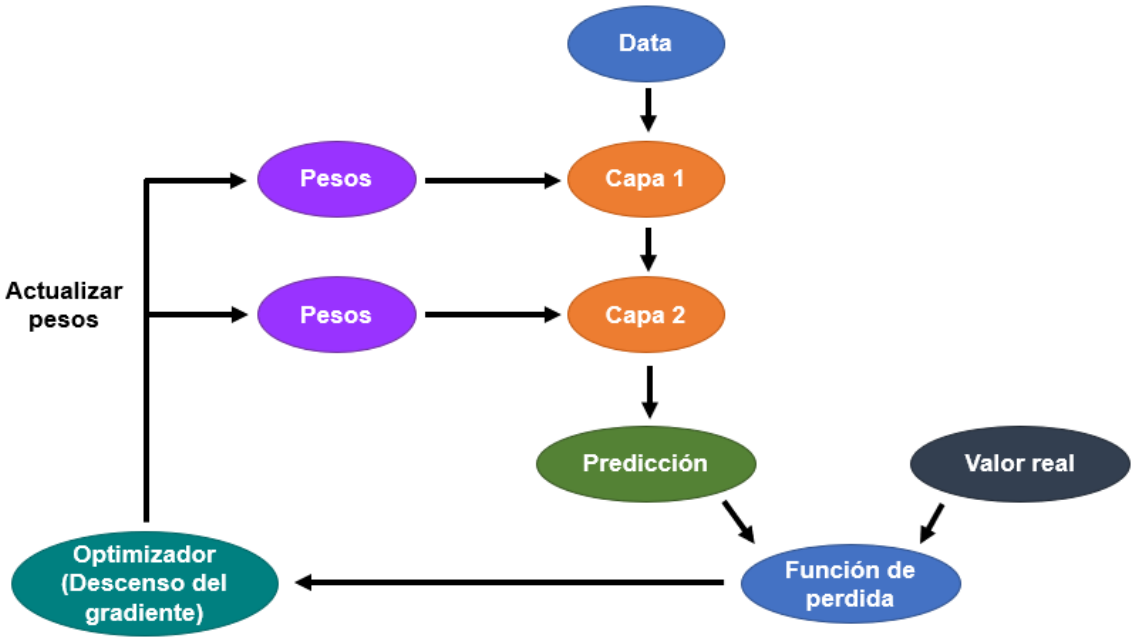


Figura 4. Arquitectura de una Red Neuronal

2.2.5. Red Neuronal Convolucional (Convolutional Neural Network con las siglas CNN)

Se tienen imágenes de entrada, esas imágenes van a pasar por una CNN, la cual va extraer características que al principio serán superficiales, sin embargo, cuando se va adentrando a lo profundo de la red van a ir siendo características

mucho más complejas, estas capas de convolución van atadas a una capa de *Max-Pooling*, la cual sirve para reducir la complejidad acortando el ancho y largo de las imágenes, y así se va capa por capa (convolución por convolución), dando cada vez una imagen más pequeña, pero más profunda en su contexto, después se aplica un capa que hace de una sola dimensión al tensor resultante, es decir, lo convierte en un solo vector, esta capa se llama *Flatten*, posteriormente al tener ese vector se puede hacer la clasificación utilizando capas densas, como se ve en la figura 5, también se le puede agregar una capa que apaga ciertas neuronas en la red de forma aleatoria en cada iteración para que así no dependan tanto una de la otra para evitar el *sobreajuste* (overfitting), a esto se le llama dilución u abandono (Dropout).

En cada convolución se va a definir un tamaño de núcleo (kernel), el cual es una matriz que se desliza desde la esquina superior izquierda a la esquina inferior derecha paso por paso, hasta que termina la imagen haciendo la operación matemática de convolución. También se determina, si poner o no relleno (*padding)*, el cual es un marco de ceros alrededor de la imagen para que cuando se aplique el núcleo con la convolución, no se disminuya el tamaño de la imagen y se mantengan los detalles, para poner relleno se debe colocar padding = [“same”], en caso de no querer aplicarlo se coloca la palabra padding = [“valid”].

En la figura 5 se puede ver como entra una imagen de tamaño 28x28 y con un canal (ya que es una imagen en blanco y negro, si fuera a color tendría 3 canales) al modelo, se aplica una convolución con un núcleo de 5x5 sin relleno (padding), por lo que disminuye un poco de tamaño la imagen, ahora de 24x24. Luego se aplica una capa *Max-Pooling* con la cual se extraen las características representativas de la imagen y se disminuye el tamaño de la imagen (alto y ancho), quedando de 12x12.

Nuevamente se hace una convolución con un núcleo de 5x5 sin relleno haciendo que quede una imagen de 8x8, posteriormente se aplica otro *Max-Pooling* con el cual crece en profundidad la salida de la capa anterior, pero disminuye en tamaño a 4x4. Después se aplica la capa *Flatten* para que el tensor quede en una sola dimensión y finalmente aplicar una red *Fully-Connected* con capas densas y con función de activación ReLU (la cual es excelente para las capas ocultas de la red neuronal), y una capa densa final con la cual se hará la predicción. Con la dilución en cada iteración ciertas neuronas de forma aleatoria se apagarán para que no dependan tanto entre sí y se reduzca el sobreajuste.

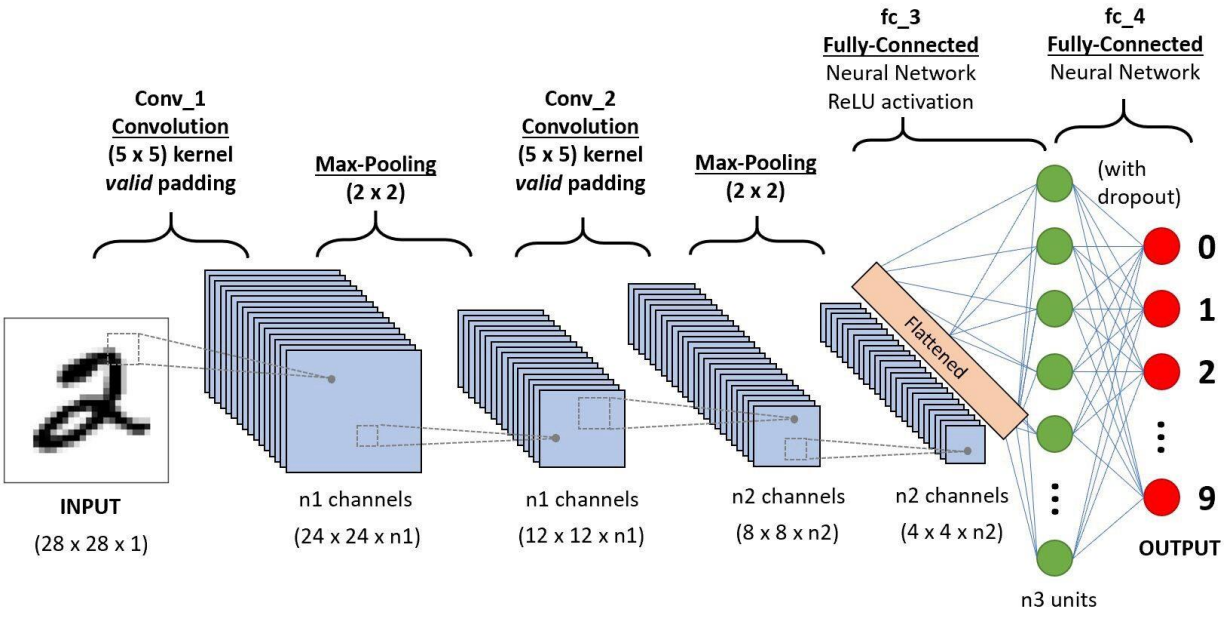


Figura 5. Funcionamiento de Red Neuronal Convolucional

Fuente: https://datapeaker.com/big-data/arquitectura-de-red-neuronal-convolucional-arquitectura-cnn/

2.3. Software para aplicaciones de IA

2.3.1. Ubuntu

Ubuntu es un sistema operativo basado en Linux que es muy popular entre los desarrolladores, y es preferido para muchas tecnologías emergentes entre las cuales se pueden destacar “Inteligencia Artificial”, “Aprendizaje Automático” y el “Aprendizaje Profundo”, ningún otro sistema operativo se puede acercar a las librerías, tutoriales y ejemplos de Ubuntu, además tiene un excelente nivel de soporte para las plataformas de código abierto y *software* más actual. Todo esto hace que sea el sistema operativo seleccionado por los entornos de trabajo más importantes para IA tales como OpenCV, TensorFlow, Theano, Keras y PyTorch, ver [13]

Debido al impacto de las GPU en la IA y Nvidia, se está invirtiendo en CUDA para Linux, desplegando así la potencia de sus tarjetas gráficas en la computación general. De hecho, el sistema operativo que maneja la Jetson Nano que se planea utilizar en este proyecto, viene con el sistema operativo Ubuntu 18.04.

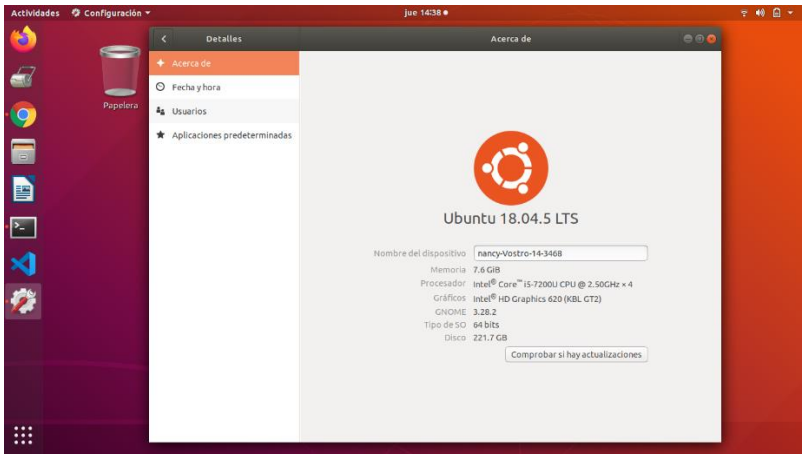


Figura 6. Interfaz de Ubuntu

2.3.2 Python

Es un lenguaje de programación de alto nivel, por lo que se puede utilizar para situaciones complejas, además es dinámico e interpretado, es decir, no es necesario compilarlo para ejecutar las aplicaciones escritas en Python. Está disponible para todos los sistemas operativos y presenta oferta de código abierto.

Actualmente es un lenguaje muy popular a nivel mundial, asimismo es una de las mejores opciones para el desarrollo de IA, ya que Python tiene bibliotecas como Numpy, Scipy y Pybrain, que se usan para la computación científica, computación avanzada y el "Aprendizaje Automático", hoy en día Python en conjunto con NumPy, scikit-learn, iPython Notebook y Matplotlib, son la base para iniciar un proyecto de IA. Un ejemplo de importantes librerías y su uso en IA son:

- NumPy: Es usada para contener datos genéricos que comprenden un objeto de matriz N-dimensional.
- Pandas: Es una biblioteca de código abierto que suministra a los usuarios estructuras de datos y herramientas analíticas que son sencillas para utilizar con Python.
- Matplotlib: Es una biblioteca de trazado 2D que se puede usar fácilmente con Python.

Debido a la gran variedad de librerías que son base de la IA se pueden usar de manera simple con Python, así como el apoyo de la comunidad y que son de código abierto, hacen a este lenguaje uno de los mejores para trabajar con inteligencia artificial, ver [14].

2.3.3. OpenCV

OpenCV significa Visión Artificial Abierta (Open Computer Vision) y es una biblioteca libre de visión artificial desarrollada por Intel, que abarca más de 2,500 algoritmos que permiten cosas como:

- Identificar objetos y rostros.
- Encontrar imágenes similares.
- Eliminar ojos rojos de las fotografías.
- Detectar determinados colores en imágenes o videos.
- Segmentar videos en imágenes y aplicar filtros.
- Identificar objetos en movimiento en un video.
- Clasificar acciones humanas que estén en videos.
- Extraer modelos 3D.
- Utilizar en campos como la robótica y la realidad aumentada.

Es multiplataforma, sirve de proveedor de infraestructura para aplicaciones de visión artificial, puede usarse con varios lenguajes de programación (entre esos Python), tiene una gran eficiencia y es gratuita, ver [15].

2.3.4. PyTorch y Tensorflow

PyTorch es un entorno de trabajo de “Aprendizaje Profundo” que fue desarrollado por Facebook AI Research, PyTorch está diseñado para funcionar perfectamente con Python y con NumPy, su sintaxis y aplicación es parecida a Python. Proporciona una interfaz sencilla de Python y da una API potente y simple, además de que se puede usar en Windows y Linux, PyTorch puede distribuir el trabajo computacional entre varios núcleos de CPU y GPU. Emplea computación dinámica lo que da flexibilidad al momento de crear redes

complejas, al crear aplicaciones de “Aprendizaje Profundo” basadas en gráficos dinámicos se pueden manipular en tiempo real. Los modelos basados en PyTorch pueden acceder de forma directa a las plataformas y tiempos de ejecución compatibles con ONNX. PyTorch usa los datos en tensores que almacenan la red neuronal, representaciones de capas ocultas y salidas, estos tensores son análogos a las matrices de *n* dimensiones en NumPy y pueden usar una GPU. A diferencia de TensorFlow, PyTorch permite a los usuarios acceder y mirar la GPU o la CPU en cualquier nivel de computación. [16]

Por otro lado, TensorFlow es un entorno de trabajo desarrollado por Google, que es especialmente para el “Aprendizaje Profundo” y las redes neutrales, el cual facilita armar un modelo de red y ejecutar el entrenamiento. Además, proporciona una arquitectura flexible que facilita implementar cálculos numéricos en una o más CPU o GPU con solamente una API. Tiene la ventaja de que es altamente escalable para el cálculo en diferentes sistemas y grandes conjuntos de datos, viene con TensorBoard para el análisis de modelos desarrollados y se puede ejecutar en Windows, Linux, Mac y Android, ver [17].

2.3.5 YOLO (You Only Look Once)

YOLO es una red neuronal convolucional que usando características de toda la imagen predice múltiples cuadros delimitadores y a la vez predice todos esos cuadros en todas las clases para una imagen de forma simultánea. YOLO divide la imagen de entrada en una cuadricula S × S y si el centro de un objeto está ubicado en una celda de la cuadricula, esa celda será la encargada de detectar ese objeto, YOLO se entrena con imágenes completas y optimiza directamente el rendimiento de la detección, además permite procesar video en tiempo real

con menos de 25 milisegundos de latencia. Existen 3 principales implementaciones de YOLO:

- Darknet: https://pjreddie.com/darknet/
- AlexeyAB/darknet: https://github.com/AlexeyAB/darknet
- Darkflow: https://github.com/thtrieu/darkflow/

Darknet es todo entorno de trabajo para redes neuronales por lo que se puede utilizar para otros objetivos además de YOLO, está escrito en C con CUDA, debido a esto permite computo con GPU, además es la implementación oficial de YOLO.

2.4. Kit de desarrollo de IA

2.4.1. Jetson Nano 2GB

El Kit para Desarrolladores Jetson Nano consiste en una computadora pequeña y poderosa, que permite ejecutar múltiples redes neuronales en paralelo para aplicaciones como: clasificación de imágenes, detección de objetos, segmentación y procesamiento de voz. Esta pequeña computadora tiene 472 GFLOPS de rendimiento informático, con una CPU ARM de cuatro núcleos y 64 bits y una GPU NVIDIA integrada de 128 núcleos. Existen 2 presentaciones, una de 4GB y otra de 2GB de RAM, en este proyecto se utiliza la segunda, además es de bajo consumo, pues tiene una entrada de 5V, ver [18].

Figura 7. Jetson Nano

Fuente: *https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/*

Asimismo, el JetPack 4.6.1 SDK provee un entorno de escritorio Linux completo para Jetson Nano basado en Ubuntu 18.04 con gráficos acelerados, soporte para NVIDIA CUDA Toolkit 10.0 y bibliotecas como cuDNN 7.3 y TensorRT 5. El SDK también incluye la capacidad de instalar de forma nativa bibliotecas populares de “Aprendizaje Automático” de código abierto como: TensorFlow, PyTorch, Caffe, Keras y MXNet, además de otras bibliotecas de visión por computadora y el desarrollo de robótica como OpenCV y ROS, ver [19].

Figura 8. Interfaz de Jetson Nano

2.4.2. NUC (Siguiente Unidad de Computación - Next Unit of Computing)

Es un tipo de PC de pequeñas dimensiones de Intel, que contiene un sistema completo metido en su minúsculo chasis. Su nombre NUC hace referencia al formato reducido donde van insertados los componentes y sirve para tareas ofimáticas, multimedia y prácticamente todo lo que puede hacer una computadora de escritorio. NUC solo viene con el estuche, la placa base y la CPU. Todo lo demás debe comprarse por separado y luego instalarlo el usuario, lo cual permite personalizarlo como se desee, ver [20].

Figura 9.NUC

CAPÍTULO III
DESARROLLO Y RESULTADOS

3.1. Metodología

A continuación, se describen los pasos a seguir en el desarrollo de este proyecto, cabe mencionar que durante el proceso se fueron haciendo modificaciones según se fueron requiriendo.

3.1.1. Selección de herramientas y materiales inicial

Investigar diferentes herramientas y materiales que permitan llevar a cabo el desarrollo del proyecto, según los resultados de las pruebas es probable que se tengan que hacer cambios en los materiales, sin embargo, en primera instancia se llevó a cabo la siguiente selección:

3.1.1.1. Hardware

- Jetson Nano 2GB
- Cámara Web Logitech C270 USB con Micrófono 3 Mpx 720p HD
- SanDisk RAM-3077 - Tarjeta microSDXC UHS-I 128GB, 160Mb/S 4K Clase A2 V30 C/Adaptador
- Ventilador
- Cargador con salida de 5V 3A
- Jumpers
- Botón
- Protoboard

- Impresora 3D de filamento
- Filamento PLA

3.1.2. Software

- Ubuntu 18.04
- Visual Studio Code
- Python
- Open CV

3.1.3. Hacer un conjunto de datos con imágenes de entrenamiento, validación y test.

Para hacer el conjunto de entrenamiento (train), validación (valid) y evaluación (test) se tomaron de primera instancia 399 fotografías de costales en diferentes condiciones de posición, lugar y luz, estas fotos se guardaron en una carpeta a la cual se llamó *data*, posteriormente con la aplicación LabelImg, la cual permite etiquetar y marcar en un recuadro lo que se desea detectar de cada imagen, se selecciona cada costal de cada una de las imágenes que se tienen en la carpeta y se les coloca la etiqueta de costal. Finalmente se utilizó un conjunto de datos de 641 imágenes para mejorar el aprendizaje, el cual se dividió en 525 imágenes de entrenamiento, 59 de validación y 58 para evaluación.

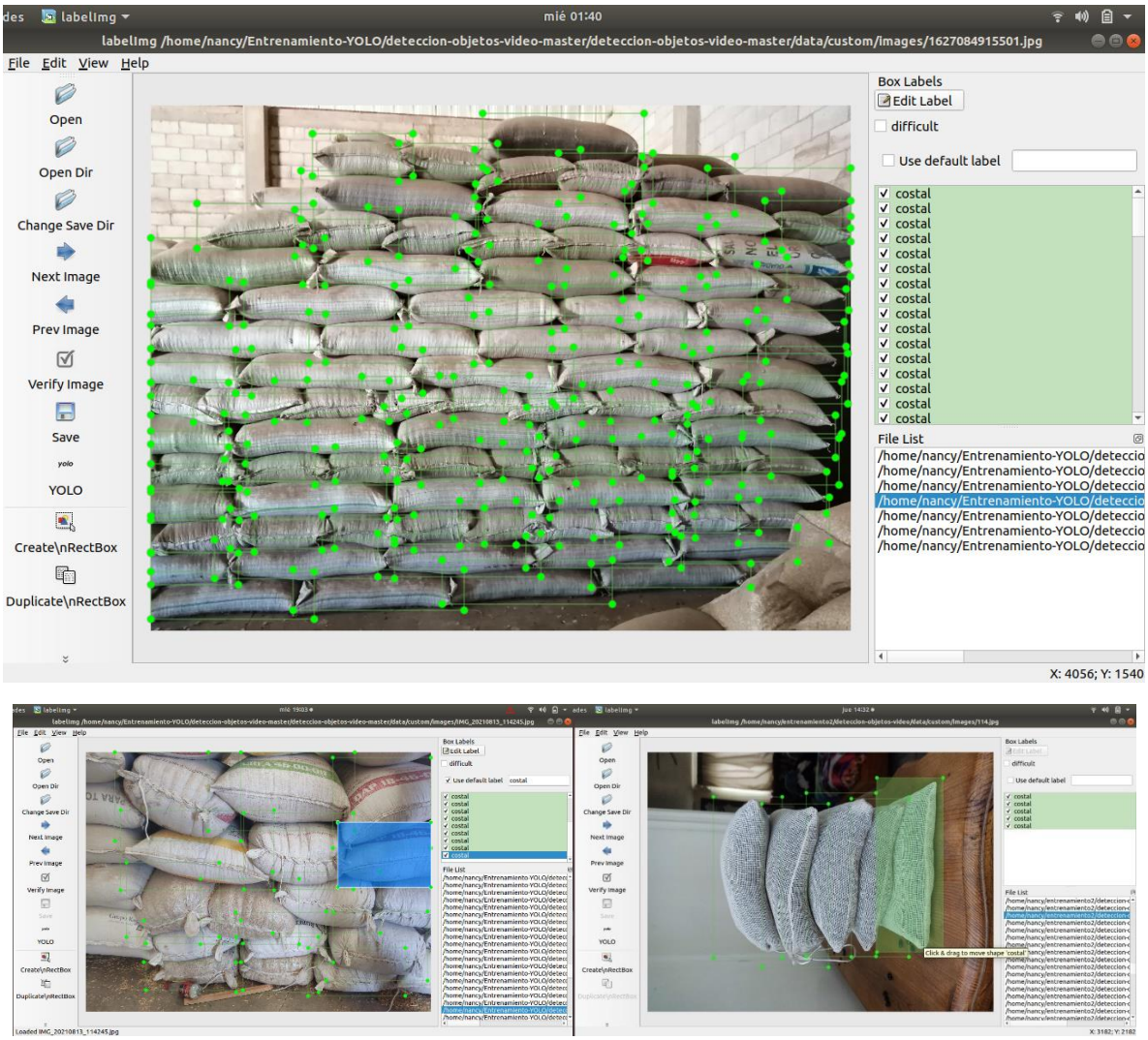


Figura 10. Etiquetado de imágenes de costales

Debido a que el almacén de granos en el cual se inspiró este proyecto se encuentra en el municipio de Villa Unión, Dgo., se optó para aumentar el conjunto de datos y realizar las pruebas haciendo uso de pequeños costales de frijol.

Figura 11. Costalitos pequeños para pruebas

3.1.4. Obtener una red neuronal convolucional

Para la red neuronal convolucional se utilizó principalmente Python en conjunto con PyTorch y YOLO v3. Con PyTorch se realizó la arquitectura de la red neuronal junto con YOLOv3, este último hace transferencia de aprendizaje, es decir, se pasan los pesos de YOLO, los cuales son pesos de capas ocultas, mismas que ya saben detectar ciertos elementos o patrones en las imágenes, es decir, al final solo se necesita entrenar la última capa.

El *script* de *detección_video.py* es el que marca con un rectángulo los costales detectados, asimismo hace el conteo, cálculo de masa según los costales contados, asimismo genera el archivo Excel con los valores de la masa, volumen y la fecha en que se realizó esa detección. También en este archivo es donde por medio de OpenCV se segmenta el video, ya sea de un archivo o en vivo, para realizar la detección con el modelo en cada imagen que conforma el video, también al final de cada detección genera un archivo de salida llamado *output.mp4* el cual se almacena en la carpeta *output* y es un video donde se ve la detección que se acaba de realizar.

El archivo *test.py* es el script que permitirá evaluar el modelo entrenado y calcular su *mAP* y f1 para saber si funciona la detección de costales. El conjunto de validación que esta guardado en *valid.txt* sirve para validar el modelo durante el entrenamiento y este es usado por el archivo de *train.py*. En el archivo de *train.py* se configura el entrenamiento, es decir, la cantidad de épocas, tamaño de lote (batch_size), configuración de datos con los que se trabajará (*train*, *valid* y *classes.name*), los puntos de control para que se generen y almacenen, el optimizador a usar (el cual fue el optimizador Adam) y métricas de cada capa de YOLO las cuales aparecen durante el entrenamiento.

En la carpeta *config* están los parámetros y configuraciones de YOLO, así como del entrenamiento personalizado. El punto de control (checkpoint) funciona para guardar los pesos del modelo de la época con mejor desempeño en un archivo, en este caso en la carpeta *checkpoint* se guarda cada punto de control de cada época de entrenamiento, con el que funciona el modelo es el punto de control final, pues es donde está la predicción que permitirá clasificar el objeto como un costal.

En la carpeta de *utils* se encuentran archivos tales como:

- ***augmentations.py***: Genera una función *horizontal_flip* de *data augmentation* y con esto se aumentará el conjunto de datos y mejorará el entrenamiento.
- ***datasets.py***: Se carga el conjunto de datos conformado por todas las imágenes y etiquetas de las carpetas *images* y *labels*, cambia el tamaño de las imágenes a uno determinado, se extraen las imágenes como un tensor *PyTorch* y aplica *data augmentation*.

- ***parse_config.py***: Analiza el archivo de configuración de capa *YOLOv3* y devuelve definiciones de módulo, asimismo examina el archivo de configuración de datos.

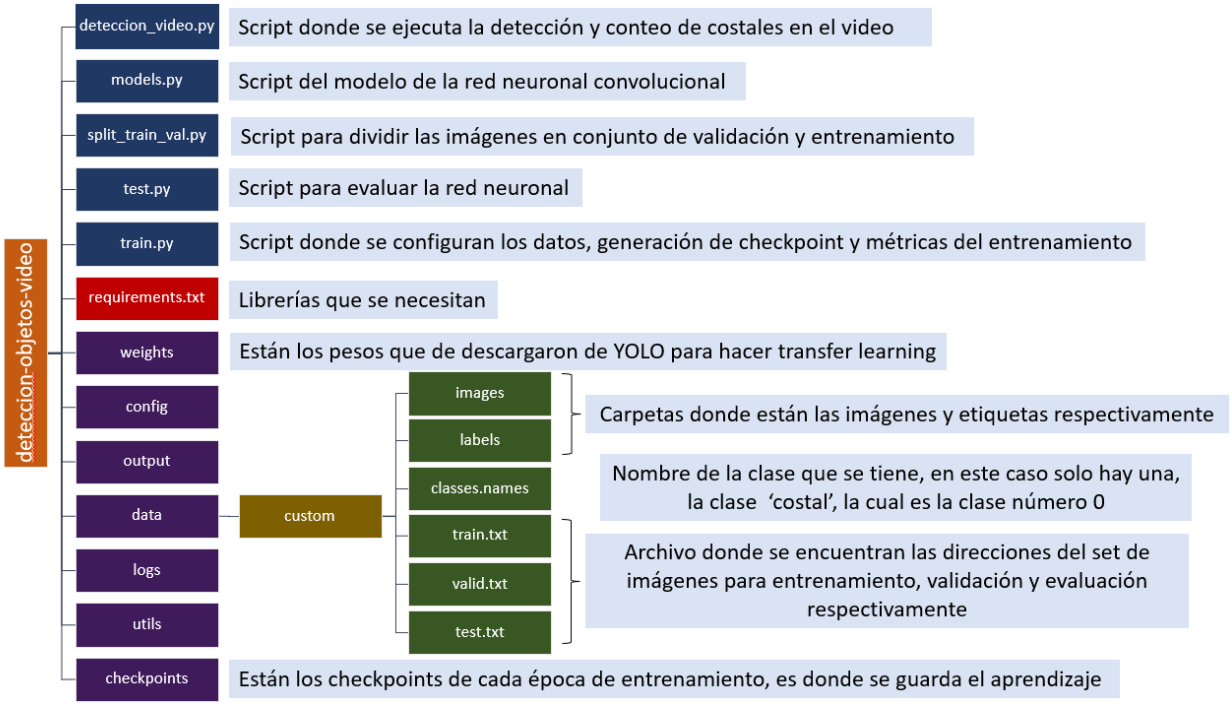


Figura 12. Estructura general del modelo de CNN de detección de costales

3.1.5. Entrenar el modelo.

Se realizó el entrenamiento de este modelo en una computadora ASUS con GPU de 32GB de RAM y disco de estado sólido, con sistema operativo Ubuntu 18.04.4 LTS.

Figura 13. Características de computadora de entrenamiento de CNN

Para cada entrenamiento se definió la cantidad de épocas y tamaño de lote, es decir, la cantidad de iteraciones que hará el modelo durante el entrenamiento y el tamaño del lote de elementos que conformarán cada época, de acuerdo a la cantidad de épocas y tamaño del lote se van a dividir las imágenes.

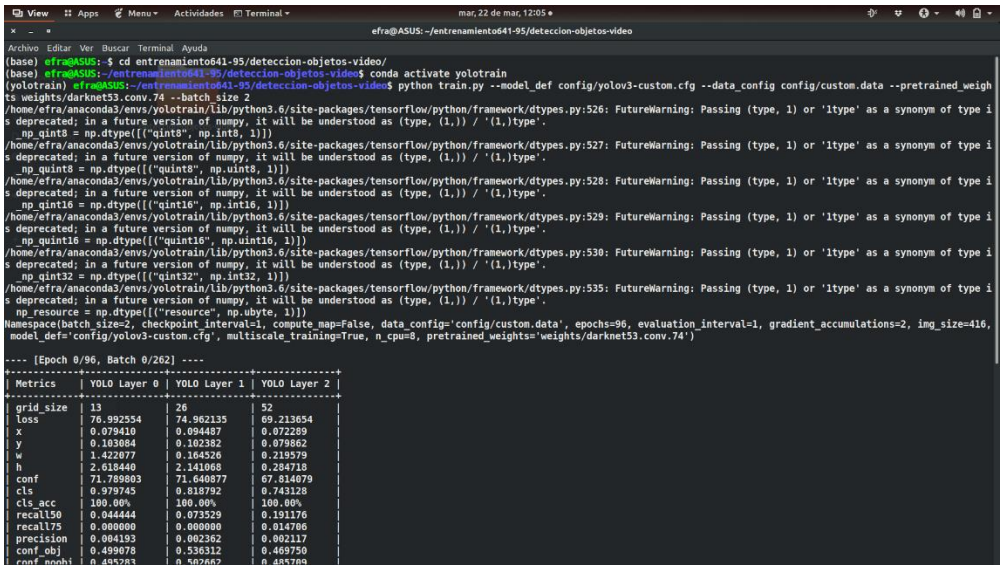


Figura 14. Ejecutar entrenamiento

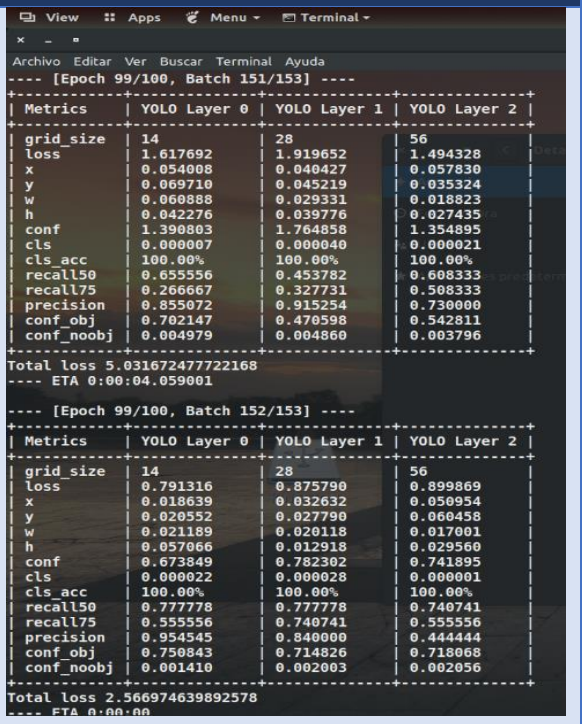
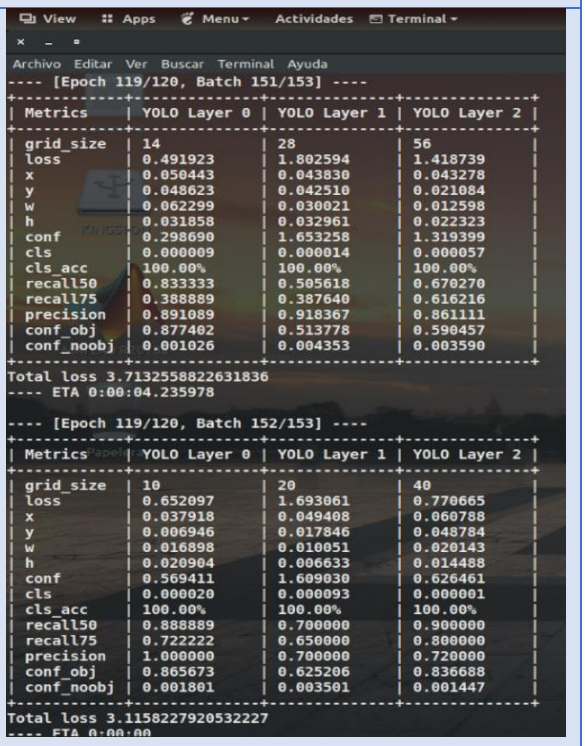
El resultado de cada entrenamiento, es decir, cada punto de control de cada época, se guardó en la carpeta de *cheackpoint*.

Figura 15. Punto de control de entrenamiento de 120 épocas

3.1.6. Entrenamientos y características

Se realizaron diversos entrenamientos para probar el funcionamiento del modelo y escogió el que tuvo mejores resultados, ya que dependiendo de la cantidad de épocas puede aumentar o disminuir las métricas de precisión, a continuación, se mencionan y comparan los más relevantes:

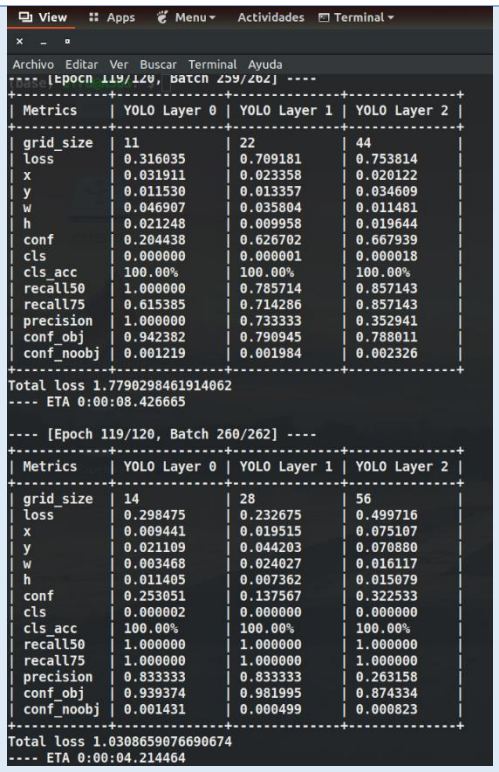
Tabla 2. Entrenamientos y sus características

No. de entrenamiento y características	Imagen de entrenamiento
Entrenamiento 1 • 100 épocas(iteraciones) • Tamaño de lote de 2 • Conjunto de datos dividido en: o 306 imágenes para entrenamiento (train) o 35 imágenes para validación (valid) o 58 imágenes para evaluación (test) • **Función de pérdida:** Valor final de 2.5669	
Entrenamiento 2 • 120 épocas (iteraciones) • Tamaño de lote de 2 • Conjunto de datos dividido en: o 306 imágenes para entrenamiento (train) o 35 imágenes para validación (valid) o 58 imágenes para evaluación (test) • **Función de pérdida:** Valor final de 3.1158	

Entrenamiento 3

- 120 épocas (iteraciones)
- Tamaño de lote de 2
- Conjunto de datos dividido en:
  - 524 imágenes para entrenamiento (train)
  - 59 imágenes para validación (valid)
  - 58 imágenes para evaluación (test)
- **Función de pérdida:**

  Valor final de 1.0308



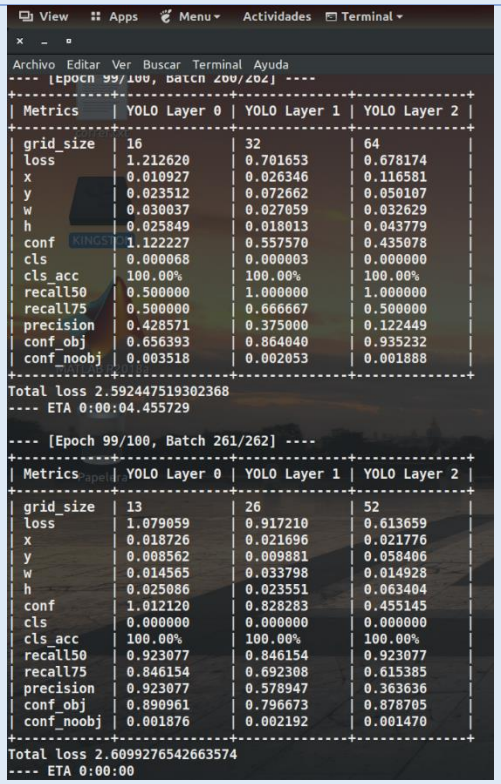
```
View  Apps  Menu  Actividades  Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
---- [Epoch 119/120, Batch 259/262] ----
+------------+--------------+--------------+--------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+--------------+--------------+--------------+
| grid_size  | 11           | 22           | 44           |
| loss       | 0.316035     | 0.709181     | 0.753814     |
| x          | 0.031911     | 0.023358     | 0.020122     |
| y          | 0.011530     | 0.013357     | 0.034609     |
| w          | 0.046907     | 0.035804     | 0.011481     |
| h          | 0.021248     | 0.009958     | 0.019644     |
| conf       | 0.204438     | 0.626702     | 0.667939     |
| cls        | 0.000000     | 0.000001     | 0.000018     |
| cls_acc    | 100.00%      | 100.00%      | 100.00%      |
| recall50   | 1.000000     | 0.785714     | 0.857143     |
| recall75   | 0.615385     | 0.714286     | 0.857143     |
| precision  | 1.000000     | 0.733333     | 0.352941     |
| conf_obj   | 0.942382     | 0.790945     | 0.788011     |
| conf_noobj | 0.001219     | 0.001984     | 0.002326     |
+------------+--------------+--------------+--------------+
Total loss 1.7790298461914062
---- ETA 0:00:08.426665

---- [Epoch 119/120, Batch 260/262] ----
+------------+--------------+--------------+--------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+--------------+--------------+--------------+
| grid_size  | 14           | 28           | 56           |
| loss       | 0.298475     | 0.232675     | 0.499716     |
| x          | 0.009441     | 0.019515     | 0.075107     |
| y          | 0.021109     | 0.044203     | 0.070880     |
| w          | 0.003468     | 0.024027     | 0.016117     |
| h          | 0.011405     | 0.007362     | 0.015079     |
| conf       | 0.253051     | 0.137567     | 0.322533     |
| cls        | 0.000002     | 0.000000     | 0.000000     |
| cls_acc    | 100.00%      | 100.00%      | 100.00%      |
| recall50   | 1.000000     | 1.000000     | 1.000000     |
| recall75   | 1.000000     | 1.000000     | 1.000000     |
| precision  | 0.833333     | 0.833333     | 0.263158     |
| conf_obj   | 0.939374     | 0.981995     | 0.874334     |
| conf_noobj | 0.001431     | 0.000499     | 0.000823     |
+------------+--------------+--------------+--------------+
Total loss 1.0308659076690674
---- ETA 0:00:04.214464
```

Entrenamiento 4

- 100 épocas (iteraciones)
- Tamaño de lote de 2
- Conjunto de datos dividido en:
  - 524 imágenes para entrenamiento (train)
  - 59 imágenes para validación (valid)
  - 58 imágenes para evaluación (test)
- **Función de pérdida:**

  Valor final de 2.6099



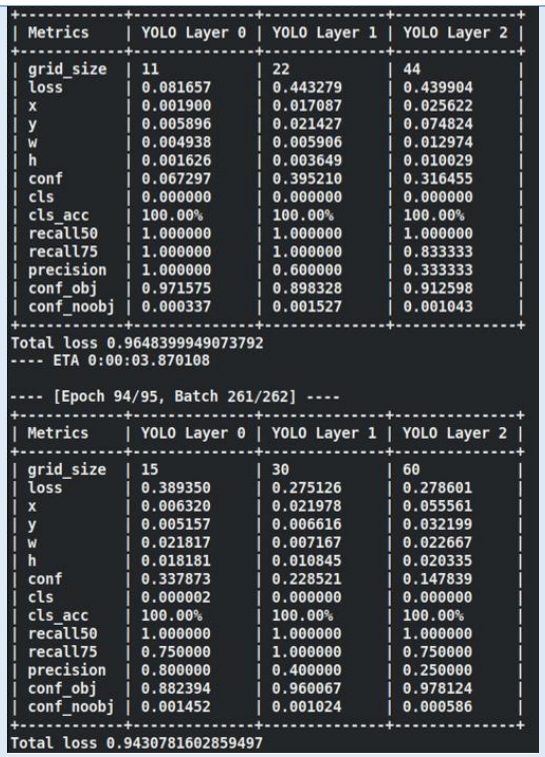
```
View  Apps  Menu  Actividades  Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
---- [Epoch 99/100, Batch 260/262] ----
+------------+--------------+--------------+--------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+--------------+--------------+--------------+
| grid_size  | 16           | 32           | 64           |
| loss       | 1.212620     | 0.701653     | 0.678174     |
| x          | 0.010927     | 0.026346     | 0.116581     |
| y          | 0.023512     | 0.072662     | 0.050107     |
| w          | 0.030037     | 0.027059     | 0.032629     |
| h          | 0.025849     | 0.018013     | 0.043779     |
| conf       | 1.122227     | 0.557570     | 0.435078     |
| cls        | 0.000068     | 0.000003     | 0.000000     |
| cls_acc    | 100.00%      | 100.00%      | 100.00%      |
| recall50   | 0.500000     | 1.000000     | 1.000000     |
| recall75   | 0.500000     | 0.666667     | 0.500000     |
| precision  | 0.428571     | 0.375000     | 0.122449     |
| conf_obj   | 0.656393     | 0.864040     | 0.935232     |
| conf_noobj | 0.003518     | 0.002053     | 0.001888     |
+------------+--------------+--------------+--------------+
Total loss 2.592447519302368
---- ETA 0:00:04.455729

---- [Epoch 99/100, Batch 261/262] ----
+------------+--------------+--------------+--------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+--------------+--------------+--------------+
| grid_size  | 13           | 26           | 52           |
| loss       | 1.079059     | 0.917210     | 0.613659     |
| x          | 0.018726     | 0.021696     | 0.021776     |
| y          | 0.008562     | 0.009881     | 0.058406     |
| w          | 0.014565     | 0.033798     | 0.014928     |
| h          | 0.025086     | 0.023551     | 0.063404     |
| conf       | 1.012120     | 0.828283     | 0.455145     |
| cls        | 0.000000     | 0.000000     | 0.000000     |
| cls_acc    | 100.00%      | 100.00%      | 100.00%      |
| recall50   | 0.923077     | 0.846154     | 0.923077     |
| recall75   | 0.846154     | 0.692308     | 0.615385     |
| precision  | 0.923077     | 0.578947     | 0.363636     |
| conf_obj   | 0.890961     | 0.796673     | 0.878705     |
| conf_noobj | 0.001876     | 0.002192     | 0.001470     |
+------------+--------------+--------------+--------------+
Total loss 2.6099276542663574
---- ETA 0:00:00
```

Entrenamiento 5

- 95 épocas (iteraciones)
- Tamaño de lote de 2
- Conjunto de datos dividido en:
  - 524 imágenes para entrenamiento (train)
  - 59 imágenes para validación (valid)
  - 58 imágenes para evaluación (test)
- **Función de pérdida:**

  Valor final de 0.9430



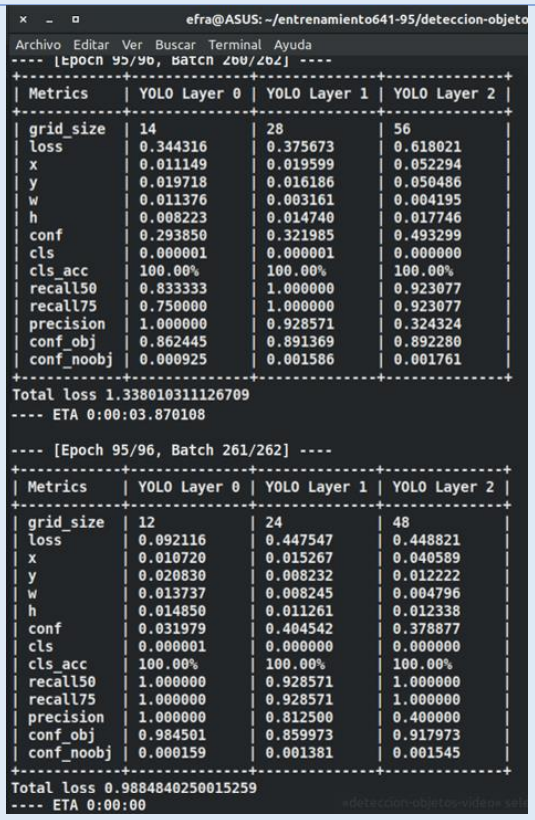
```
+------------+-------------+-------------+-------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+-------------+-------------+-------------+
| grid_size  | 11          | 22          | 44          |
| loss       | 0.081657    | 0.443279    | 0.439904    |
| x          | 0.001900    | 0.017087    | 0.025622    |
| y          | 0.005896    | 0.021427    | 0.074824    |
| w          | 0.004938    | 0.005906    | 0.012974    |
| h          | 0.001626    | 0.003649    | 0.010029    |
| conf       | 0.067297    | 0.395210    | 0.316455    |
| cls        | 0.000000    | 0.000000    | 0.000000    |
| cls_acc    | 100.00%     | 100.00%     | 100.00%     |
| recall50   | 1.000000    | 1.000000    | 1.000000    |
| recall75   | 1.000000    | 1.000000    | 0.833333    |
| precision  | 1.000000    | 0.600000    | 0.333333    |
| conf_obj   | 0.971575    | 0.898328    | 0.912598    |
| conf_noobj | 0.000337    | 0.001527    | 0.001043    |
+------------+-------------+-------------+-------------+
Total loss 0.9648399949073792
---- ETA 0:00:03.870108

---- [Epoch 94/95, Batch 261/262] ----
+------------+-------------+-------------+-------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+-------------+-------------+-------------+
| grid_size  | 15          | 30          | 60          |
| loss       | 0.389350    | 0.275126    | 0.278601    |
| x          | 0.006320    | 0.021978    | 0.055561    |
| y          | 0.005157    | 0.006616    | 0.032199    |
| w          | 0.021817    | 0.007167    | 0.022667    |
| h          | 0.018181    | 0.010845    | 0.020335    |
| conf       | 0.337873    | 0.228521    | 0.147839    |
| cls        | 0.000002    | 0.000000    | 0.000000    |
| cls_acc    | 100.00%     | 100.00%     | 100.00%     |
| recall50   | 1.000000    | 1.000000    | 1.000000    |
| recall75   | 0.750000    | 1.000000    | 0.750000    |
| precision  | 0.800000    | 0.400000    | 0.250000    |
| conf_obj   | 0.882394    | 0.960067    | 0.978124    |
| conf_noobj | 0.001452    | 0.001024    | 0.000586    |
+------------+-------------+-------------+-------------+
Total loss 0.9430781602859497
```

Entrenamiento 6

- 96 épocas (iteraciones)
- Tamaño de lote de 2
- Conjunto de datos dividido en:
  - 524 imágenes para entrenamiento (train)
  - 59 imágenes para validación (valid)
  - 58 imágenes para evaluación (test)
- **Función de pérdida:**

  Valor final de 0.9884



```
efra@ASUS: ~/entrenamiento641-95/deteccion-objeto
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
---- [Epoch 95/96, Batch 260/262] ----
+------------+-------------+-------------+-------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+-------------+-------------+-------------+
| grid_size  | 14          | 28          | 56          |
| loss       | 0.344316    | 0.375673    | 0.618021    |
| x          | 0.011149    | 0.019599    | 0.052294    |
| y          | 0.019718    | 0.016186    | 0.050486    |
| w          | 0.011376    | 0.003161    | 0.004195    |
| h          | 0.008223    | 0.014740    | 0.017746    |
| conf       | 0.293850    | 0.321985    | 0.493299    |
| cls        | 0.000001    | 0.000001    | 0.000000    |
| cls_acc    | 100.00%     | 100.00%     | 100.00%     |
| recall50   | 0.833333    | 1.000000    | 0.923077    |
| recall75   | 0.750000    | 1.000000    | 0.923077    |
| precision  | 1.000000    | 0.928571    | 0.324324    |
| conf_obj   | 0.862445    | 0.891369    | 0.892280    |
| conf_noobj | 0.000925    | 0.001586    | 0.001761    |
+------------+-------------+-------------+-------------+
Total loss 1.338010311126709
---- ETA 0:00:03.870108

---- [Epoch 95/96, Batch 261/262] ----
+------------+-------------+-------------+-------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+-------------+-------------+-------------+
| grid_size  | 12          | 24          | 48          |
| loss       | 0.092116    | 0.447547    | 0.448821    |
| x          | 0.010720    | 0.015267    | 0.040589    |
| y          | 0.020830    | 0.008232    | 0.012222    |
| w          | 0.013737    | 0.008245    | 0.004796    |
| h          | 0.014850    | 0.011261    | 0.012338    |
| conf       | 0.031979    | 0.404542    | 0.378877    |
| cls        | 0.000001    | 0.000000    | 0.000000    |
| cls_acc    | 100.00%     | 100.00%     | 100.00%     |
| recall50   | 1.000000    | 0.928571    | 1.000000    |
| recall75   | 1.000000    | 0.928571    | 1.000000    |
| precision  | 1.000000    | 0.812500    | 0.400000    |
| conf_obj   | 0.984501    | 0.859973    | 0.917973    |
| conf_noobj | 0.000159    | 0.001381    | 0.001545    |
+------------+-------------+-------------+-------------+
Total loss 0.9884840250015259
---- ETA 0:00:00
```

3.1.7. Diseñar un acomodo de costales en la estiba que permita el cálculo del volumen de la forma más exacta posible, asimismo crear un programa que lo pueda realizar.

Para que funcione el cálculo del volumen de costales se requiere que la estiba tenga acomodados los costales tal como se ve en la figura 16, en donde el primer piso de la estiba está acomodado con la parte más angosta del costal al frente, luego se coloca el segundo piso con la parte más larga hacia el frente, después nuevamente se colocan los costales con la parte más angosta hacía el frente y se repite el proceso hasta terminar la estiba.

Figura 16. Acomodo de costales

La figura 17 muestra en (I) el apilamiento de la primera capa (inferior), donde b es el número de costales en la primera fila, y c es el número de costales a través del apilamiento. (II) muestra el apilamiento de la segunda capa (de abajo hacia arriba). Como puede verse, el patrón de superposición de los costales está

destinado a entrelazar las capas, asegurando que todos los costales permanezcan en la pila. Para facilitar el cálculo del volumen b y c siempre deben ser número par.

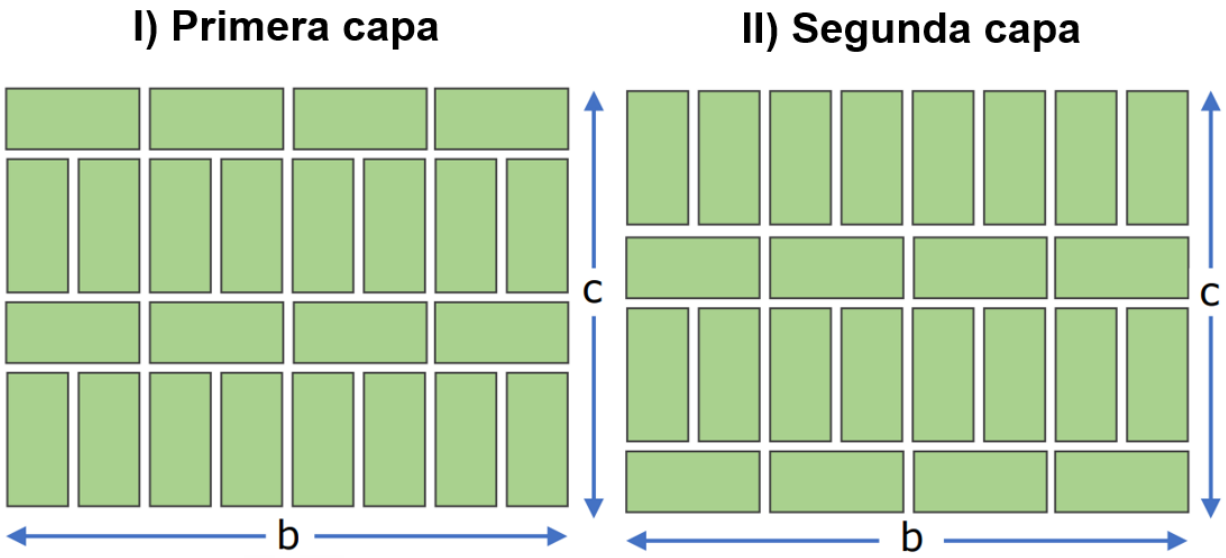


Figura 17. Distribución secuencial de capas de la estiba de costales

Independientemente de la altura de la pila, el volumen se puede calcular fácilmente considerando que cada capa tiene el mismo número de costales (excepto la capa superior), así que, como primer paso, el área de la primera capa se calcula de acuerdo con la siguiente ecuación.

$$Area = \left(b + \frac{b}{2}\right)\left(\frac{c}{2}\right)$$

La última ecuación proviene del hecho de que la profundidad de una capa contiene un número par de filas, pero dependiendo de la capa, la primera fila puede contener el doble de costales que la siguiente fila (entrelazada), o puede

contener la mitad de los costales que la siguiente fila, por lo que cada par de filas en cada capa suma $b + \frac{b}{2}$ costales donde b es el conteo de costales en la primera capa. Por lo tanto, para calcular el área, este producto se tiene que multiplicar por el total de los pares de hileras arriba descritos, que, en cada capa, es la mitad de la cuenta de costales o $\frac{c}{2}$. La expresión anterior podría simplificarse mediante la siguiente fórmula.

$$Area = \frac{3bc}{4}$$

Una vez que se obtiene el área para calcular el volumen, el área se multiplica por la altura de la pila, dando como resultado la ecuación.

$$Volumen = \frac{3abc}{4}$$

En base a las operaciones anteriores, se desarrolló el siguiente código mostrado en la figura 20, que permite calcular el volumen de la estiba, en ella también se puede observar cómo calcula el volumen de los ejemplos de estiba de la figura 18. En este programa se toma en cuenta un posible error de conteo, en el cual falte contar un costal en b o en c, el código que se encarga de arreglarlo para dar un volumen correcto. En la imagen 19 se muestra la salida del programa de cálculo de volumen.

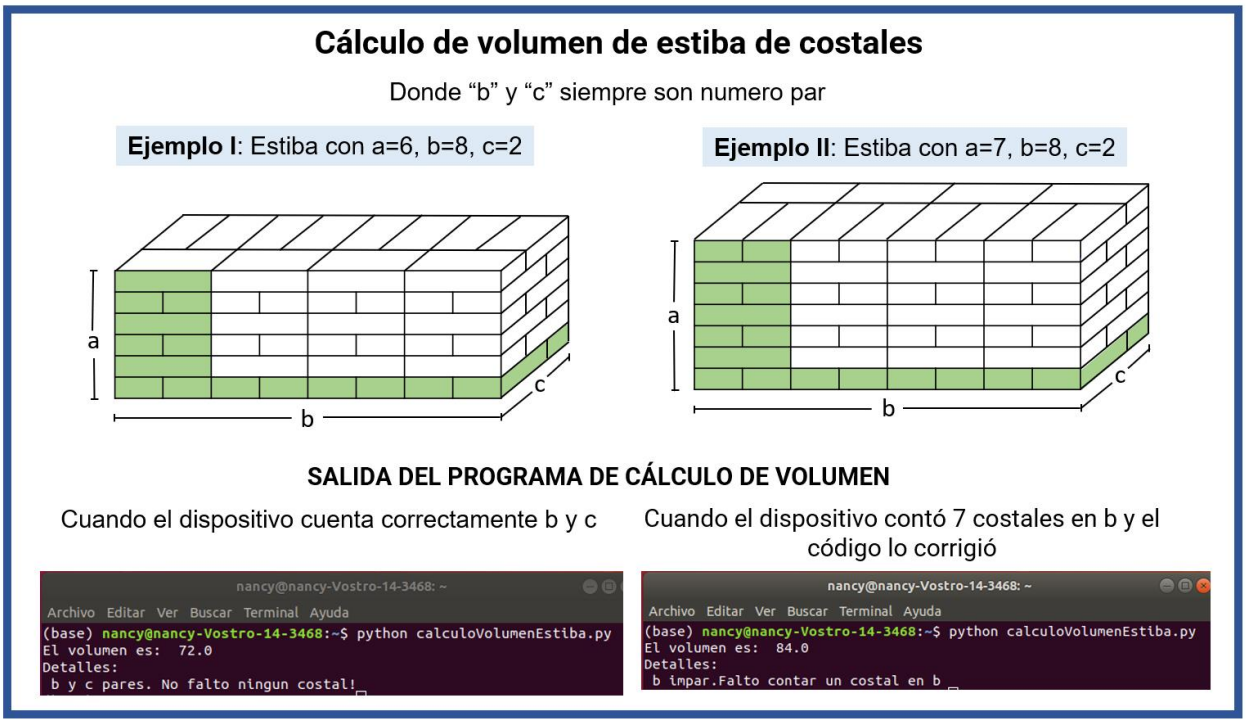


Figura 18. Ejemplo de cálculo de volumen

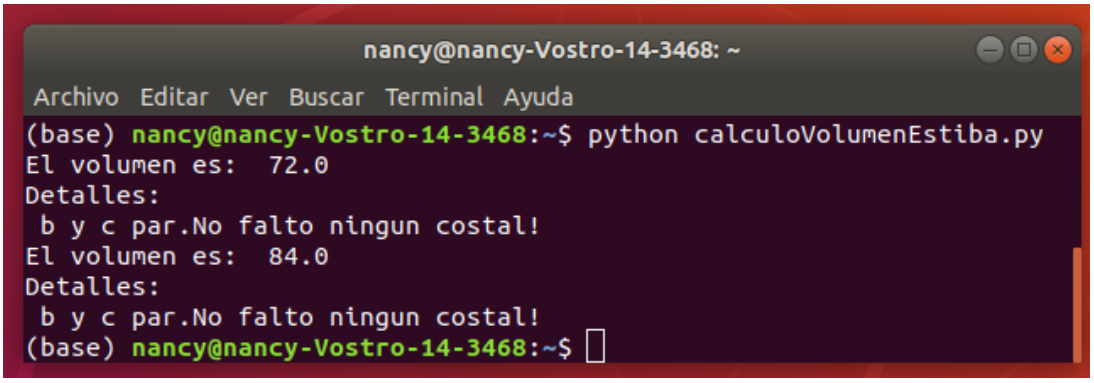


Figura 19. Salida del programa de cálculo de volumen

```python
# a  altura, puede ser par o impar
# b  ancho, siempre va a ser par
# c  profundidad, siempre va a ser par

def volumen(a, b, c):

    # condiciones para saber si b y c es par o impar
    # si es b o c es impar, el escaner omitio uno, por error por lo que se le suma 1
    if b % 2 == 0 and c % 2 == 0:
        area = (3*b*c)/4
        tipo = "b y c par, todo bien!"
    elif b % 2 != 0 and c % 2 != 0:
        area = (3*(b+1)*(c+1))/4
        tipo = "b y c impar.Falto contar un costal en b y c"
    elif b % 2 == 0 and c % 2 != 0:
        area = (3*b*(c+1))/4
        tipo = "c impar.Falto contar un costal en c"
    elif b % 2 != 0 and c % 2 == 0:
        area = (3*(b+1)*c)/4
        tipo = "b impar.Falto contar un costal en b"

    volumen = area*a
    print("El volumen es: ", volumen)
    print("Detalles: \n", tipo)
    return


# Ejemplo de valores escaneados de una estiba

volumen(6, 8, 2)  # a=6, b=8, c=2
volumen(7, 8, 2)  # a=7, b=8, c=2
```

Figura 20. Programa de cálculo de volumen de estiba de costales

3.1.8. Realizar un código que cuente los costales, calcule el volumen y masa en base a ese conteo, para posteriormente guardarlo en un archivo Excel.

En la figura 21 se ve en el lado izquierdo una captura de la terminal de cómo va contando los costales que va detectando y en el lado derecho se ve la detección, cada costal detectado se marca con un recuadro y en la parte

superior dice el nombre de la clase a la que pertenece, es decir, que lo identifica como “costal”. En el conteo se puede ver varias veces el texto de “La cantidad de costales es: 3” esto se debe a que la detección es por cada fotograma que constituye el video.

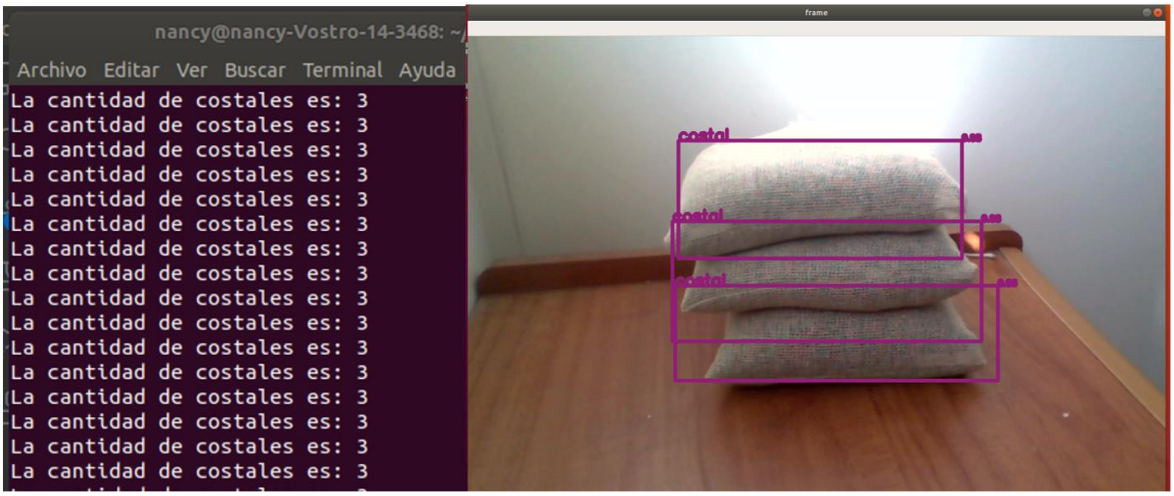


Figura 21. Detección y conteo de costales

En la figura 22 se ve cómo se genera un archivo de Excel llamado “Costal” después de finalizar el escaneo.

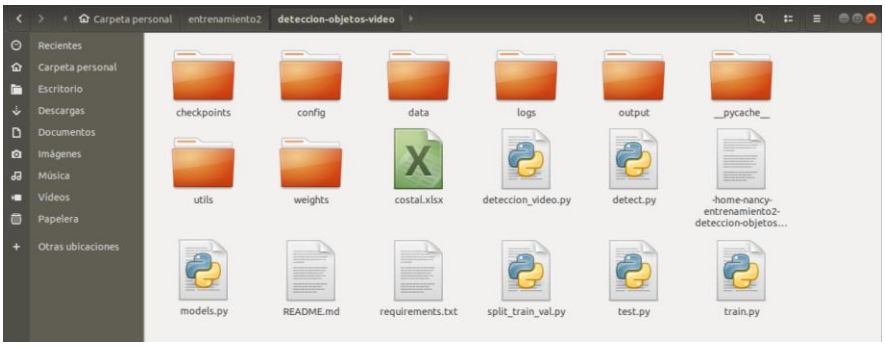


Figura 22. Generación de archivo de Excel

El archivo de Excel se conforma por una hoja de cálculo llamada “Conteo de Costal”, en ella se encuentra fecha y hora en que se hizo el conteo, cantidad

total de costales detectados y la masa total de acuerdo al número de costales. En la figura 23 se muestra un ejemplo de la hoja de Excel, en la cual se toman los costales que tienen una masa individual de 1.5Kg, estos cálculos están basados en la detección de la figura 21.

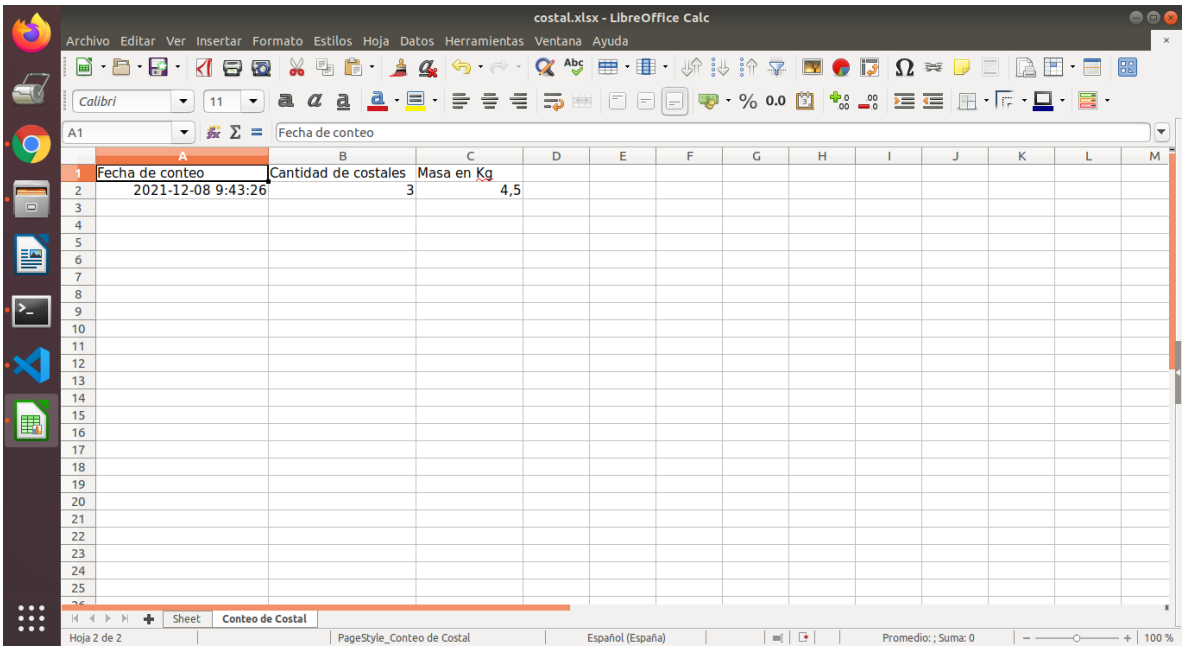


Figura 23. Archivo de Excel "Costal"

3.1.9. Diseñar carcasa del dispositivo.

La carcasa del dispositivo está contemplada para ser impresa en 3D de material PLA, esta debe tener buena ventilación y un área para colocar un ventilador, ya que la Jetson Nano puede llegar a calentarse mucho si se usa por un largo periodo de tiempo y a pesar de que ésta tiene un disipador, es necesario mantener una buena ventilación para su correcto funcionamiento. En la figura 24 se puede observar un modelo CAD hecho en SolidWorks, el cual simula la apariencia del prototipo final.

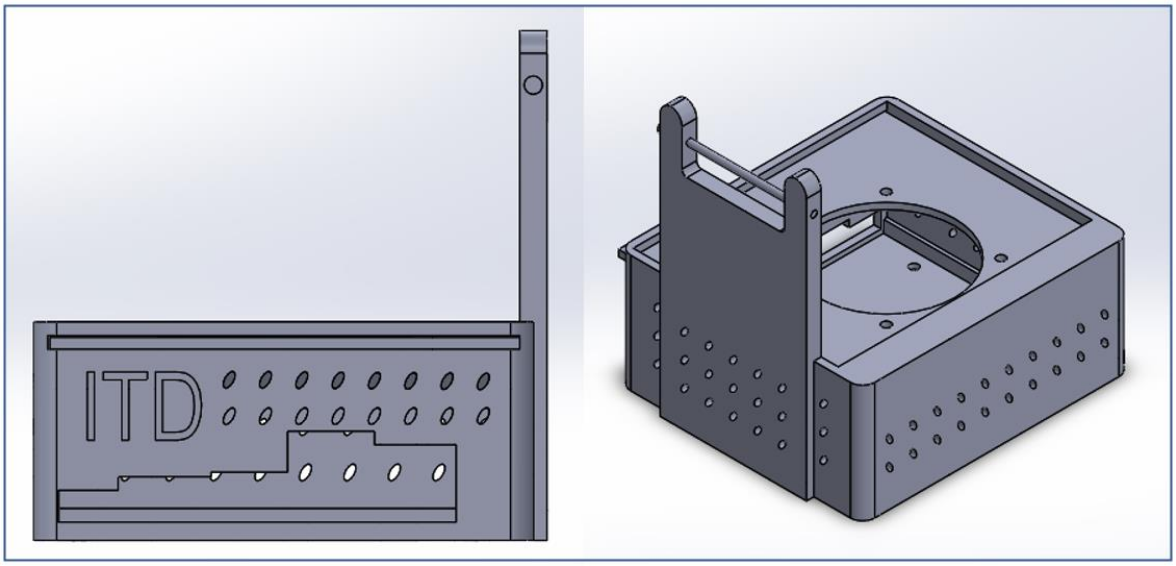


Figura 24. Diseño CAD de carcasa

3.1.10. Planeación de funcionamiento.

Se pasa el dispositivo por las dimensiones a escanear *a*, *b* y *c* de la estiba de costales con el acomodo mostrado anteriormente, recordando que este dispositivo es un conjunto que con otro proyecto que se encargará de la movilización, el vehículo que lo moverá enviará un pulso de aviso según vaya acabando de recorrer cada dimensión, esa señal sirve para comenzar y finalizar el conteo, en total serían 3 pulsos, empieza el programa y comienza a detectar y contar, llega el primer pulso y se termina el conteo de *a*, vuelve a comenzar la detección, llega el segundo pulso con el cual saca *b*, después de esa interrupción comienza de nuevo a detectar para el conteo de *c*, con el último pulso se finaliza, con esto ya se tienen los valores de *a*, *b* y *c* necesarios para calcular el volumen con un programa en Python, el cual tiene las operaciones necesarias para sacar de forma exacta el volumen.

Posteriormente, se calcula la masa multiplicando el valor del volumen, es decir, el número total de costales que conforman la estiba para finalmente almacenar los resultados de volumen y masa en un archivo Excel en donde también se verán la hora y fecha exacta en que se obtuvieron.

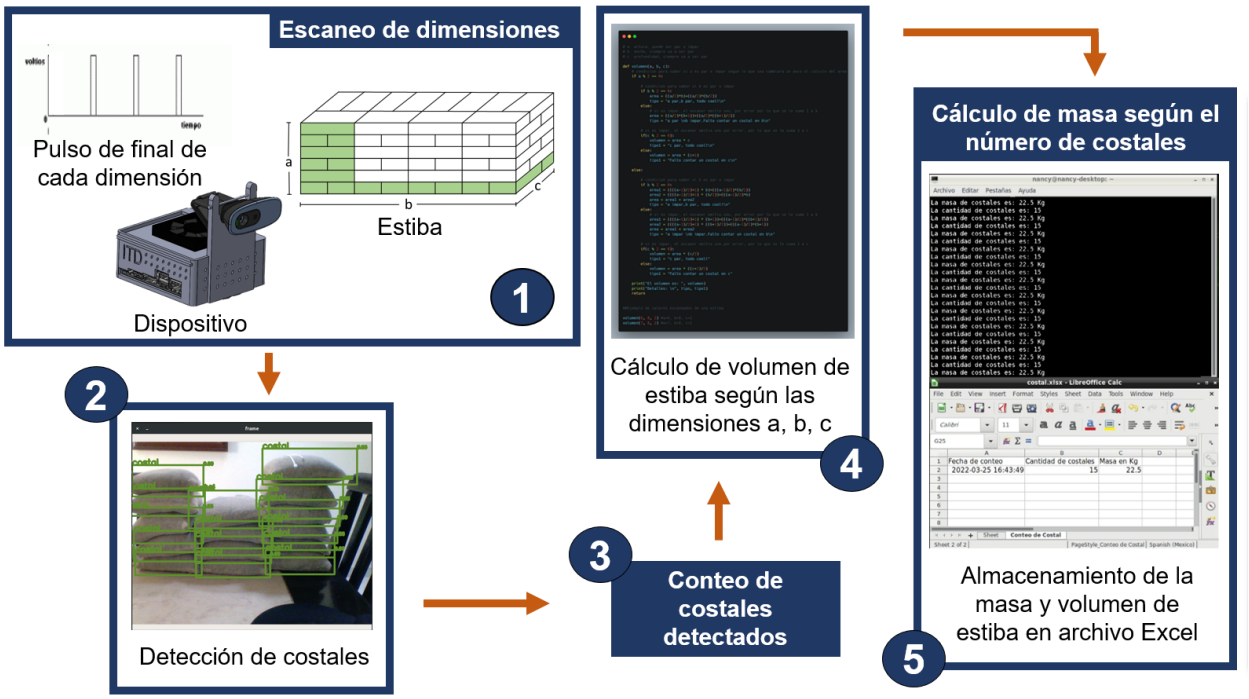


Figura 25. Funcionamiento del dispositivo

3.1.11. Armar dispositivo integrando el software y hardware para el prototipo final.

La figura 26 muestra un diseño CAD elaborado en SolidWorks con el que se ejemplifica el dispositivo final, está conformado por la carcasa, cámara web, Jetson Nano y ventilador. Este proyecto es una parte que trabaja en conjunto

con el trabajo del Ingeniero Omar Vargas, el cual está desarrollando un sistema para mover de forma automática el dispositivo de escaneo de estibas, este sistema enviará un pulso al dispositivo para que comience a escanear cuando llegue al final de cada lado de la estiba que debe medir, por lo que para simular esa señal se utiliza un botón conectado a las entradas de la Jetson Nano, y se presiona cada vez que se finalice la medición de un lado de la estiba.

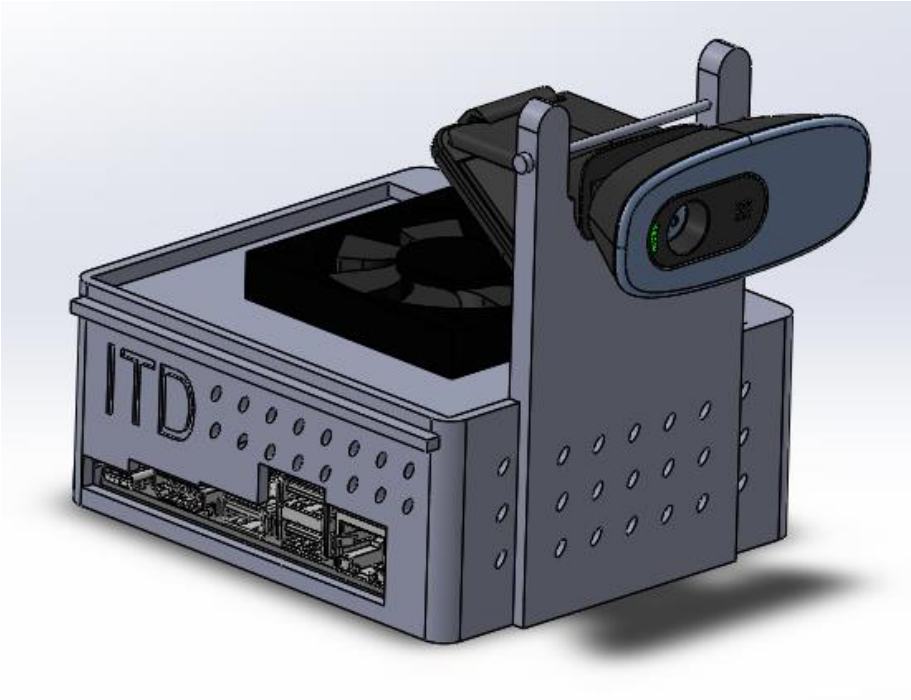


Figura 26. Diseño CAD de dispositivo final

3.1.12. Hacer pruebas de dispositivo

Se realizaron pruebas para ver si el dispositivo ejecutaba de manera óptima el algoritmo, así como su correcto funcionamiento. Se pudo observar que el algoritmo era demasiado pesado para ejecutarse en la Jetson Nano, por lo que

se intentó aumentar la memoria RAM haciendo uso de la SD. Debido a los resultados de estas pruebas se buscaron otras minicomputadoras donde se pueda ejecutar sin elevar tanto el costo final del dispositivo.

3.2. Resultados

En esta sección se observan los resultados obtenidos del desarrollo de este proyecto, asimismo se muestran las modificaciones derivadas de las pruebas que se iban realizando.

3.2.1 Pruebas y evaluación del funcionamiento del modelo de detección de costales.

Para realizar la evaluación del modelo y calcular su mAP y f1, se ejecuta el programa *test.py* el cual hace uso del conjunto de evaluación. Esto se hace para cada uno de los modelos entrenados y ver cuál es el mejor. Los valores de mAP y f1 se evaluaron según el *conf_thres*, que es el umbral de confianza del objeto que le dice al modelo que solo asuma que es un costal, si esta 85% o 95% seguro para este caso.

3.2.1.1. Entrenamiento 1

Este entrenamiento dio un mAP de 0.7441 y un f1 de 0.8434 con un umbral de confianza de 0.85.

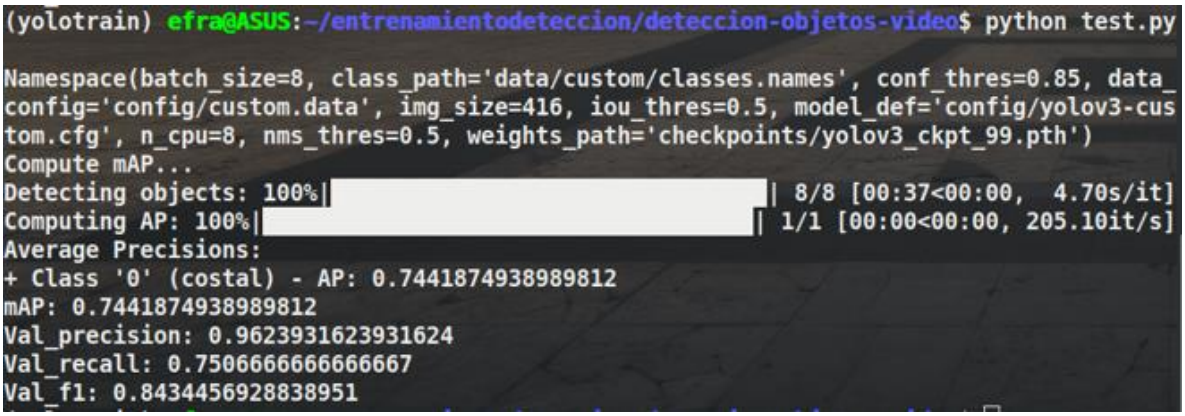


Figura 27. Métrica de evaluación con un umbral de confianza de 0.85 del Entrenamiento 1

Se obtuvo un mAP de 0.5447 y f1 de 0.7053, con un umbral de confianza de 0.95, como se ve en la figura 28.

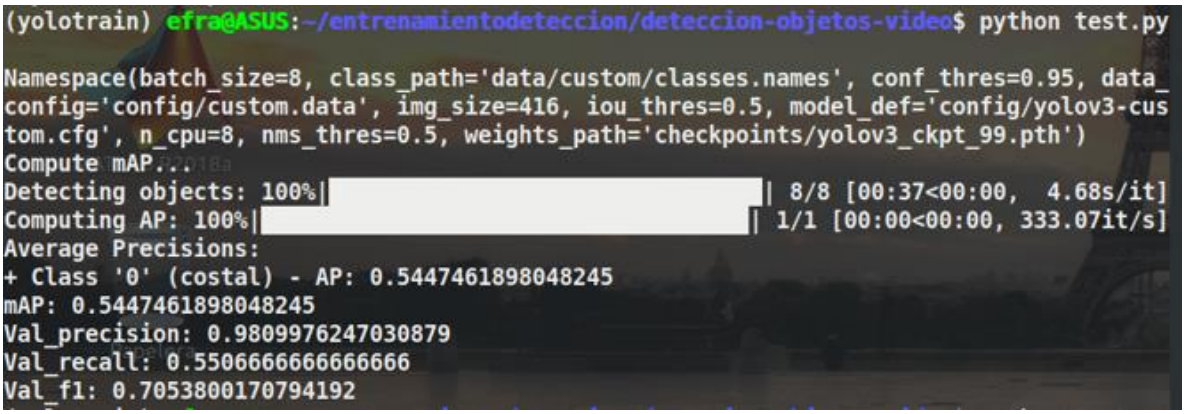


Figura 28. Métrica de evaluación con un umbral de confianza de 0.95 del Entrenamiento 1

A continuación, se puede observar una imagen de detección con este entrenamiento con un umbral de confianza de 0.85.

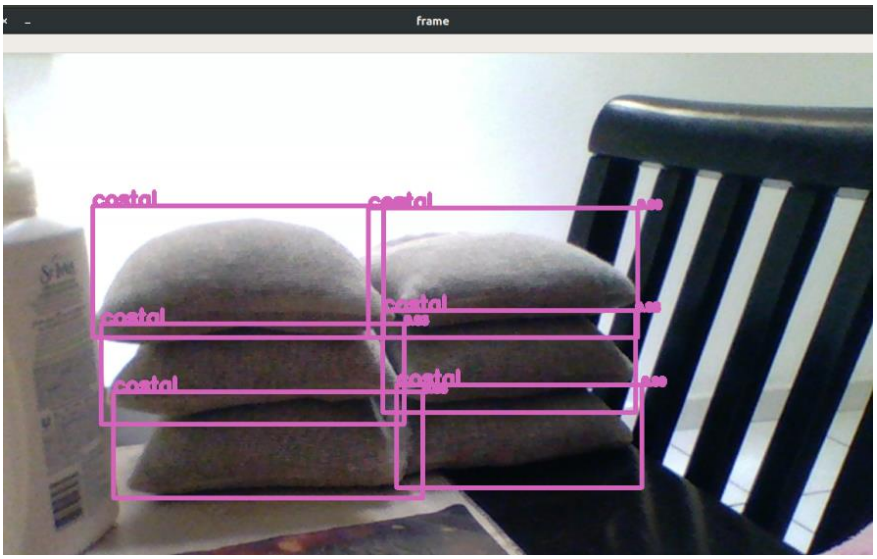


Figura 29. Detección de costales Entrenamiento 1

3.2.1.2. Entrenamiento 2

Este entrenamiento dio un mAP de 0.8392 y f1 de 0.9092, con un umbral de confianza de 0.85.

```
(yolotrain) efra@ASUS:~/entrenamiento400-120/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.85, data
_config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-c
ustom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_119.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [00:37<00:00,  4.64s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 229.33it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.8392862074700349
mAP: 0.8392862074700349
Val_precision: 0.9890282131661442
val_recall: 0.8413333333333334
val_f1: 0.9092219020172911
```

Figura 30. Métrica de evaluación con un umbral de confianza de 0.85 del Entrenamiento 2

Se obtuvo un mAP de 0.7024 y f1 de 0.8237, con un umbral de confianza de 0.95, como se puede ver en la figura 31.

```
(yolotrain) efra@ASUS:~/entrenamiento400-120/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.95, data
_config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-c
ustom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_119.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [00:37<00:00,  4.63s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 496.60it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.7024110287578276
mAP: 0.7024110287578276
Val_precision: 0.9924812030075187
val_recall: 0.704
val_f1: 0.8237129485179406
```

Figura 31. Métrica de evaluación con un umbral de confianza de 0.95 del Entrenamiento 2

A continuación, se puede observar una imagen de detección con este entrenamiento con un umbral de confianza de 0.85.

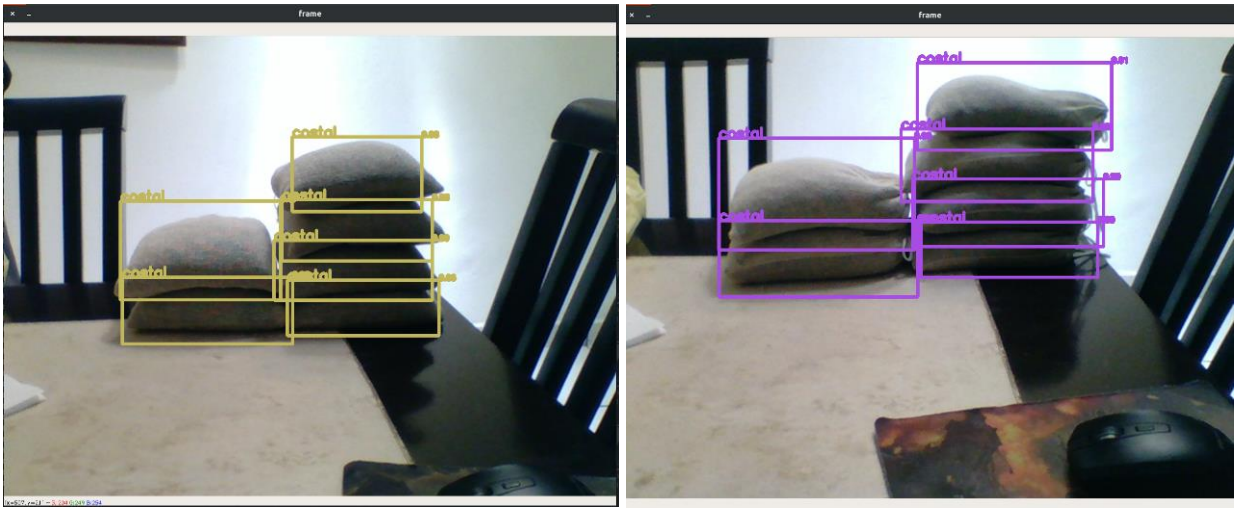


Figura 32. Detección de costales de Entrenamiento 2

Se puede observar que a pesar de que la cantidad del conjunto de datos que se usó era el mismo en el entrenamiento 1 y 2, el segundo entrenamiento al ser de más épocas, el entrenamiento mejoró y pasó de tener un mAP de 0.7441 a 0.8391, sin embargo, el valor de f1 disminuyó.

3.2.1.3 Entrenamiento 3

Este entrenamiento dio un mAP de 0.7104 y un f1 de 0.8296 con un umbral de confianza de 0.85.

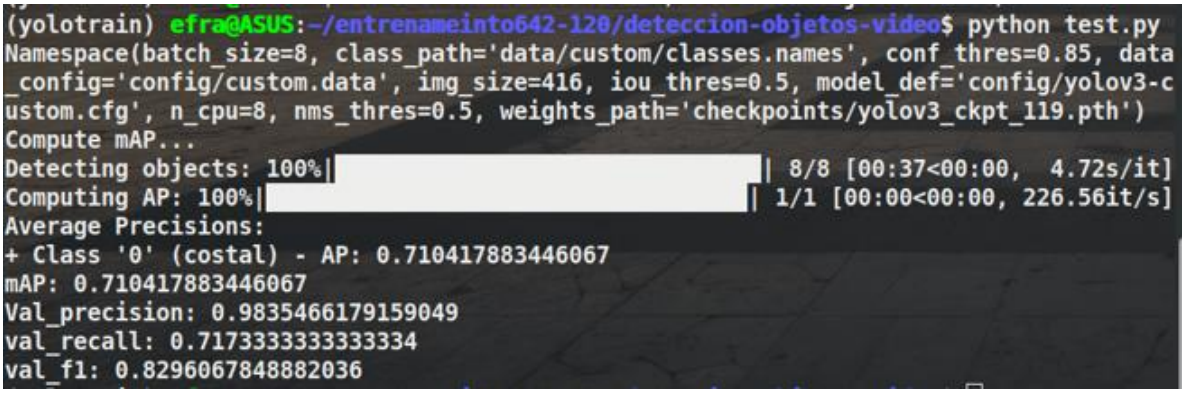


Figura 33. Métrica de evaluación con un umbral de confianza de 0.85 del Entrenamiento 3

Se obtuvo un mAP de 0.4856 y f1 de 0.6524, con un umbral de confianza de 0.95, como se ve en la figura 34.

```
(yolotrain) efra@ASUS:~/entrenameinto642-120/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.95, data
_config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-c
ustom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_119.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [00:37<00:00,  4.71s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 264.17it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.4856403201768263
mAP: 0.4856403201768263
Val_precision: 0.9786666666666667
val_recall: 0.48933333333333334
val_f1: 0.6524444444444445
```

Figura 34. Métrica de evaluación con un umbral de confianza de 0.95 del Entrenamiento 3

A continuación, se puede observar una imagen de detección con este entrenamiento con un umbral de confianza de 0.85.

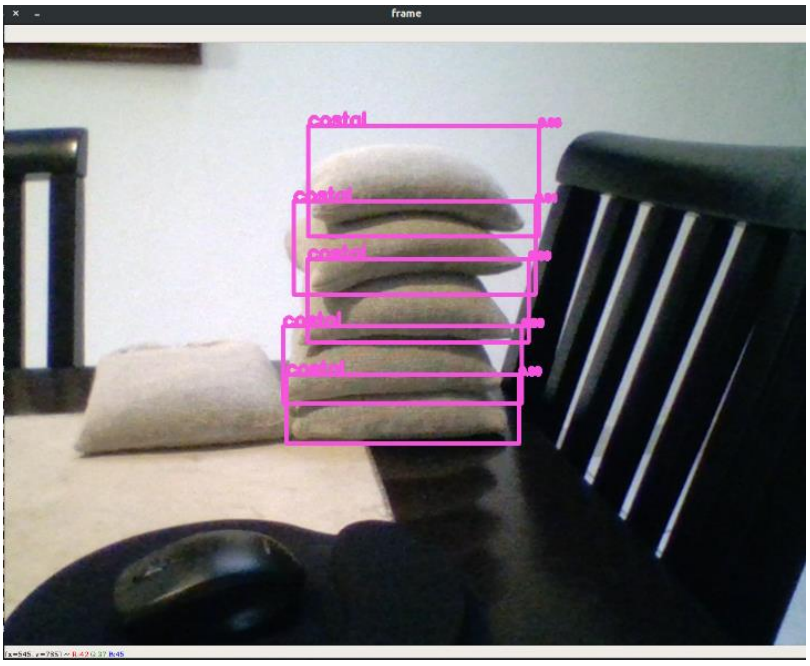


Figura 35. Detección de costales de Entrenamiento 3

En este entrenamiento en comparación a los anteriores, se observa que, a pesar de tener más épocas, no se mejora la precisión en mAP y F1, sino que estas disminuyen, lo cual indicó que, con la cantidad en el conjunto de datos, lo mejor para el modelo era disminuir el número de épocas, a diferencia de los primeros entrenamientos, que con menos imágenes se necesitaban mayor número de épocas.

3.2.1.4. Entrenamiento 4

Este entrenamiento dio un mAP de 0.8910 y un f1 de 0.9311 con un umbral de confianza de 0.85.

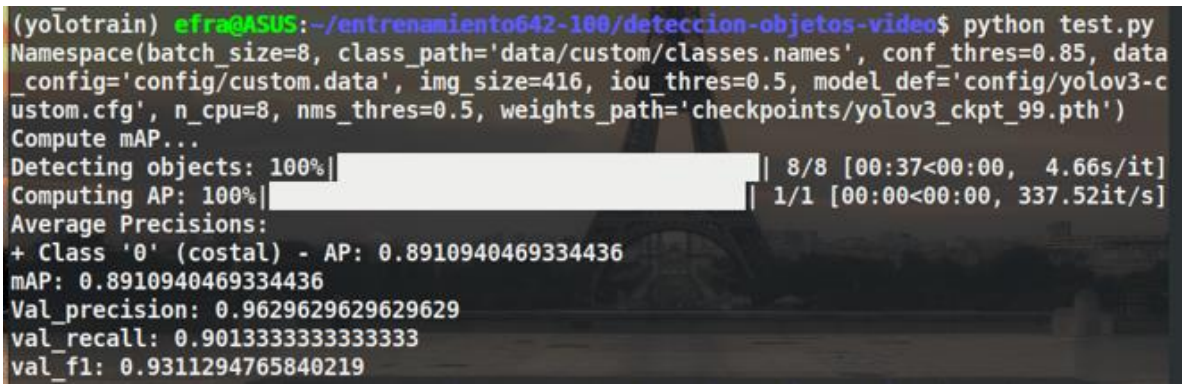


Figura 36. Métrica de evaluación un con umbral de confianza de 0.85 del Entrenamiento 4

Se obtuvo un mAP de 0.7453 y un f1 de 0.8483, con un umbral de confianza de 0.95, como se muestra en la figura 37.

```
(yolotrain) efra@ASUS:~/entrenamiento642-100/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.95, data
_config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-c
ustom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_99.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [00:37<00:00,  4.67s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 198.64it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.7453362628244864
mAP: 0.7453362628244864
Val_precision: 0.9707903780068728
val_recall: 0.7533333333333333
val_f1: 0.8483483483483483
```

Figura 37. Métrica de evaluación con umbral de confianza de 0.95 del Entrenamiento 4

A continuación, se puede observar una imagen de detección con este entrenamiento con un umbral de confianza de 0.85.

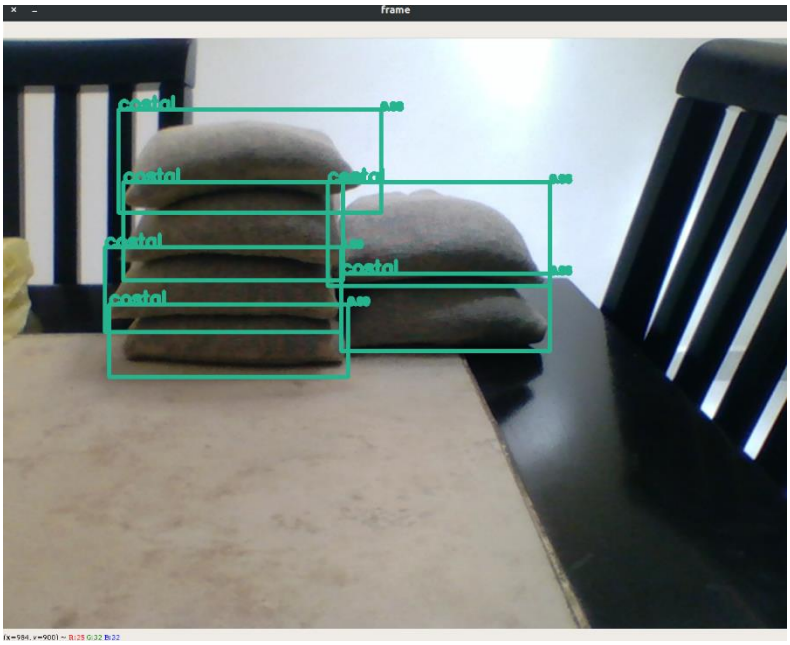


Figura 38. Detección de costales de Entrenamiento 4

Aquí ya se puede observar que, al disminuir el número de épocas en comparación con la anterior, se mejoró notablemente la exactitud del modelo, pues pasó de mAP de 0.7104 a 0.8910.

3.2.1.5 Entrenamiento 5

Este entrenamiento dio un mAP de 0.9235 y un f1 de 0.9555 con un umbral de confianza de 0.85.

Figura 39. Métrica de evaluación con umbral de confianza de 0.85 del Entrenamiento 5

Se obtuvo un mAP de 0.8311 y un f1 de 0.9026, con un umbral de confianza de 0.95, como se muestra en la figura 40.

```
(yolotrain) efra@ASUS:~/entrenamiento641-94/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.95, data_
config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-cus
tom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_94.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [01:20<00:00, 10.06s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 240.94it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.8311865640278141
mAP: 0.8311865640278141
Val_precision: 0.9827315541601256
Val_recall: 0.8346666666666667
Val_f1: 0.9026676279740447
```

Figura 40. Métrica de evaluación con umbral de confianza de 0.95 del Entrenamiento 6

A continuación, se puede observar una imagen de detección con este entrenamiento con un umbral de confianza de 0.85.

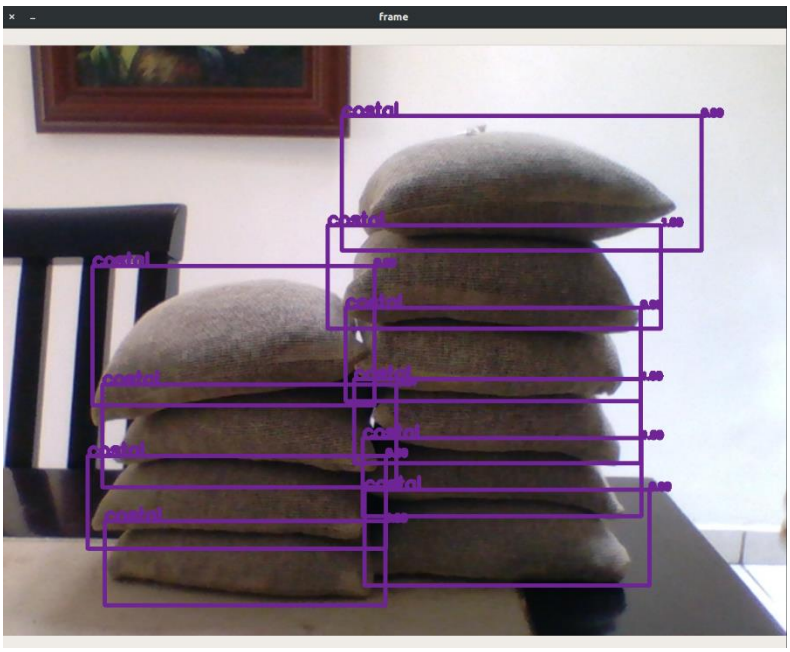


Figura 41. Detección de costales de Entrenamiento 5

3.2.1.6. Entrenamiento 6

Este entrenamiento dio un mAP de 0.9257 y un f1 de 0.9416 con un umbral de confianza de 0.85.

```
(yolotrain) efra@ASUS:~/entrenamiento641-95/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.85, data_
config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-cus
tom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_95.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [00:40<00:00,  5.04s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 168.24it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.9257663160247582
mAP: 0.9257663160247582
Val_precision: 0.9551630434782609
Val_recall: 0.9373333333333334
Val_f1: 0.946164199192463
```

Figura 42. Métrica de evaluación con umbral de confianza de 0.85 del Entrenamiento 6

Se obtuvo un mAP de 0.8116 y un f1 de 0.8892, con un umbral de confianza de 0.95, como se muestra en la figura 43.

```
(yolotrain) efra@ASUS:~/entrenamiento641-95/deteccion-objetos-video$ python test.py
Namespace(batch_size=8, class_path='data/custom/classes.names', conf_thres=0.95, data_
config='config/custom.data', img_size=416, iou_thres=0.5, model_def='config/yolov3-cus
tom.cfg', n_cpu=8, nms_thres=0.5, weights_path='checkpoints/yolov3_ckpt_95.pth')
Compute mAP...
Detecting objects: 100%|                                        | 8/8 [01:20<00:00, 10.00s/it]
Computing AP: 100%|                                             | 1/1 [00:00<00:00, 263.13it/s]
Average Precisions:
+ Class '0' (costal) - AP: 0.8116767853270601
mAP: 0.8116767853270601
Val_precision: 0.9730586370839936
Val_recall: 0.8186666666666667
Val_f1: 0.8892107168718321
```

Figura 43. Métrica de evaluación con umbral de confianza de 0.95 del Entrenamiento 6

A continuación, se puede observar una imagen de detección con este entrenamiento con un umbral de confianza de 0.85.

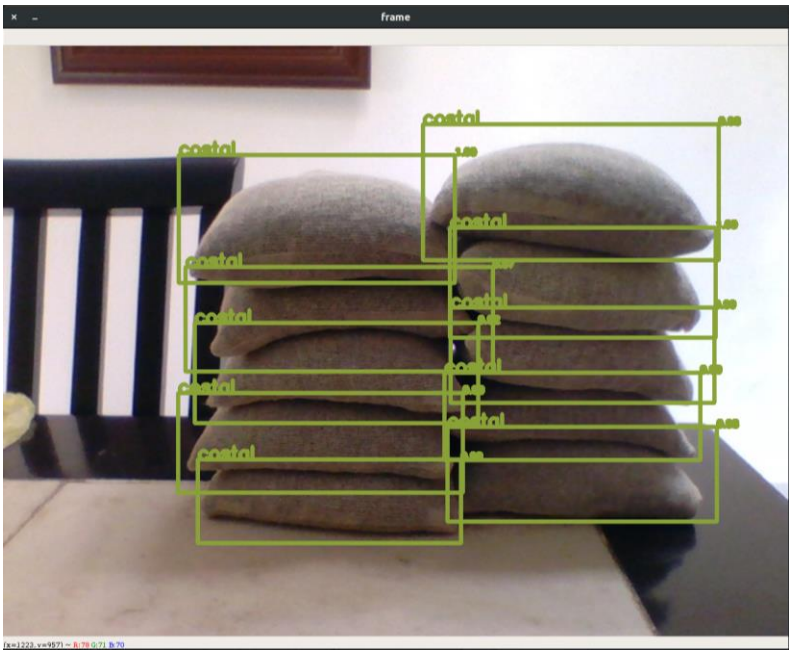


Figura 44. Detección de costales de Entrenamiento 6

3.2.2. Selección de entrenamiento en base a los resultados

En la siguiente tabla se puede comparar los resultados de las métricas de evaluación de cada entrenamiento para selección del mejor de todos, estos datos fueron sacados en un umbral de confianza de 0.85, el cual es suficiente para el tipo de objeto que se va a detectar, ya que es una forma definida que no cambiará mucho, así como la distancia en la cual se hará el escáner será cercana para captar solo los costales, por lo que no habrá otros objetos que puedan ser detectados.

Tabla 3. Métricas de evaluación de entrenamientos

Métrica de Evaluación	Entrenamientos					
	1	2	3	4	5	6
mAP	0.7441	0.8392	0.7104	0.8910	0.9235	0.9257
Precisión	0.9623	0.9890	0.9835	0.9629	0.9817	0.9551
Recall	0.7506	0.8413	0.7173	0.9013	0.9306	0.9373
F1-Score	0.8434	0.9092	0.8296	0.9311	0.9555	0.9461

El entrenamiento que mejor resultados tuvo en la evaluación, es el número 5 por lo que se utilizó en el dispositivo para la detección de costales. A continuación, se ve como detecta y cuenta 15 costales para posteriormente guardar el resultado en un archivo Excel.

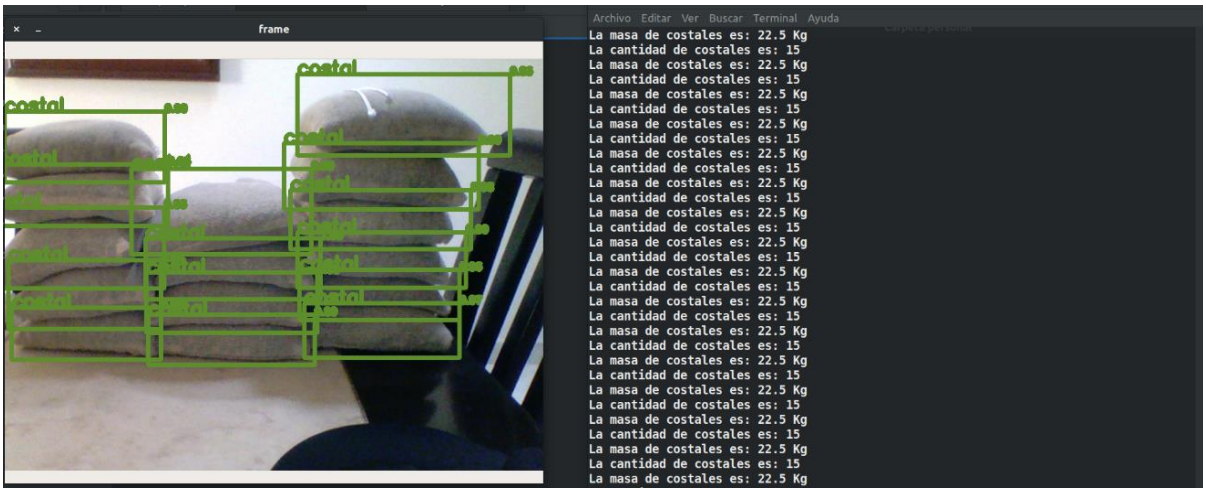


Figura 45. Detección y conteo de costales

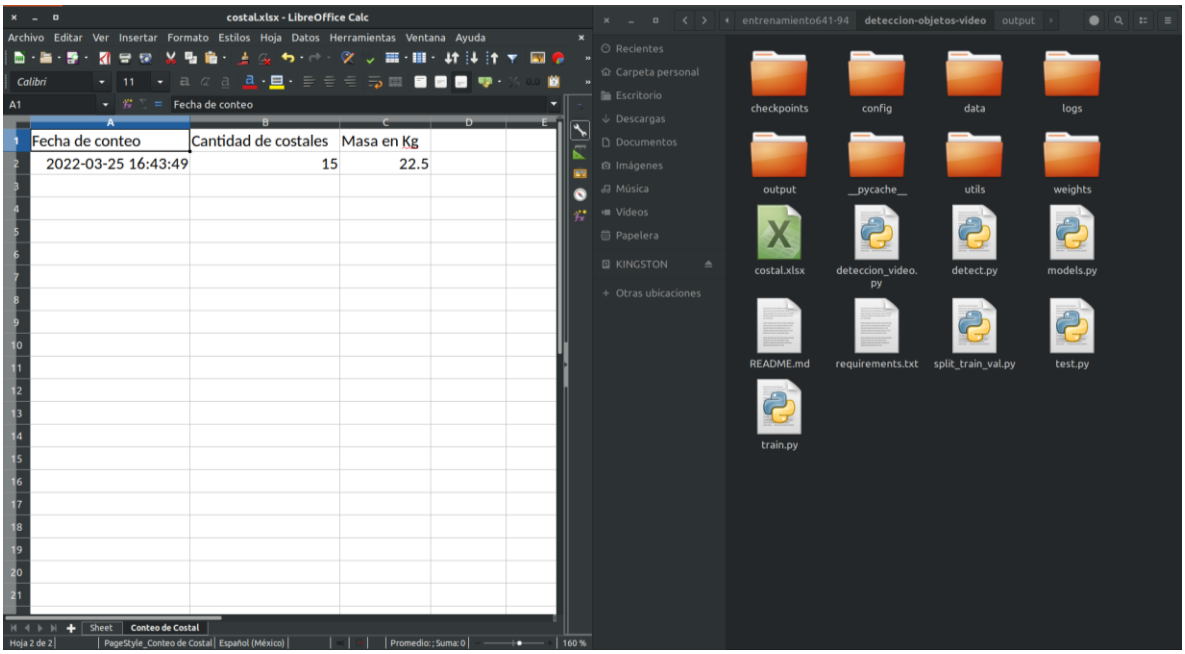


Figura 46. Resultado en Excel del conteo y detección de costales

3.2.3. Pruebas del algoritmo en una minicomputadora

A continuación, se puede ver una imagen del hardware del dispositivo, en donde se puede ver conectado un botón para simular los pulsos de cambio de escaneo de las dimensiones *a*, *b* y *c* de la estiba.

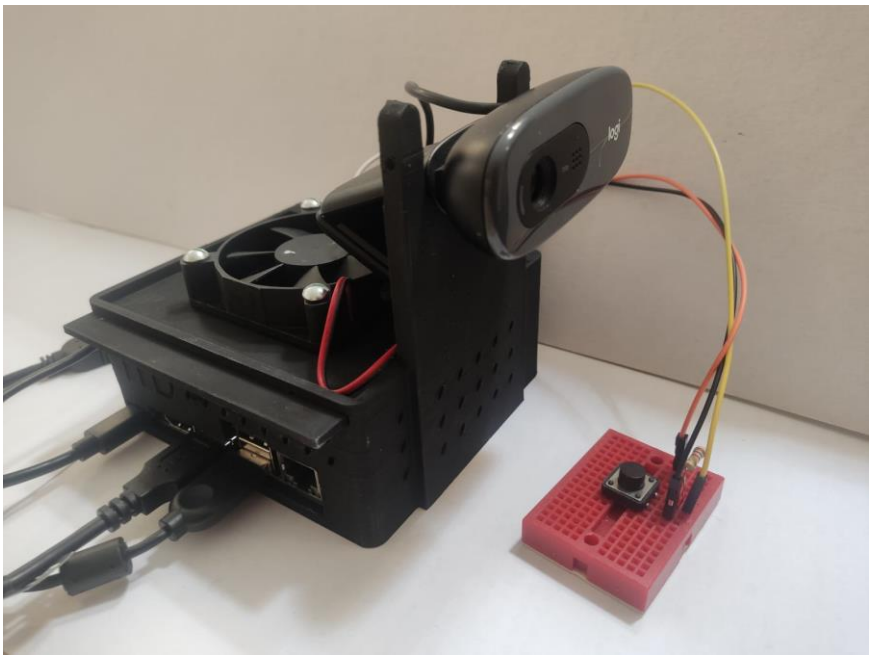


Figura 47. Dispositivo

Se hicieron diversas pruebas de funcionamiento en 2 laptops diferentes, en las cuales el algoritmo funcionó perfectamente, estas laptops son:

- Dell modelo empresarial de 8Gb de RAM, un disco de estado sólido de 250Gb con un procesador CORE i5 y sin tarjeta gráfica.
- ASUS modelo *gamer* de 32Gb de RAM, un disco de estado sólido de 134Gb con un procesador AMD Ryzen 53550 y una tarjeta gráfica AMD Raven.

Durante las pruebas se vio que la Jetson Nano de 2 Gb de RAM no es suficientemente potente para ejecutar el algoritmo actual, debido a esto se hizo una ampliación de memoria del dispositivo haciendo uso de la memoria SD, sin embargo, esto no fue suficiente, además la velocidad de procesamiento de la memoria SD no es tan rápida como la de la RAM, asimismo se eliminó el almacenaje del video llamado *output.mp4*, el cual es una video de lo detectado,

con esto se busca eliminar el procesamiento al ejecutar esa parte del código, haciendo un poco más ligero el mismo. También se hizo que la Jetson Nano tuviera el sistema operativo más ligero, así como solo las librerías necesarias para ejecutar el algoritmo. Sin embargo, a pesar de esto, el algoritmo sigue sin poder ser ejecutado en la Jetson Nano de 2Gb; luego se hizo una prueba en una Jetson Nano de 4Gb, en donde se puede ejecutar el programa, pero alrededor de los 3 minutos este se detiene y deja de funcionar, pues se queda sin suficiente memoria para procesar. Por lo que se trataron de ver más opciones para solucionar esto, las cuales consistieron:

1. Crear un nuevo algoritmo.
2. Cambiar de computadora.
3. Hacer procesamiento en la nube.
4. Hacer procesamiento por medio de un servidor local conectado a una red.

Análisis de las opciones:

Opción 1

Debido al tiempo que se tiene para finalizar este proyecto, se descarta esta opción, pues sería comenzar nuevamente con otra red y entrenamiento, así como evaluación de la misma, lo cual llevaría demasiado tiempo del cual no se dispone.

Opción 2

Actualmente debido a la escasez de computadoras pequeñas como Jetson Nano, Raspberry entre otras, además que el precio y dificultad para conseguirlas ha aumentado, ya que las Jetson Nano triplicaron su precio. Sin embargo, se pueden hacer pruebas con las computadoras a las cuales se

pueda tener acceso, además de buscar una opción que no aumente tanto el costo del prototipo, así como el que sea funcional para el algoritmo.

Opción 3

El procesamiento en la nube requiere que en los almacenes haya una buena conectividad a internet, con lo cual no se cuenta, sin mencionar que el procesamiento de video es pesado por lo que la contratación de internet específica para este dispositivo supondría un gasto más para la empresa, erogación que es una de las cosas que se desea evitar.

Opción 4

Crear una red interna para que la Jetson Nano envíe la información a un servidor local, implica que debe ser una computadora en especial para que sirva de servidor y haga el procesamiento, esto es un gasto extra para empresa, además del propio dispositivo de cálculo de volumen y masa, así como se necesitará un experto que sepa manejarlo y le dé mantenimiento cada determinado tiempo.

3.3.4. Selección de opción

Debido a lo descrito de cada opción se seleccionó la opción 2, ya que por la intervención del Mtro. Edgar López Díaz se logró tener acceso a una minicomputadora NUC con la que se pudieron hacer pruebas a distancia por medio de Teamviewer, para determinar el tipo de computadora en la que se puede ejecutar el algoritmo, dicha NUC tiene una arquitectura X86 a diferencia de la Jetson Nano que posee una arquitectura ARM. En este computador se puede ejecutar el algoritmo como en las laptops mencionadas anteriormente,

dando una referencia de las características necesarias de una computadora pequeña para correr este programa.

Figura 48. NUC

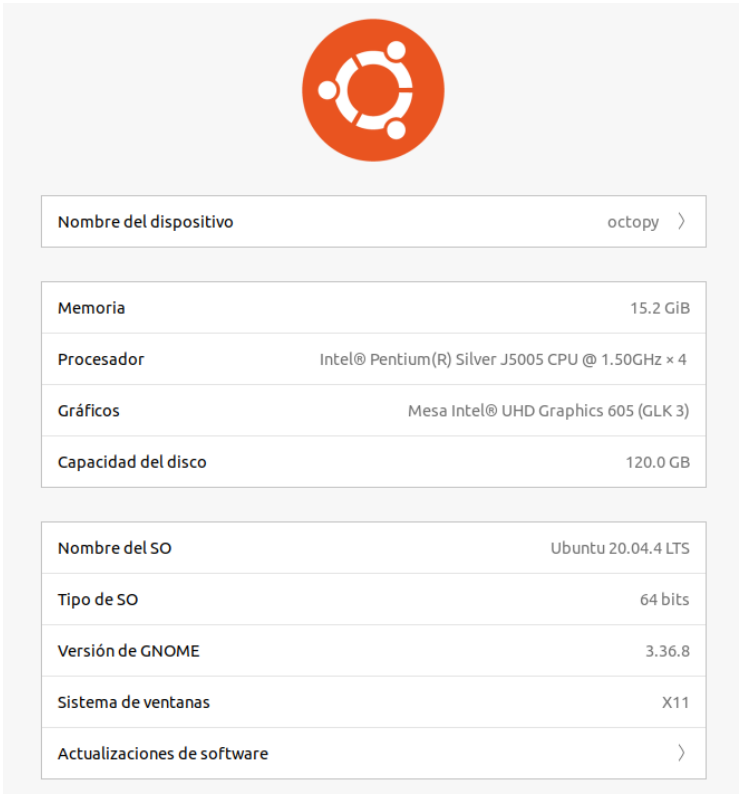


Figura 49. Características de la NUC

A continuación, se ve como corre el programa en la minicomputadora NUC y se activa la cámara para empezar a detectar.

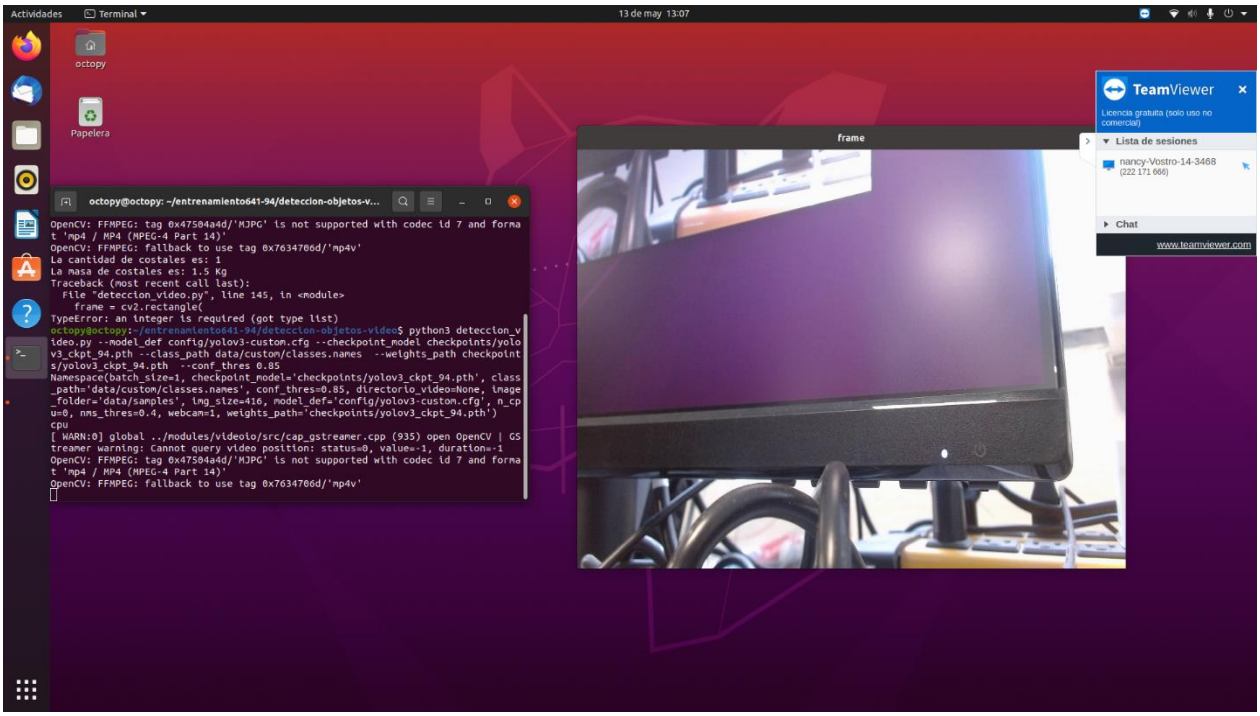


Figura 50. Ejecución de algoritmo de detección en NUC

En la siguiente figura se puede ver detección de costales ejecutada en la NUC, para esta detección se hizo uso de una Tablet con fotos de costales. Enseguida, se ven tres detecciones diferentes donde se pudo observar el correcto funcionamiento del algoritmo de conteo costales. Para aligerar un poco el procesamiento, se eliminó el recuadro que marcaba lo que iba detectando en la imagen, sin embargo, lo verdaderamente importante es que vaya contando los costales y calculando la masa según los costales contados, actividad que se hizo correctamente tal y como se aprecia en las figuras 51, 53 y 55.

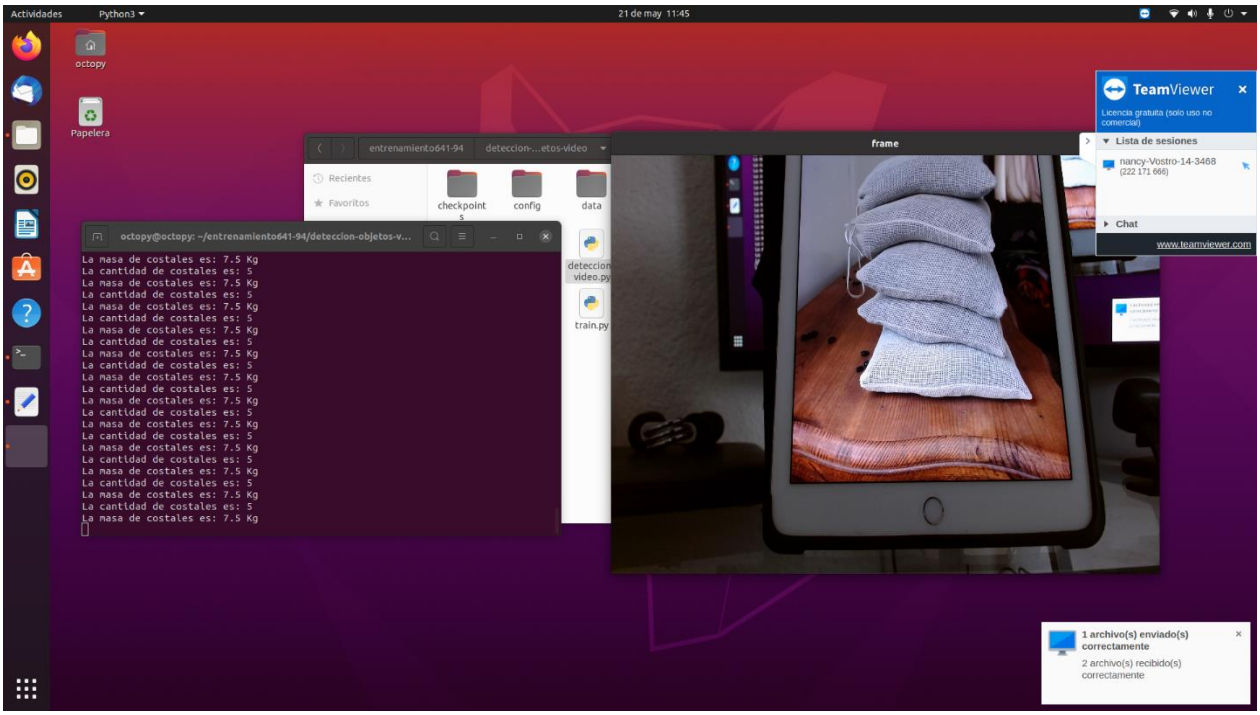


Figura 51. Detección y conteo de costales en NUC-1

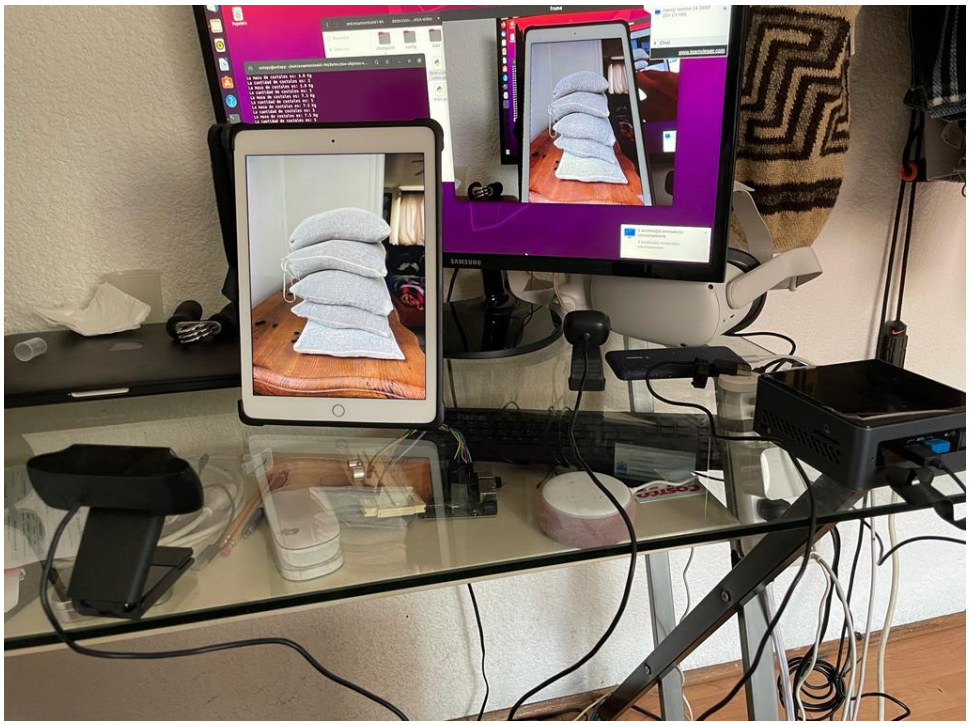


Figura 52. Prueba 1 en NUC-Hardware

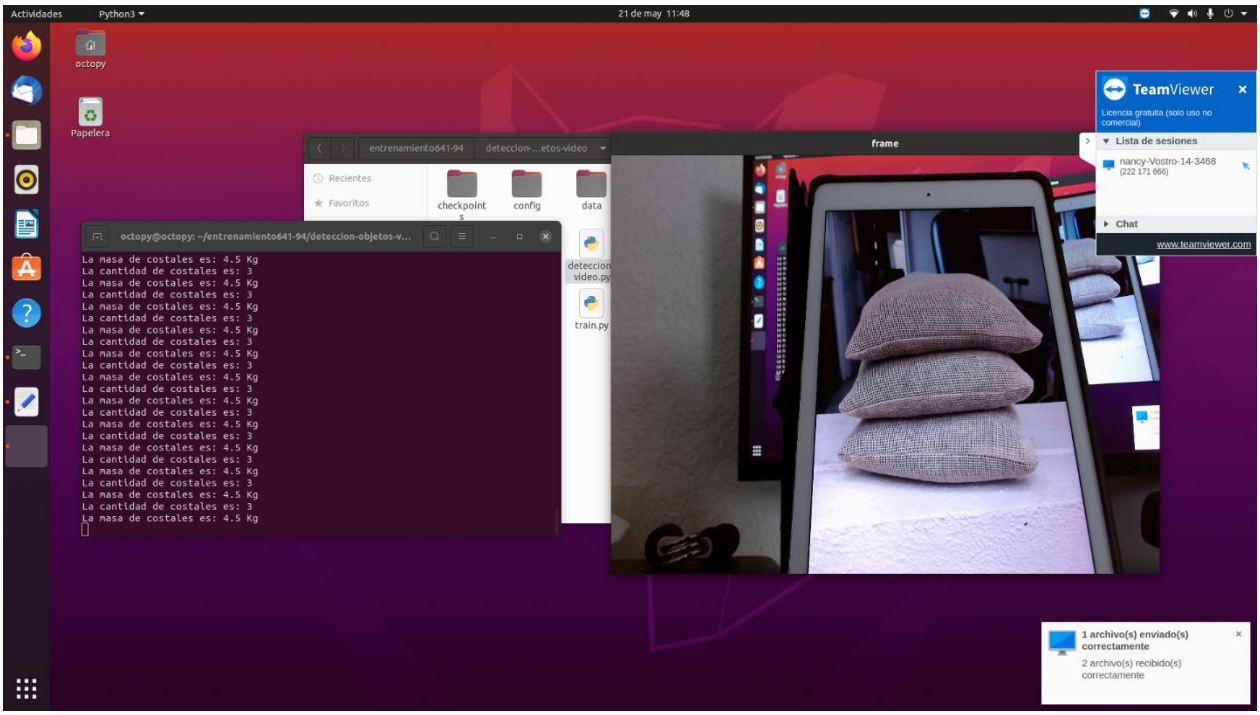


Figura 53. Detección y conteo de costales en NUC - 2

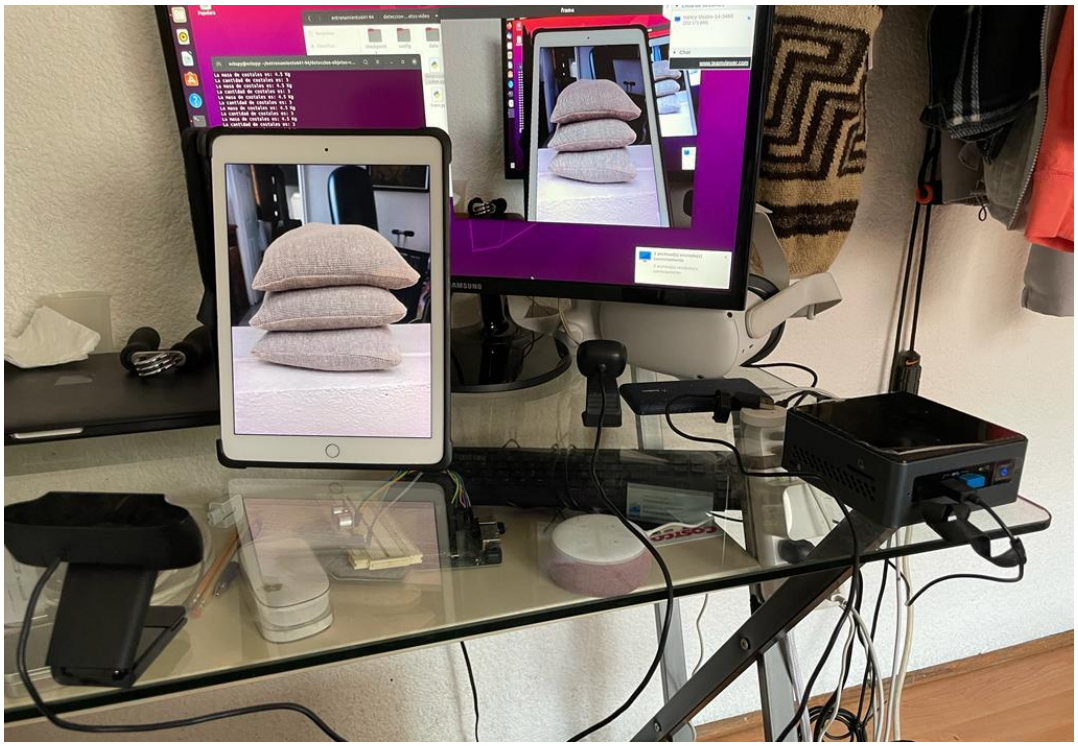


Figura 54. Prueba 2 en NUC- Hardware

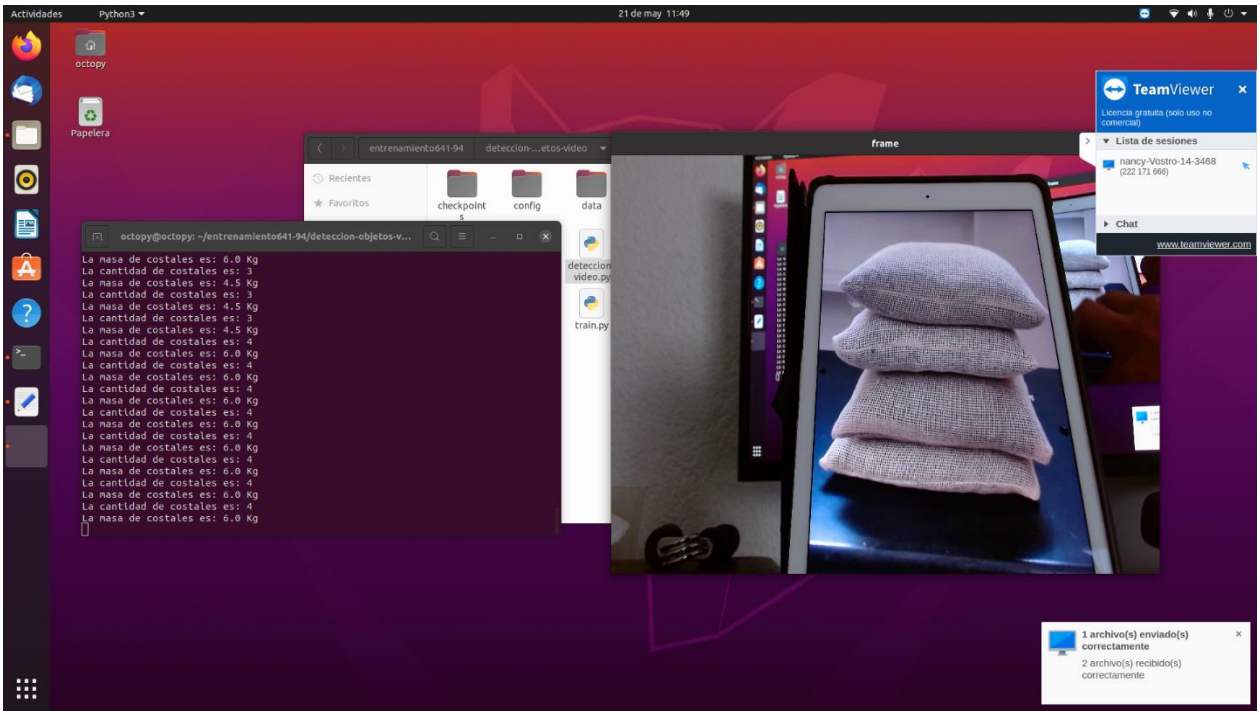


Figura 55. Detección y conteo de costales en NUC - 3

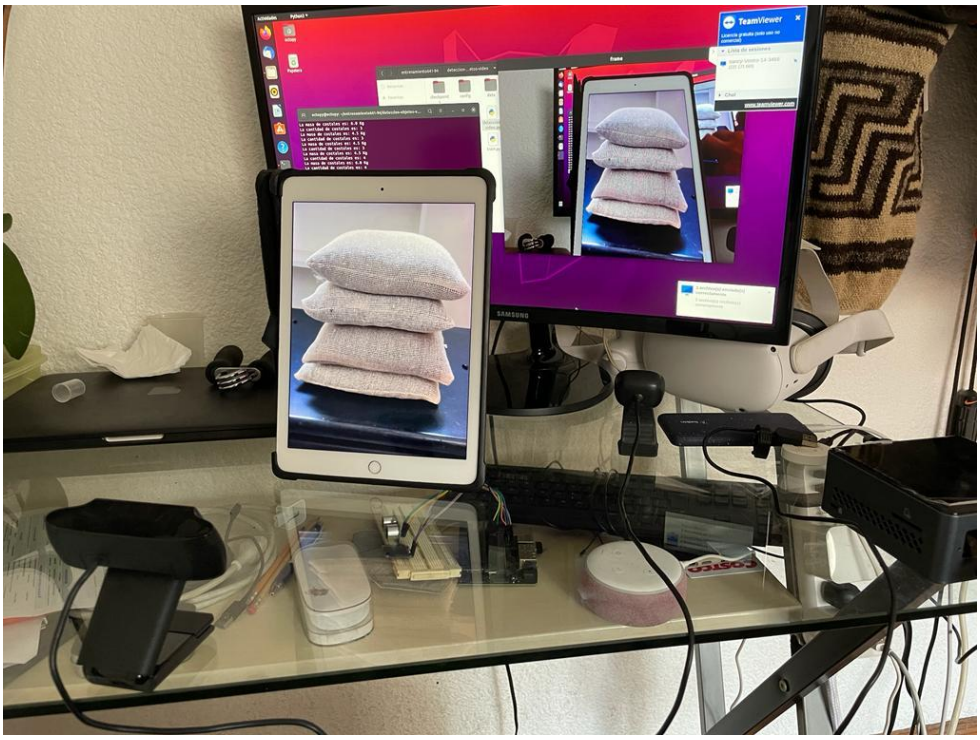


Figura 56. Prueba 3 en NUC – Hardware

En las pruebas se puede observar que, dependiendo de la luz y definición de la imagen, afecta la exactitud de la detección y conteo, como se ve en la siguiente imagen la luz extrema hace que sea difícil para el algoritmo detectar los costales, una posible mejora consiste en ampliar aún más el conjunto de imágenes de entrenamiento, para obtener una mayor exactitud de la detección en una mayor cantidad de luces y calidad de imagen.

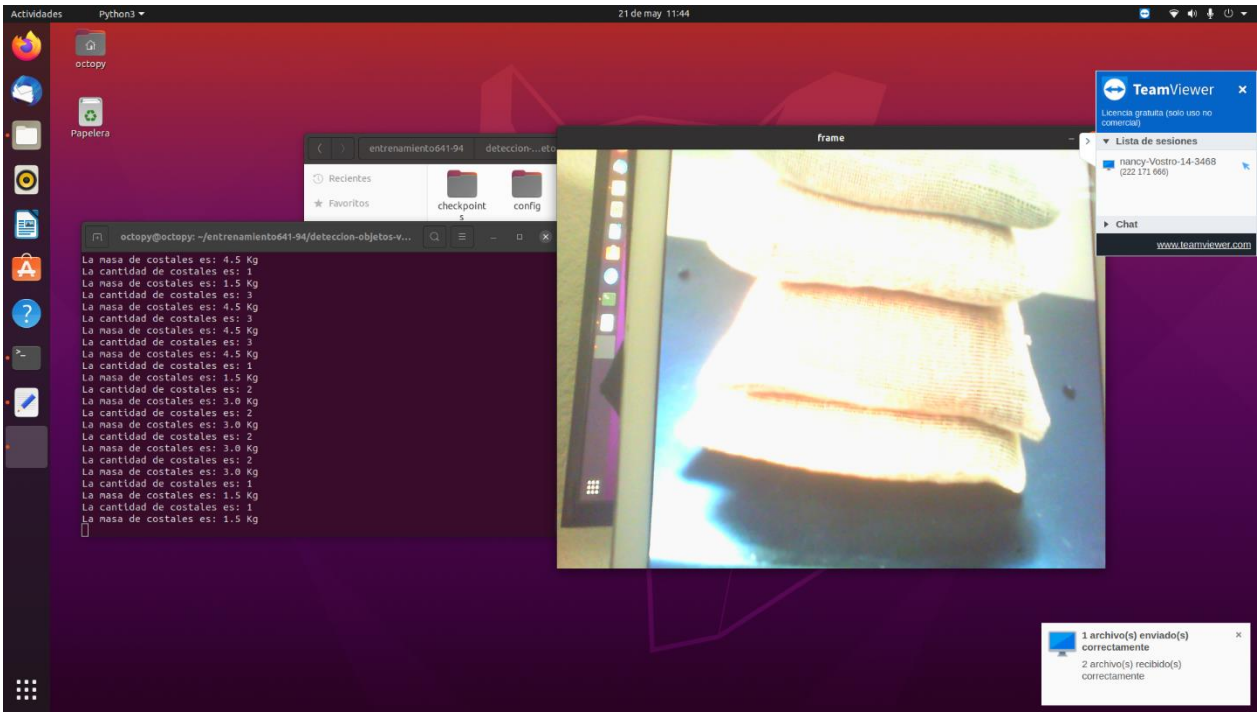


Figura 57. Prueba con mala calidad de imagen y luz

Debido a que la NUC a diferencia de la Jetson Nano no tiene entradas y salidas para interactuar con el exterior, se usará un Arduino Nano para conectar un sensor que detecte la señal del vehículo en el cual irá el dispositivo de conteo de costales, así se enviarán 3 pulsos para activar el funcionamiento del dispositivo como se explicó anteriormente. Además, debido a las diferencias

físicas de la NUC, el diseño de la carcasa se modificó un poco adaptándose a la NUC. En las figuras 58 y 59 se ve como la carcasa almacena la NUC, el Arduino Nano, cámara web y un sensor para detectar la señal del vehículo.

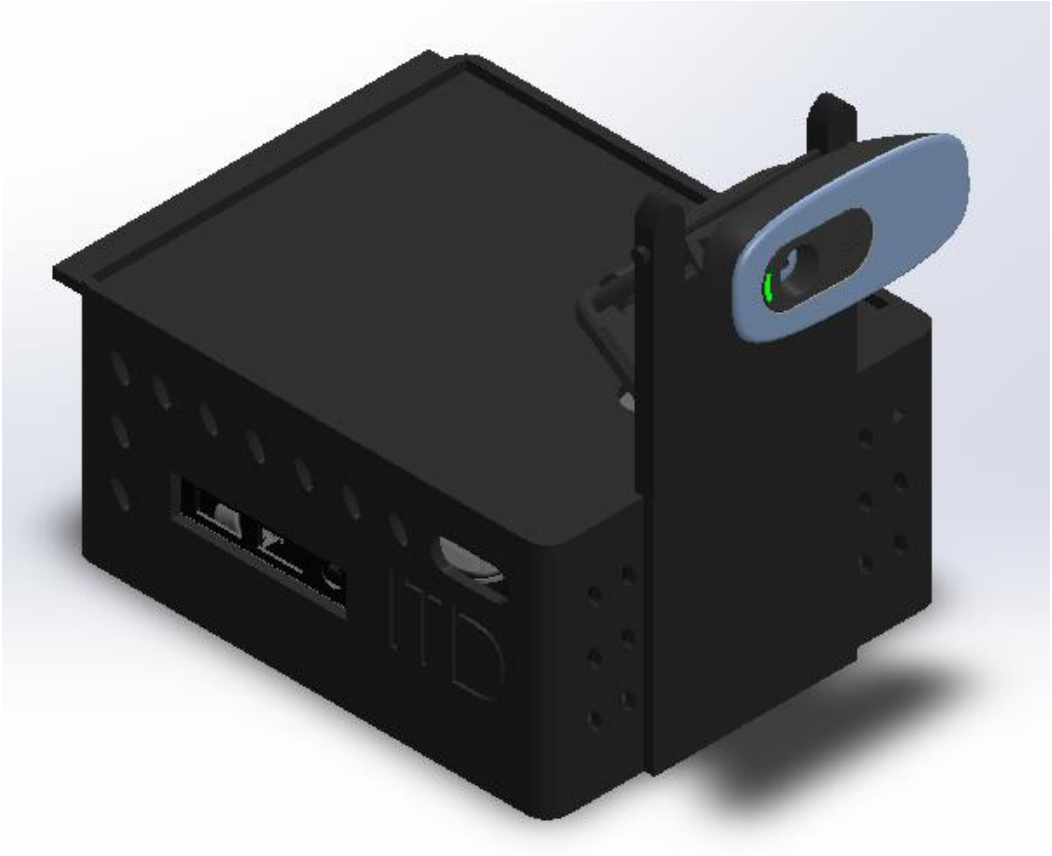


Figura 58. Diseño CAD del dispositivo armado con carcasa modificada

Figura 59. Diseño CAD de dispositivo con NUC y Arduino Nano

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4. Conclusión

La función hecha en Python para calcular el volumen de una estiba completa que sigue el acomodo mencionado, saca el volumen de una estiba de forma exacta, sin embargo, la exactitud de todo el dispositivo depende del modelo de detección de costales, el cual actualmente tiene un F1 score de 0.9555.

Debido a que el programa es muy pesado para la Jetson Nano que se tiene, a pesar de que se amplió la memoria RAM tomando parte de la SD, no obstante, la velocidad de la memoria SD nunca llegará a ser como la de una RAM, por lo que se buscó optimizar el código para que pudiera ser ejecutado en la Jetson Nano, ya que debido a que cambiar de computadora aumentaría mucho el costo del dispositivo y más aún con la crisis actual de silicio que ha afectado a la producción de computadoras tales como raspberry, Jetson Nano y otros componentes, implicando ello que no se tengan muchas opciones en el mercado o que sean muy caras,.

Sin embargo, se logró llevar a cabo una prueba del algoritmo en una NUC por medio *Teamviewrs*, en donde el programa funcionó correctamente, con lo que se obtuvieron referencias de las características que se requieren para el funcionamiento del sistema de detección.

Una vez superada la dificultad anterior, se busca hacer una evaluación estadística más exacta del cálculo de volumen del dispositivo y hacer las mejoras pertinentes según los resultados de dicha evaluación. Este proyecto tiene el potencial para mejorar la logística de un almacén, al agilizar el proceso de cálculo del volumen y la masa de una estiba, con este control de inventario

se podría tener con mayor precisión las ganancias a obtener, en virtud a que se sabe la cantidad que se tiene en el almacén.

4.1. Trabajo futuro y recomendaciones

Actualmente este dispositivo está diseñado específicamente para calcular la masa de una estiba de costales de una sola masa determinada de frijol, por ejemplo, que todos los costales tengan una masa de 60 kilogramos, una mejora futura consiste en colocar una pequeña interfaz para seleccionar el tipo de grano y tamaño de costal, para que de esta manera sea posible calcular la masa de cualquier estiba de costales, verbigracia, en el caso de que las estibas en el almacén estén conformadas por costales de diferentes masas, es decir, la estiba A) está integrada por costales de 80 kilogramos, la estiba B) por costales de 60 kilogramos y la estiba C) por 40 kilogramos, se podrá seleccionar en la interfaz la masa individual que conforma la estiba que se desea calcular.

Otro proyecto a futuro, consistirá en diseñar y desarrollar un método para que el dispositivo pueda calcular la masa de estibas incompletas, ya que la versión actual solo calcula la cantidad de costales y masa de estibas completas. Asimismo, una mejora más sería hacer más robusto el modelo para una mejor precisión del dispositivo.

REFERENCIAS

[1] Vashistha, A., Singhal, A., & Behl, R. (2020). Sack counting system leveraging video analytics.

[2] Flores, D., González-Hernández, I., Lozano, R., Vazquez-Nicolas, J. M., & Hernandez Toral, J. L. (2021). Automated Agave Detection and Counting Using a Convolutional Neural Network and Unmanned Aerial Systems. Drones, 5(1), 4.

[3] Afonso, M., Fonteijn, H., Fiorentin, F. S., Lensink, D., Mooij, M., Faber, N., ... & Wehrens, R. (2020). Tomato fruit detection and counting in greenhouses using deep learning. Frontiers in plant science, 1759.

[4] Lin, Z., & Guo, W. (2020). Sorghum panicle detection and counting using unmanned aerial system images and deep learning. Frontiers in Plant Science, 11, 1346

[5] Li, W., Chen, P., Wang, B., & Xie, C. (2019). Automatic localization and count of agricultural crop pests based on an improved deep learning pipeline. Scientific reports, 9(1), 1-11.

[6] Apolo-Apolo, O. E., Martínez-Guanter, J., Egea, G., Raja, P., & Pérez-Ruiz, M. (2020). Deep learning techniques for estimation of the yield and size of citrus fruits using a UAV. European Journal of Agronomy, 115, 126030.

[7] Liu, X., Chen, S. W., Liu, C., Shivakumar, S. S., Das, J., Taylor, C. J., ... & Kumar, V. (2019). Monocular camera based fruit counting and mapping with semantic data association. IEEE Robotics and Automation Letters, 4(3), 2296-2303.

[8] Chen, S. W., Shivakumar, S. S., Dcunha, S., Das, J., Okon, E., Qu, C., ... & Kumar, V. (2017). Counting apples and oranges with Deep learning: A data-driven approach. IEEE Robotics and Automation Letters, 2(2), 781-788.

[9] Bargoti, S., & Underwood, J. (2017, May). Deep fruit detection in orchards. In 2017 IEEE International Conference on Robotics and Automation (ICRA) (pp. 3626-3633). IEEE.

[10] Xu, R., Li, C., Paterson, A. H., Jiang, Y., Sun, S., & Robertson, J. S. (2018). Aerial images and convolutional neural network for cotton bloom detection. Frontiers in plant science, 8, 2235.

[11] Pérez, R. M., Solano Arias, J., & Méndez Porras, A. (2019). Introducción al Aprendizaje Automático con YOLO. Obtenido de Instituto Tecnológico de Costa Rica: https://revistas.ulatina.ac.cr/index.php/tecnologiavital/article/download/250/260/#:~:text=UTILIZANDO%20YOLOV3&text=Se%20describe%20el%20sistema%20de%20reconocimiento%20de%20objetos%20YOLO%2C%20la,el%20modelo%20entrenado%20en%20YOLOv3.

Rich, E., Knight, K., Calero, P. A. G., & Bodega, F. T. (1994). Inteligencia artificial (Vol. 1). McGraw-Hill.

[12] Anónimo. (s.f). Ubunlog. Obtenido de Ubuntu es un sistema muy popular entre los desarrolladores. ¿Por qué?: https://ubunlog.com/ubuntu-es-popular-para-desarrolladores/

[13] Unipython. (2021). Unipython. Obtenido de ¿Cómo se relaciona la Inteligencia Artificial (IA) con python?: https://unipython.com/como-se-relaciona-la-inteligencia-artificial-ia-con-python/

[14] Rodríguez, H. (2021). Crehana. Obtenido de ¿Qué es OpenCV?: ¡Descubre todo acerca de la visión artificial!: https://www.crehana.com/blog/desarrollo-web/que-es-opencv/

[15] Ciberseguridad. (s.f). ¿Qué es Pytorch? Todo lo que debes saber. Obtenido de Ciberseguridad: https://ciberseguridad.com/guias/nuevas-tecnologias/machine-learning/pytorch/#%C2%BFQue_es_PyTorch

[16] Muñoz, L. (2019). ¿Qué es TensorFlow? Obtenido de el Periódico: https://www.elperiodicodearagon.com/sociedad/2019/06/10/tensorflow-46637663.html

[17] Franklin, D. (2022). NVIDIA Desarrollador. Obtenido de NVIDIA Corporation: https://developer.nvidia.com/blog/jetson-nano-ai-computing/

[18]

[19] Nvidia Corporation. (2022). Nvidia. Obtenido de Jetson Nano: https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/

[20] Intel Corporation. (2022). Intel. Obtenido de Productos Intel NUC: https://www.intel.la/content/www/xl/es/products/details/nuc.html

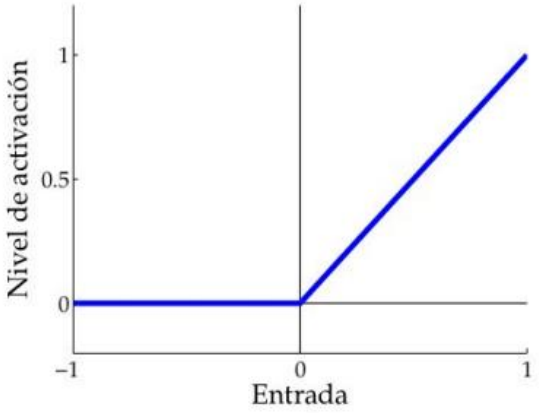
ANEXO

Función de activación

Sirve para deformar la salida lineal de cada capa.

Función lineal rectificada/ReLU

Se usa como función de activación en las capas ocultas de la red neuronal.



$$y = f_{relu}(z) = \begin{cases} z & \text{si } z \leq 0 \\ 0 & \text{si } z < 0 \end{cases} \qquad \frac{d}{dx} f_{relu}(z) = u(z) = \mathbf{1}_{z \geq 0} = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

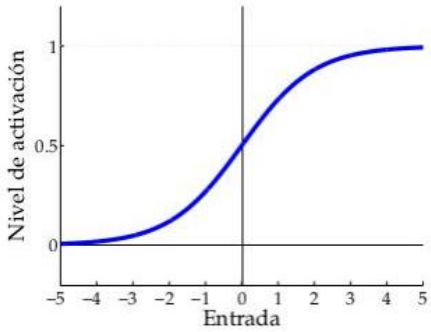
Figura 60. Función ReLU

Función sigmoidal/sigmoid

Se usa como función de activación en la última capa de la red neuronal, es decir, en la capa que realiza la predicción. Comúnmente se usa en problemáticas del tipo:

- Clasificación binaria

- Multiclase o clasificación multietiqueta
- Regresión de valores entre 0 y 1



$$y = f_{logistic}(z) = \sigma(z) = \frac{1}{1+e^{-z}} \qquad \frac{d}{dx}\left(\frac{1}{1+\exp(-x)}\right) = \frac{e^{-x}}{(e^{-x}+1)^2}$$

Figura 61. Función sigmoidal

Métricas de evaluación del modelo

Permiten valorar el rendimiento de un modelo de “Aprendizaje Automático”, con esto se puede estimar la precisión de la generalización del modelo sobre los datos futuros.

Matriz de confusión

Es una representación matricial de los resultados de las predicciones de cualquier prueba binaria, que se usa comúnmente para representar el rendimiento de un modelo de clasificación sobre un conjunto de datos de prueba con valores reales que se conocen.

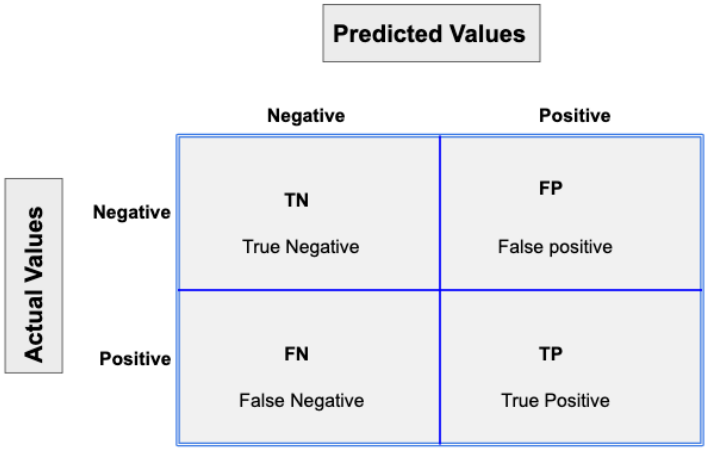


Figura 62. Matriz de confusión con dos etiquetas de clase

Fuente: https://www.datasource.ai/es/data-science-articles/metricas-de-evaluacion-de-modelos-en-el-aprendizaje-automatico

Según como coincida cada predicción con el valor real, podrá haber cuatro posibles resultados de estas:

- Verdadero Positivo (TP): Predicción de verdadero y verdadero en realidad.
- Verdadero Negativo (TN): Predicción de falso y falso en realidad.
- Falso Positivo (FP): Predicción de verdadero y falso en la realidad.
- Falso Negativo (FN): Predicción de falso y verdadero en la realidad.

mAP (mean Average Precision)

La precisión promedio calcula el valor de precisión promedio para el valor de *recall* que va de 0 a 1, entre más cercana a 1 es mejor.

Precisión

Proporción de identificaciones positivas que fueron hechas correctamente y debe ser lo más cercana al 100% para reducir los falsos positivos.

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

Puntuación F1

Es la media armónica de la precisión y exhaustividad, donde la puntuación de F1 alcanza su mejor valor en 1 y peor en 0.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Recall, Sensibilidad o TPR (Tasa de True Positive)

Es la proporción de positivos reales identificados, debe ser lo más cercana al 100% para reducir los falsos negativos.

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

Tensor

En matemáticas y física, un tensor es cierta clase de entidad algebraica de varios componentes, que generaliza los conceptos de escalar vector y matriz, de manera que sea independiente de cualquier sistema de coordenadas. Las dimensiones de los datos son:

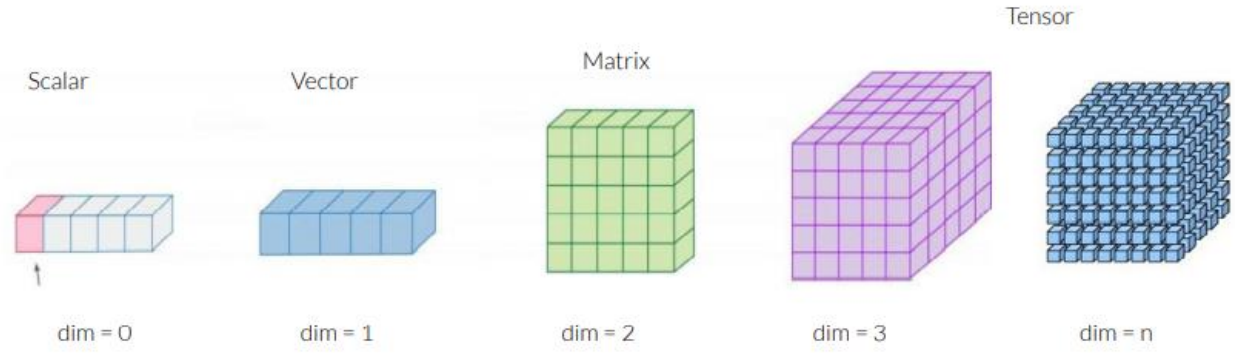


Figura 63. Dimensión de un tensor

Se realizó y publicó un artículo de conferencia basado en esta tesis:

Título del articulo: Sack Detection and Counting Using Deep Learning

Editora: IEEE Conference Publication

Publicado en: 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)

Página: IEE Xplore

Link: https://ieeexplore.ieee.org/document/9872638

DOI: 10.1109/ICECET55527.2022.9872638

ISBN: 978-1-6654-7087-2

Print on Demand(PoD) ISBN: 978-1-6654-7088-9