GUIA DE OBSERVACIÓN PARA EXPOSICIÓN INDIVIDUAL Y/O POR EQUIPO

DOCENTE: Joel Francisco Pava Chipol	ASIGNATURA: TALLER DE INVESTIGACIÓN II
DATOS GENERALES DEL PROCESO DE EVALUACIÓN	
PERIODO: **Agosto-Diciembre 2024**	**UNIDAD:**
TEMA:	**FECHA DE PRESENTACIÓN:**

INSTRUCCIÒN

Revisar los documentos o actividades que se solicitan y marque en los apartados "SI" cuando la evidencia a evaluar se cumple; en caso contrario marque "NO". En la columna "OBSERVACIONES" ocúpela cuando tenga que hacer comentarios referentes a lo observado.

VALOR DEL REACTIVO	CARACTERÍSTICA A CUMPLIR (REACTIVO)	CUMPLE		OBSERVACIONES
		SI	NO	
10%	**Puntualidad**: para iniciar y concluir la exposición.			
10%	**Esquema de diapositiva**. Colores y tamaño de letra apropiada. Sin saturar las diapositivas de texto. **Portada:** Nombre de la escuela (logotipo), Carrera, Asignatura, Profesor, Alumnos, Matricula, Grupo, Lugar y fecha de entrega.			
5%	**Ortografía**: (cero errores ortográficos).			
10%	**Exposición.** a. Utiliza las diapositivas como apoyo, no lectura total			
20%	b. Desarrollo del tema fundamentado y con una secuencia estructurada.			
10%	C. Organización de los integrantes del equipo.			
5%	D. Expresión no verbal (gestos, miradas y lenguaje corporal).			
30%	**Preparación de la exposición.** Dominio del tema. Habla con seguridad.			
100%	**CALIFICACIÓN**			

INTEGRANTES **EQUIPO:_____**

LISTA DE COTEJO DE INVESTIGACION DOCUMENTAL

DOCENTE: Joel Francisco Pava Ch ipol	ASIGNATURA: TALLER DE INVESTIGACION II
PERIODO: **Agosto-Diciembre 2024**	**UNIDAD:**

DATOS GENERALES DEL PROCESO DE EVALUACIÓN

NOMBRE DEL ALUMNO O NUMERO DEL EQUIPO:	
TEMA:	**FECHA DE ENTREGA**:

INSTRUCCIONES

Revisar las actividades que se solicitan y marque en los apartados “SI” cuando la evidencia se cumple; en caso contrario marque “NO”. En la columna “OBSERVACIONES” indicaciones que puedan ayudar al alumno a saber cuáles son las condiciones no cumplidas, si fuese necesario.

VALOR DEL REACTIVO	CARACTERÍSTICA A CUMPLIR (REACTIVO)	CUMPLE		OBSERVACIONES
		SI	NO	
10%	Presentación El trabajo cumple con los requisitos de: a. Buena presentación b. Mismo formato (letra arial 14 para títulos con negritas y contenido arial 12, texto justificado) c. Limpieza y orden d. Ortografía (El documento es redactado de forma correcta sin faltas de ortografía)			
30%	Ideas relevantes: Presenta el contenido más relevante del tema abordado, se centra en la idea principal y compara información de referencias formales de mínimo tres autores.			
10%	Imágenes y gráficos de apoyo: Presenta imágenes, fotografías, tablas, gráficos de apoyo o fórmulas que respalden la información presentada.			
30%	Coherencia y cohesión: Maneja el lenguaje técnico apropiado y presenta en todo el documento coherencia y secuencia entre párrafo.			
10%	Referencias bibliográficas: De fuentes formales y citadas al final del documento de forma correcta.			
10%	Responsabilidad: Entregó el resumen en la fecha y hora señalada.			
100%	CALIFICACIÓN			

EXÁMENES

Nombre del Proyecto:
Integrantes del equipo:
Resumen:
Antecedentes:
Introducción:
Referencias:
Formato ISO, IEEE.

Nombre del Proyecto:

CONTEO DE POLLOS AUTOMATIZADOS MEDIANTE EL USO DE IA Y SOFTWARE "YOLO V3"
Equipo (Integrantes):

- CINTA SEBA JOSUE DAVID
- DEL MORAL CAMACHO JOSE ANTONIO
- FIGUEROA CORRO JUNI ALAN

Resumen:

Antecedentes:

En el trabajo de Huscio (2016)[1], propone la utilización de un ascensor, para transportar las gallinas, asimismo, no solo se puede ocupar para ello, sino también, para transporte de diferentes artículos. El accionamiento del ascensor se basa en un planar mecatrónico; sistema de posicionamiento con rodamiento aerostático. Esta solución original de un ascensor mecatrónico sin cables puede ser utilizado para este mismo fin o diferentes artículos como ya se mencionó. En el mismo año el autor Wei Liu[2] propone un diseño mecánico que se compone de brazos robóticos con un fijo dispositivo de barrido para reacomodar los pollos muertos, ocupando una cinta transportadora que se encargará de trasladar los pollos y una caché de almacenamiento para retener los pollos en un área segura y un vehículo para una mejor automatización en la inteligencia artificial en la industria avícola, basándose en el algoritmo YOLO V4. Ya que el sistema de identificación se utiliza de manera novedosa en el pequeño sistema y las condiciones de los pollos en un gallinero pueden ser o estar monitoreadas remotamente usando una cámara, y los pollos muertos se pueden identificar a través de estas.

El autor Ben Cao (2021)[3] propone un método automatizado para contar pollos de manera más rápida y precisa con el análisis de imágenes RGB capturadas por cámaras de vigilancia en el lugar. Diseñaron de forma innovadora una red convolucional completa, DenseFCN, y para el conteo de los pollos utilizaron imágenes mediante el método de supervisión puntual, que logra una precisión del 93,84% y 9,27 fotogramas por segundo (FPS) de manera que la foto procesada es analizada píxel por píxel. Menciona que a comparación del método de detección de objetos convencional actual, el método de supervisión puntual es más eficaz para medir la densidad de pollo en un entorno de granja. Por su parte el autor Yiqinbao (2021)[4], diseño un sistema para detectar los pollos muertos o vivos en un espacio, esto con la ayuda del software ZigBeee, el cual será el encargado de recolectar todos los datos obtenidos mediante el movimiento del ave, usando una serie de colores mediante un anillo en el pie del pollo, el cual consta de 8 luces; pollo muerto es una luz roja, pollo enfermo una luz amarilla, pollo débil activo con una luz verde y un pollo activo no emite luz.

El autor Neethirajan (2022)[5] propone un modelo de aprendizaje profundo basado en You Only Look Once (Yolo V5) para detectar pollos domésticos a partir de videos con varios fondos complejos. La red Yolo V5 está entrenada y probada en un conjunto de datos, lo que resulta en una mayor precisión de seguimiento y al usar los filtros de Kalman, el modelo propuesto puede rastrear varios pollos simultáneamente al asociar pollos individuales a lo largo de los cuadros de video para aplicaciones en línea y en y en tiempo real.

De acuerdo a las investigaciones de los autores pasados nuestro trabajo de investigación se llevará a cabo un proyecto para el conteo de pollos automatizados mediante el uso de inteligencia artificial y el uso del software “yolo v3” para emplear algoritmos específicos para solo enfocarse en la detección de los pollos, previamente utilizando cámaras en diferentes lugares para realizar el conteo, todo esto disminuyendo la intervención humana y logrando que el trabajo sea más sencillo y rápido de contabilizar en los almacenes que resguardan a los pollos.

INTRODUCCIÓN.

La avicultura o producción avícola se refiere a la práctica de criar aves con un objetivo comercial. Este objetivo puede dividirse en dos grandes ramas: la venta de huevos y la venta de carne. Para las industrias, criaderos, almacenes y granjas de pollos es muy importante tener en cuenta la cantidad de pollos que llegan y los que se mantienen en dicho lugar, ya que, al no contabilizarlos, puede traer pérdidas económicas.

En el mercado hay variedad de herramientas para realizar el conteo, por ejemplo: El Contador de Aves CA-12000 (figura 1) ha sido desarrollado para satisfacer la necesidad de la industria avícola en realizar el recuento de pollos. El recuento es hecho por medio de dos sensores fotoeléctricos que registran la presencia del gancho y del ave simultáneamente [6]. Otro contador es “El contador de pollitos de Petersime” (figura 2) este es una máquina que logra una precisión más alta que el recuento manual. Es por esto que elimina los errores de los empleados de la planta de incubación y funciona más rápido, lo que se traduce en unos pollitos más sanos, ya que tienen que pasar menos tiempo en la sala de extracción [7]. Un ejemplo más seria el contador de pollitos de cinco carriles (figura 3) que cuenta con sensores de alta velocidad y el exclusivo sistema de banda de tres niveles, brindando un manejo suave de los pollitos mientras mantiene una precisión de alto volumen [8].

Figura 1 CA-12000 Contador de Aves

Figura 2 El contador de pollitos de Petersime.

Figura 3 KL AUTOMATION | CONTADOR DE POLLITOS DE CINCO CARRILES.

Para entender más sobre el tema, a continuación, se muestra la implementación de un conteo de pollos en una empresa de Colombia llamada “Pasto-Nariño” de POLLOS FRESCOS DE COLOMBIA LIMITADA (POFRESCOL LTDA) Se dedica a la cría de aves de corral, elaboración de alimentos preparados para animales y comercio al por mayor de productos alimenticios, cuenta con una estructura o línea de proceso muy bien elaborada que va desde una etapa de recepción de pollos hasta la etapa de despacho.

Diariamente a la planta ingresan alrededor de 12.000 pollos, el operario debe controlar variables de gran importancia en el proceso que dificultan un desarrollo óptimo a la hora de desempeñar su trabajo, estas variables pueden ser, tener desperdicios de tiempo, mala utilización de los equipos, agotamiento del operario o simplemente el mismo error humano no garantiza un registro ciento por ciento eficaz. Es por esta razón implementaron sistema automático de reconocimiento y conteo de pollos en la parte de recepción como al momento de su despacho, implementando tecnología electrónica por computadora, que minimice ese déficit

económico mensual, dentro del proceso de producción para el comercio y venta del producto elevando la calidad de estas etapas en la planta POFRESCOL LTDA [9].

Dando a lo anteriormente mencionado se necesita de una contabilidad y esto se logra mediante los inventarios porque en el sector es importante conocer el costo unitario de las aves avícolas. La tarea de contabilizar a los pollos puede llegar a ser muy tardado y cansado para los operadores que realizan este tipo de trabajos, por tanto, el proyecto que se propone es el crear un sistema automatizado de contabilidad mediante el software YOLOv3 y también dando paso a la Inteligencia Artificial, en conjunto con la utilización de cámaras, basándonos en autores que anteriormente hicieron algo parecido.

Metodología:

El proyecto se enfoca en el Depósito de pollo " La Victoria", donde se desarrollará, en la figura 4 se muestra parte de la construcción del depósito.

Figura 4 Depósito de pollo " La Victoria".

El tipo de investigación pertenece a innovación, optimización y desarrollo tecnológico. Para el desarrollo exitoso del proyecto se planteó una metodología la cual se divide en 3 fases.

Fase 1.

Realizar prototipo del sistema de reconocimiento y conteo automático de pollos usado en la empresa de pollos “La Victoria”. A continuación, se muestran las herramientas principalesque se necesitan para crear el sistema.

- YOLOv3 es una red neuronal convolucional (CNN) para realizar la detección de objetos en tiempo real. Las CNN son sistemas basados en clasificadores que pueden procesar imágenes de entrada como conjuntos estructurados de datos y reconocer patrones entre ellos (ver imagen a continuación). YOLO tiene la ventaja de ser mucho más rápido que otras redes y aún mantiene la precisión. Permite que el modelo mire la imagen completa en el momento de la prueba, por lo que sus predicciones se basan en el contexto global de la imagen. YOLO y otros algoritmos de redes neuronales convolucionales "puntúan" regiones en función de sus similitudes con clases predefinidas. Las regiones de puntuación alta se notan como detecciones positivas de cualquier clase con la que se identifiquen más de cerca. Por ejemplo, en una transmisión en vivo de tráfico, YOLO se puede usar para detectar diferentes tipos de vehículos según las regiones del video que obtengan una puntuación alta en comparación con las clases predefinidas de vehículos, ver figura 5.

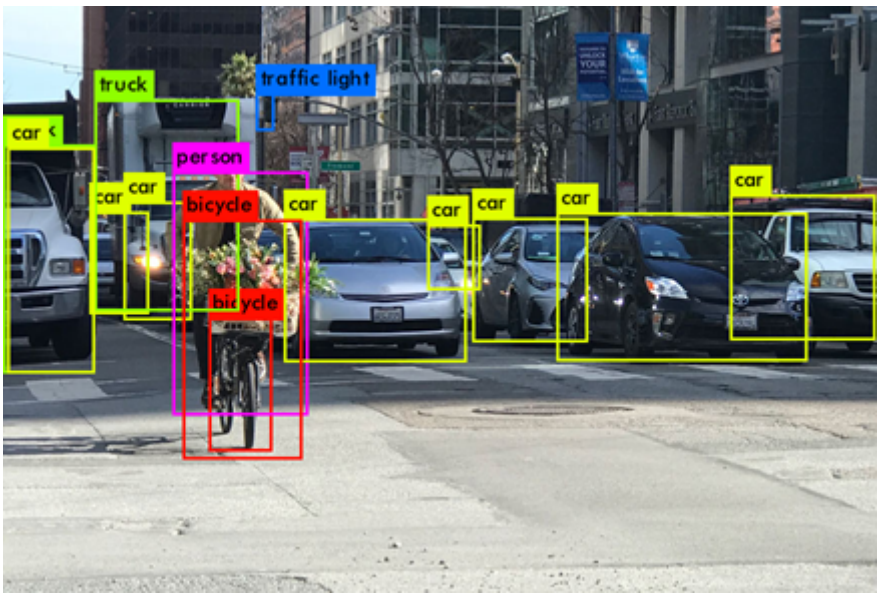

Figura 5 Ejemplo de visión artificial de YOLOv3.

- CPU (Unidad Central de Procesamiento): Se recomienda utilizar una CPU potente para ejecutar el código de YOLO V3. Las CPUs modernas de gama alta, como los procesadores Intel Core i7 o i9, o los equivalentes de AMD, proporcionan un rendimiento sólido para tareas de visión por computadora.

- GPU (Unidad de Procesamiento Gráfico): La aceleración por GPU es esencial para una ejecución rápida y eficiente de YOLO V3. Se recomienda utilizar una GPU compatible con CUDA (Compute Unified Device Architecture) de NVIDIA, como las series NVIDIA GeForce RTX o NVIDIA Quadro. EstasGPUs ofrecen un procesamiento paralelo masivo que mejora significativamente el rendimiento de la detección de objetos.

- Memoria RAM: Se necesita suficiente memoria RAM para cargar el modelo y procesar las imágenes de entrada. Se recomienda tener al menos 8 GB de RAM, aunque para un rendimiento óptimo, especialmente al trabajar con grandes conjuntos de datos o videos de alta resolución, se recomienda disponer de 16 GB o más.

- Almacenamiento: Se requiere espacio de almacenamiento adecuado para el software YOLO V3, el modelo entrenado y los conjuntos de datos. Además, asegúrese de tener suficiente espacio libre para almacenar las imágenes o videos de entrada y los resultados del conteo. Un disco duro de estado sólido (SSD) es preferible debido a su mayor velocidad de lectura/escritura en comparación con los discos duros convencionales (HDD).

- Sistema operativo: YOLO V3 es compatible con varios sistemas operativos, incluyendo Windows, Linux y macOS. Elija el sistema operativo que mejor se adapte a sus necesidades y que sea compatible con las bibliotecas y dependencias requeridas por YOLO V3.

- Cámaras de vigilancia: Las cámaras de vigilancia son ampliamente utilizadas para aplicaciones de seguridad. Marcas como Arlo, Nest y Ring ofrecen cámaras de vigilancia que se pueden utilizar con YOLO v3. Así que cualquiera de las marcas anteriores se puede utilizar en este proyecto.

Fase 2.

Desarrollar sistema de visión artificial para el conteo y reconocimiento de pollos. En esta fase se investigará el modelo de reconocimiento de objetos más óptimo para implementarlo, en esta fase se tratará de reducir la deformación de las fotos capturadas con la cámara para realizar los respectivos recortes en la imagen con la mayor exactitud posible con la finalidad de obtener solo el espacio de trabajo en el que se realiza el conteo de los pollos, además se realizará pruebas de eficacia del sistema para reconocer los pollos sobre el área donde se encuentran alojados.

A continuación, se muestra la serie de pasos de la detección de objetos YOLOv3 con Keras.

Paso 1: Salta al primer paso, las siguientes son las bibliotecas y dependencias necesarias, ver figura 6.

```
import os
import scipy.io
import scipy.misc
import numpy as np
import pandas as pd
import PIL
import struct
import cv2
from numpy import expand_dims
import tensorflow as tf
from skimage.transform import resize
from keras import backend as K
from keras.layers import Input, Lambda, Conv2D, BatchNormalization, LeakyReLU, ZeroPadding2D, UpSampling2D
from keras.models import load_model, Model
from keras.layers.merge import add, concatenate
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from matplotlib import pyplot
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from matplotlib.patches import Rectangle
%matplotlib inline
```

Figura 6 Código de las bibliotecas y dependencias.

Paso 2: A continuación, la **WeightReader** clase se utiliza para analizar el archivo "yolov3. pesos” y cargue los pesos del modelo en la memoria, ver figura 7.

```
clase WeightReader:
        def __init__(self, peso_archivo):
                con open(weight_file, 'rb') como w_f:
                        mayor, = struct.unpack('i', w_f.read(4))
                        menor, = struct.unpack('i', w_f.read(4))
                        revisión, = struct.unpack('i', w_f.read(4))
                        si (mayor*10 + menor) >= 2 y mayor < 1000 y menor < 1000:
                                w_f.leer(8)
                        demás:
                                w_f.leer(4)
                        transponer = (mayor > 1000) o (menor > 1000)
                        binario = w_f.leer()
                auto.desplazamiento = 0
                self.all_weights = np.frombuffer(binario, dtype='float32')

        def read_bytes(uno mismo, tamaño):
                self.desplazamiento = self.desplazamiento + tamaño
                devuelve self.all_weights[self.offset-size:self.offset]

        def load_weights(self, modelo):
                para i en el rango (106):
                        intentar:
                                conv_layer = model.get_layer('conv_' + str(i))
                                print("Cargando pesos de convolución #" + str(i))
                                si no estoy en [81, 93, 105]:
                                        capa_norma = modelo.get_capa('bnorm_' + str(i))
                                        tamaño = np.prod(norm_layer.get_weights()[0].shape)
                                        beta = self.read_bytes(tamaño) # sesgo
                                        gamma = self.read_bytes(tamaño) # escala
                                        media = self.read_bytes(tamaño) # media
                                        var = self.read_bytes(tamaño) # varianza
                                        pesos = norm_layer.set_weights([gamma, beta, mean, var])
                                si len(conv_layer.get_weights()) > 1:
                                        sesgo = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
                                        kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                                        kernel = kernel.reshape(lista(invertida(conv_layer.get_weights()[0].shape)))
                                        kernel = kernel.transponer([2,3,1,0])
                                        conv_layer.set_weights([núcleo, sesgo])
                                demás:
                                        kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                                        kernel = kernel.reshape(lista(invertida(conv_layer.get_weights()[0].shape)))
                                        kernel = kernel.transponer([2,3,1,0])
                                        conv_layer.set_weights([núcleo])
                        excepto ValueError:
                                print("sin convolución #" + str(i))

        def restablecer (auto):
                auto.desplazamiento = 0
```

Figura 7 WeightReader.

Paso 3: YOLO v3 está utilizando una nueva red para realizar la extracción de características que es innegablemente más grande en comparación con YOLO v2. Esta red se conoce como Darknet-53 ya que toda la red se compone de 53 capas convolucionales con conexiones de acceso directo, ver figura 8 y 9.

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

Figura 8 La red YOLO v3 tiene 53 capas convolucionales.

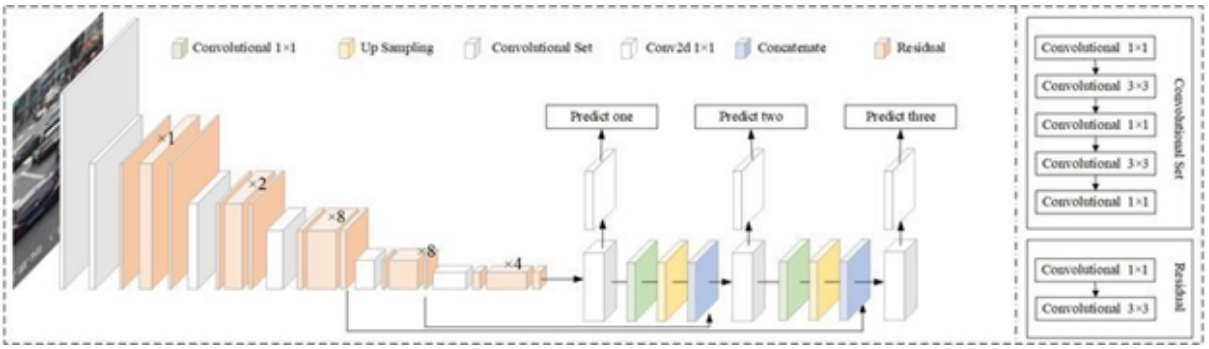

Figura 9 Funcionamiento de YOLOv3

Por lo tanto, el siguiente código (figura 10) se compone de varios componentes que son:

- **_conv_block**función que se utiliza para construir una capa convolucional
- **make_yolov3_model**función que se utiliza para crear capas de convolucional y apilar juntos como un todo.

```
def _conv_block(inp, convs, skip=True):
        x = entrada
        cuenta = 0
        para conv en conv:
                if count == (len(convs) - 2) y omitir:
                        omitir_conexión = x
                contar += 1
                if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x) # relleno peculiar ya que darknet prefiere izquierda y arriba
                x = Conv2D(conv['filtro'],
                                conv['núcleo'],
                                zancadas=conv['zancada'],
                                padding='valid' if conv['stride'] > 1 else 'same', # relleno peculiar ya que darknet prefiere izquierda y arriba
                                nombre='conv_' + str(conv['layer_idx']),
                                use_bias=False if conv['bnorm'] else True)(x)
                if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' + str(conv['layer_idx']))(x)
                if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))(x)
        return add([skip_connection, x]) if skip else x

def make_yolov3_model():
        input_image = Entrada (forma = (Ninguno, Ninguno, 3))
        # Capa 0 => 4
        x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                                                {'filtro': 64, 'núcleo': 3, 'zancada': 2, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 1},
                                                {'filtro': 32, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 2},
                                                {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])
        # Capa 5 => 8
        x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                                        {'filtro': 64, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 6},
                                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])
        # Capa 9 => 11
        x = _conv_block(x, [{'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                                        {'filtro': 128, 'núcleo': 3, 'zancada': 1, 'bnorm': verdadero, 'fugas': verdadero, 'layer_idx': 10}])
        # Capa 12 => 15
        x = _conv_block(x, [{'filtro': 256, 'núcleo': 3, 'paso': 2, 'bnorm': Verdadero, 'fugas': Verdadero, 'capa_idx': 12},
                                        {'filtro': 128, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 13},
                                        {'filtro': 256, 'núcleo': 3, 'zancada': 1, 'bnorm': verdadero, 'fugas': verdadero, 'layer_idx': 14}])
        # Capa 16 => 36
        para i en el rango (7):
                x = _conv_block(x, [{'filtro': 128, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'capa_idx': 16+i*3},
                                                {'filtro': 256, 'núcleo': 3, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 17+i*3}])
```

```
    skip_36 = x
    # Capa 37 => 40
    x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                                        {'filtro': 256, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 38},
                                        {'filtro': 512, 'núcleo': 3, 'zancada': 1, 'bnorm': verdadero, 'fugas': verdadero, 'layer_idx': 39}])
    # Capa 41 => 61
    para i en el rango (7):
            x = _conv_block(x, [{'filtro': 256, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'capa_idx': 41+i*3},
                                                {'filtro': 512, 'núcleo': 3, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'capa_idx': 42+i*3}])
    skip_61 = x
    # Capa 62 => 65
    x = _conv_block(x, [{'filtro': 1024, 'núcleo': 3, 'zancada': 2, 'bnorm': Verdadero, 'fugas': Verdadero, 'capa_idx': 62},
                                        {'filtro': 512, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 63},
                                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 64}])
    # Capa 66 => 74
    para i en el rango (3):
            x = _conv_block(x, [{'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 66+i*3},
                                                {'filtro': 1024, 'núcleo': 3, 'zancada': 1, 'bnorm': verdadero, 'fugas': verdadero, 'layer_idx': 67+i*3}])
    # Capa 75 => 79
    x = _conv_block(x, [{'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 75},
                                        {'filtro': 1024, 'núcleo': 3, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 76},
                                        {'filtro': 512, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 77},
                                        {'filtro': 1024, 'núcleo': 3, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 78},
                                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 79}], skip=False)
    # Capa 80 => 82
    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 80},
                                                {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': Falso, 'leaky': Falso, 'layer_idx': 81}], skip=False)
    # Capa 83 => 86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 84}], skip=False)
    x = UpSampling2D(2)(x)
    x = concatenar ([x, skip_61])
    # Capa 87 => 91
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 87},
                                        {'filtro': 512, 'núcleo': 3, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 88},
                                        {'filtro': 256, 'núcleo': 1, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 89},
                                        {'filtro': 512, 'núcleo': 3, 'zancada': 1, 'bnorm': Verdadero, 'fugas': Verdadero, 'layer_idx': 90},
                                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 91}], skip=False)
    # Capa 92 => 94
    yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 92},
                                                {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': Falso, 'leaky': Falso, 'layer_idx': 93}], skip=False)
```

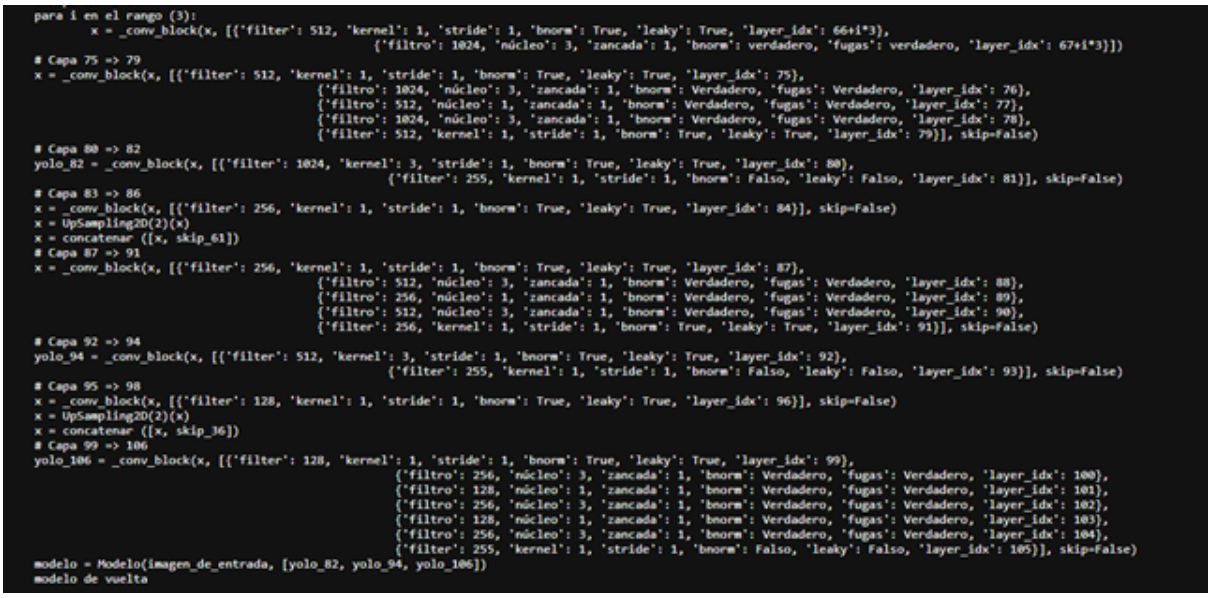

Figura 10 _conv_block y make_yolov3_model.

Etapa 4: A continuación, el siguiente código (figura 11) se explica a continuación:

- Definir el modelo YOLO v3
- Cargue los pesos preentrenados que ha descargado anteriormente
- Guarde el modelo usando savela función Keras y especificando el nombre del archivo

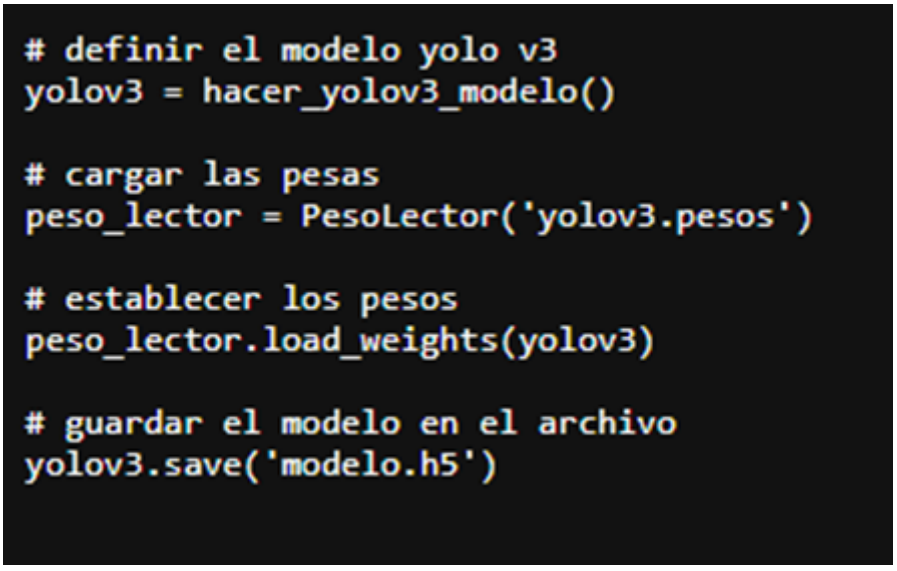

Figura 11 Definir el modelo YOLO v3.

Etapa 4:

Este paso consiste en decodificar el resultado de la predicción en cuadros delimitadores, ver figura 12.

El resultado de la predicción de YOLO v3 tiene la forma de una lista de matrices que difícilmente se pueden interpretar. Como YOLO v3 es una detección multiescala, se decodifica en tres escalas diferentes en forma de (13, 13, 225), (26, 26, 225) y (52, 52, 225)

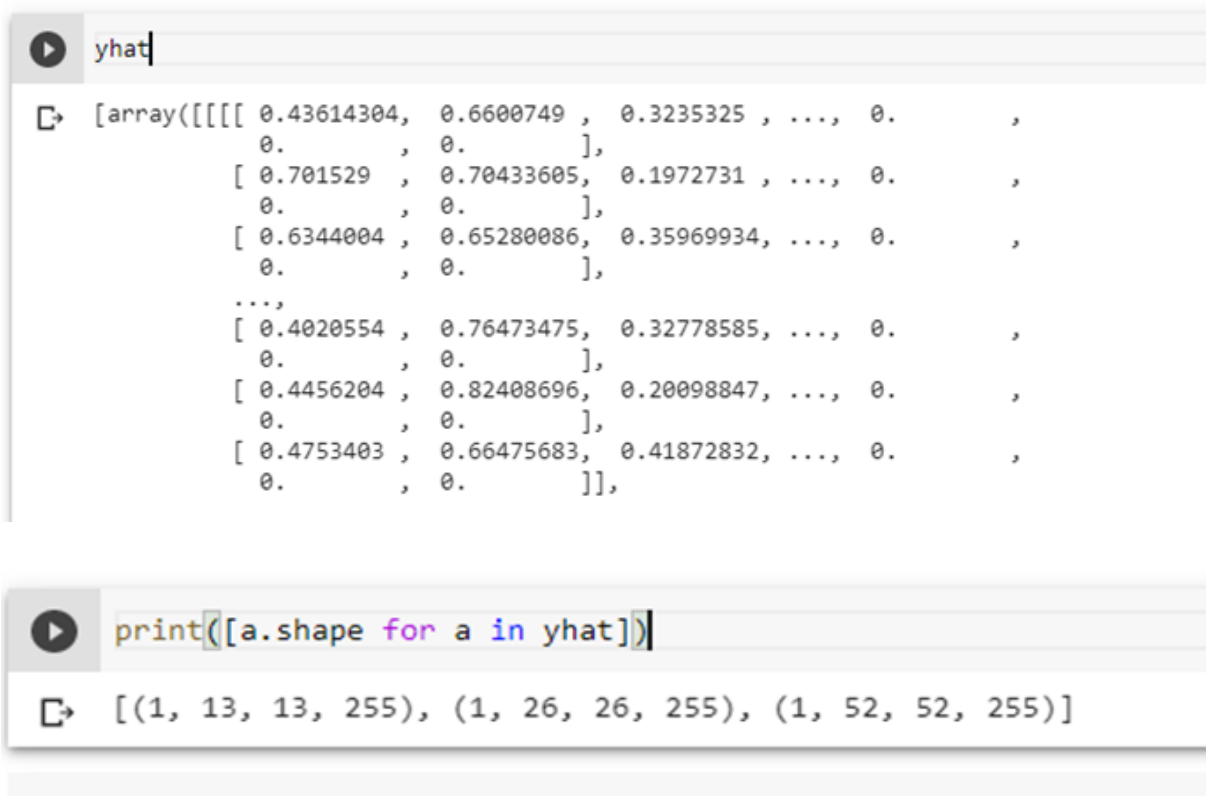

Figura 12 Una porción de la salida de predicción de YOLOv3 antes de que se decodifique.

- **decode_netout:** La función se utiliza para decodificar la salida de predicción en cuadros, ver figura 13.

```
clase BoundBox:
  def __init__(self, xmin, ymin, xmax, ymax, objness = Ninguno, clases = Ninguno):
    self.xmin = xmin
    self.ymin = ymin
    self.xmax = xmax
    self.ymax = ymax
    self.objness = objness
    self.classes = clases
    self.etiqueta = -1
    puntuación propia = -1

  def get_label(auto):
    si auto.etiqueta == -1:
      self.etiqueta = np.argmax(self.clases)

    volver self.label

  def get_score(self):
    si self.score == -1:
      self.score = self.classes[self.get_label()]
    devolver self.get_score

def _sigmoide(x):
  volver 1. /(1. + np.exp(-x))

def decode_netout(netout, anclas, obj_thresh, net_h, net_w):
        grid_h, grid_w = netout.shape[:2]
        caja_nb = 3
        netout = netout.reshape((grid_h, grid_w, nb_box, -1))
        nb_class = netout.shape[-1] - 5
        cajas = []
        salida neta[..., :2] = _sigmoid(salidaneta[..., :2])
        salida neta[..., 4:] = _sigmoid(salida neta[..., 4:])
        salida neta[..., 5:] = salida neta[..., 4][..., np.nuevoeje] * salida neta[..., 5:]
        salida neta[..., 5:] *= salida neta[..., 5:] > obj_thresh

        para i en el rango (grid_h*grid_w):
                fila = i / grilla_w
                col = i % grid_w
                para b en el rango (nb_box):
                        # 4º elemento es la puntuación de objetividad
                        objetividad = netout[int(fila)][int(col)][b][4]
                        if(objectness.all() <= obj_thresh): continuar
                        # los primeros 4 elementos son x, y, w y h
                        x, y, w, h = netout[int(fila)][int(col)][b][:4]
                        x = (col + x) / grid_w # posición central, unidad: ancho de la imagen
                        y = (fila + y) / grid_h # posición central, unidad: altura de la imagen
                        w = anclas [2 * b + 0] * np.exp (w) / net_w # unidad: ancho de imagen
                        h = anclas [2 * b + 1] * np.exp (h) / net_h # unidad: altura de la imagen
                        # los últimos elementos son probabilidades de clase
                        clases = netout[int(fila)][col][b][5:]
                        box = BoundBox(xw/2, yh/2, x+w/2, y+h/2, objetividad, clases)
                        cajas.append(caja)
        cajas de devolución
```

Figura 13 decode_netout

En pocas palabras, así es como **decode_netout** funciona, la función se muestra a continuación, en la figura 14:

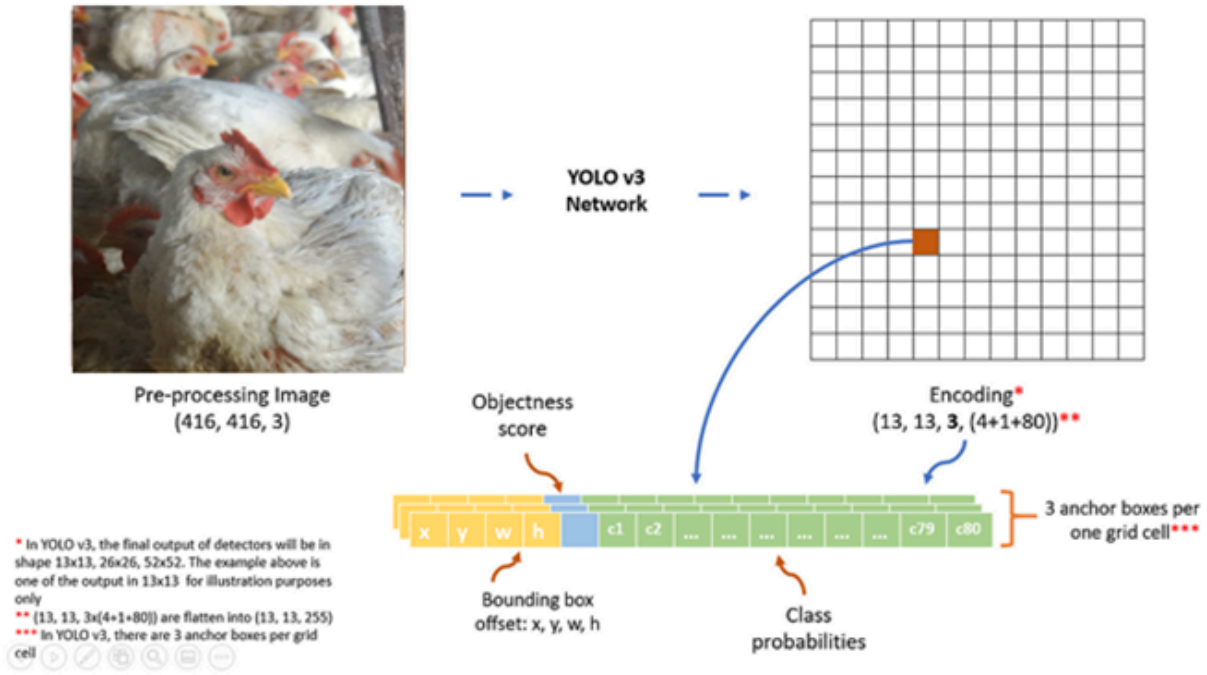

Figura 14 Funcionamiento de decode_netout.

Paso 5: Escale y estire los cuadros decodificados para que encajen en la forma de la imagen original, ver figura 15.

```
def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
        nuevo_w, nuevo_h = net_w, net_h
        para i en el rango (len (cajas)):
                x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
                y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
                cajas[i].xmin = int((cajas[i].xmin - x_offset) / x_scale * image_w)
                cajas[i].xmax = int((cajas[i].xmax - x_offset) / x_scale * image_w)
                cajas[i].ymin = int((cajas[i].ymin - y_offset) / escala_y * imagen_h)
                cajas[i].ymax = int((cajas[i].ymax - y_offset) / y_scale * image_h)
```

Figura 15 Cuadros decodificados.

Paso 6: después de que se haya decodificado la salida de la red, no mostrará una predicción fluida directamente al objeto. Las cajas decodificadoras dieron como resultado varias cajas superpuestas. Como se puede ver en la figura 16, el modelo ha detectado que hay tres pollos en la imagen. Sin embargo, se trata simplemente de cajas superpuestas sobre un objeto, que en este caso son los pollos.

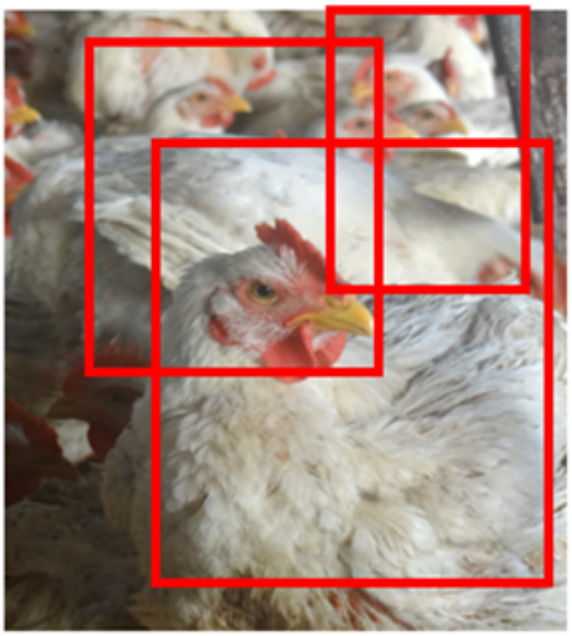

Figura 16 Localización de los pollos.

Por lo tanto, la supresión no máxima (NMS) tiene lugar para filtrar y obtener las casillas correctas, ver figura 17.

- **bbox_iou**La función se usa para calcular el IOU (intersección sobre unión) al obtener el valor **_interval_overlap**de dos cajas

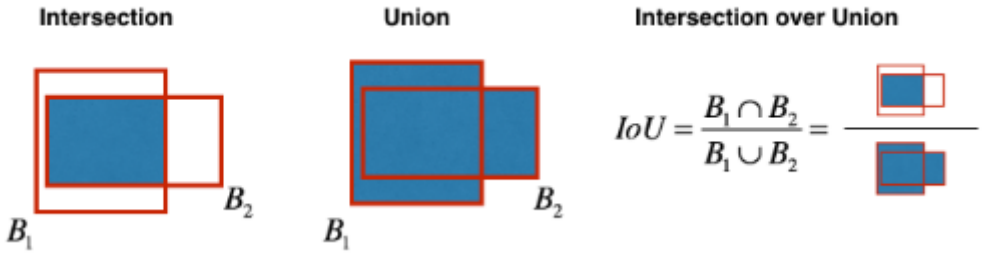

Figura 17 Cálculo del IOU (intersección sobre unión).

- **do_nms** La función se utiliza para realizar NMS.

En general, así es como se realiza NMS:

1. Seleccione la casilla que tenga la puntuación más alta.
2. Calcule la superposición de intervalos de este cuadro con otros cuadros y omita los cuadros que se superponen significativamente (iou >= iou_threshold).

3. Repita hasta el paso 1 e itere hasta que no haya más casillas con una puntuación más baja que la casilla seleccionada actualmente.

Esto omitirá todos los cuadros que tengan una gran superposición con los cuadros seleccionados. Solo queda la casilla "correcta", ver figura 18.

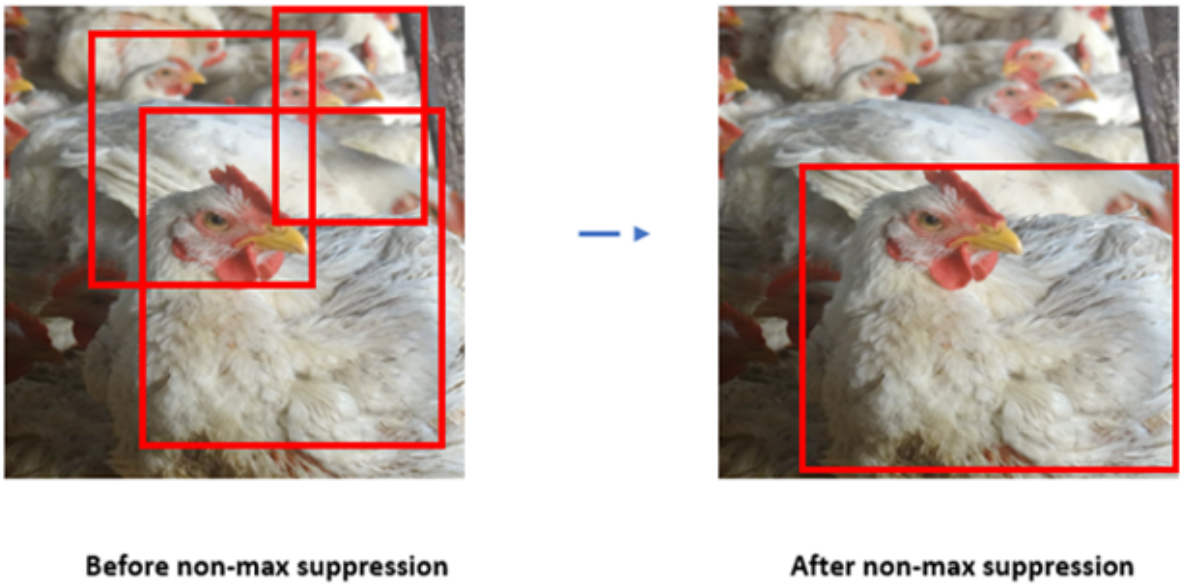

Figura 18 Max suppression.

- **get_boxes**La función se utiliza para obtener las casillas que han sido seleccionadas a través del filtro NMS.
- **draw_boxes**La función se usa para dibujar un cuadro rectangular en la imagen de entrada usando matplotlib.patches.Rectanglela clase.

```
def _interval_overlap(intervalo_a, intervalo_b):
        x1, x2 = intervalo_a
        x3, x4 = intervalo_b
        si x3 < x1:
                si x4 < x1:
                        volver 0
                demás:
                        retorno mínimo(x2,x4) - x1
        demás:
                si x2 < x3:
                         volver 0
                demás:
                        retorno mínimo(x2,x4) - x3

def bbox_iou(caja1, caja2):
        intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
        intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
        intersección = intersección_w * intersección_h
        w1, h1 = caja1.xmax-caja1.xmin, caja1.ymax-caja1.ymin
        w2, h2 = caja2.xmax-caja2.xmin, caja2.ymax-caja2.ymin
        unión = w1*h1 + w2*h2 - intersección
        flotador de retorno (intersección) / unión

def do_nms(cajas, nms_thresh):
        si len(cajas) > 0:
                nb_class = len(cajas[0].clases)
        demás:
                devolver
        para c en el rango (nb_class):
                sorted_indices = np.argsort([-box.classes[c] for box in boxes])
                for i in range(len(sorted_indices)):
                        índice_i = índices_ordenados[i]
                        if boxes[index_i].classes[c] == 0: continuar
                        para j en el rango (i + 1, len (índices ordenados)):
                                índice_j = índices_ordenados[j]
                                si bbox_iou(cajas[índice_i], cajas[índice_j]) >= nms_umbral:
                                        cajas[índice_j].clases[c] = 0

# obtener todos los resultados por encima de un umbral
def get_boxes(cajas, etiquetas, umbral):
        v_boxes, v_labels, v_scores = lista(), lista(), lista()
        # enumerar todas las cajas
        para caja en cajas:
                # enumerar todas las etiquetas posibles
                para i en el rango (len (etiquetas)):
                        # verificar si el umbral para esta etiqueta es lo suficientemente alto
                        if box.classes[i] > trillar:
                                v_boxes.append(caja)
                                v_etiquetas.append(etiquetas[i])
                                v_scores.append(caja.clases[i]*100)
                                # no se rompa, muchas etiquetas pueden activarse para una caja
        devolver v_boxes, v_labels, v_scores

# dibujar todos los resultados
def draw_boxes(nombre de archivo, v_boxes, v_labels, v_scores):

        # cargar la imagen
        datos = pyplot.imread(nombre de archivo)
        #trazar la imagen
        pyplot.imshow(datos)
        # obtener el contexto para dibujar cuadros
        hacha = pyplot.gca()
        # graficar cada cuadro
        para i en el rango (len (v_boxes)):
                caja = v_cajas[i]
                # obtener coordenadas
                y1, x1, y2, x2 = caja.ymin, caja.xmin, caja.ymax, caja.xmax
                # calcular ancho y alto de la caja
                ancho, alto = x2 - x1, y2 - y1
                # crear la forma
                rect = Rectángulo ((x1, y1), ancho, alto, relleno = Falso, color = 'amarillo', ancho de línea = '2')
                # dibujar la caja
                hacha.add_patch(rect)
                # dibujar texto y puntuación en la esquina superior izquierda
                etiqueta = "%s (%.3f)" % (v_etiquetas[i], v_puntuaciones[i])
                pyplot.text(x1, y1, etiqueta, color='amarillo')
        # mostrar la trama
        pyplot.mostrar()
```

Figura 19 Código de la supresión no máxima (NMS)

Paso 7: Estamos cerca del final de la implementación de YOLO v3. En este paso, tenemos que declarar varias configuraciones (figura 20) que son:

- **anchors:** cuidadosamente elegido en base a un análisis del tamaño de los objetos en el conjunto de datos COCO.
- **class_threshold**: el umbral de probabilidad para los objetos detectados
- **labels**: etiquetas de clase del conjunto de datos COCO

```
# definir los anclajes
anclas = [[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]

# definir el umbral de probabilidad para los objetos detectados
clase_umbral = 0.6

# definir las etiquetas
etiquetas = ["persona", "bicicleta", "automóvil", "moto", "avión", "autobús", "tren", "camión",
        "barco", "semáforo", "boca de incendios", "señal de alto", "parquímetro", "banco",
        "pájaro", "gato", "perro", "caballo", "oveja", "vaca", "elefante", "oso", "cebra", "jirafa",
        "mochila", "paraguas", "bolso", "corbata", "maleta", "frisbee", "esquís", "snowboard",
        "pelota deportiva", "cometa", "bate de béisbol", "guante de béisbol", "monopatín", "tabla de surf",
        "raqueta de tenis", "botella", "copa de vino", "taza", "tenedor", "cuchillo", "cuchara", "tazón", "plátano",
        "manzana", "sándwich", "naranja", "brócoli", "zanahoria", "perrito caliente", "pizza", "rosquilla", "pastel",
        "silla", "sofá", "planta en maceta", "cama", "mesa de comedor", "inodoro", "monitor de televisión", "portátil", "ratón",
        "control remoto", "teclado", "teléfono celular", "microondas", "horno", "tostador", "fregadero", "nevera",
        "libro", "reloj", "jarrón", "tijeras", "oso de peluche", "secador de pelo", "cepillo de dientes"]
```

Figura 20 Código usado para el paso 7.

Finalmente, el modelo YOLO está listo para hacer una predicción, en la que el siguiente código permite al usuario ingresar una imagen para detectar objetos, ver figura 21.

```
de archivos de importación de google.colab
subir = archivos.subir()

para fn en upload.keys():
  photo_filename = '/contenido/' + fn

  # definir la forma de entrada esperada para el modelo
  entrada_w, entrada_h = 416, 416

  image, image_w, image_h = load_image_pixels(photo_filename, (input_w, input_h))

  # hacer predicción
  yhat = yolov3.predict(imagen)
  # resumir la forma de la lista de arreglos
  print([a.shape for a in yhat])

  cajas = lista()
  para i en el rango (len (yhat)):
    # decodificar la salida de la red
    cajas += decode_netout(yhat[i][0], anclas[i], class_threshold, input_h, input_w)

  # corregir los tamaños de los cuadros delimitadores para la forma de la imagen
  correct_yolo_boxes(cajas, image_h, image_w, input_h, input_w)

  # suprimir cajas no máximas
  do_nms(cajas, 0.5)

  # obtener los detalles de los objetos detectados
  v_boxes, v_labels, v_scores = get_boxes(boxes, etiquetas, class_threshold)

  # resumir lo que encontramos
  para i en el rango (len (v_boxes)):
    imprimir(v_etiquetas[i], v_puntuaciones[i])

  # dibujar lo que encontramos
  draw_boxes (foto_nombre de archivo, v_boxes, v_labels, v_scores)
```

Figura 21 Código para la predicción.

En la figura 22 se observa la producción:

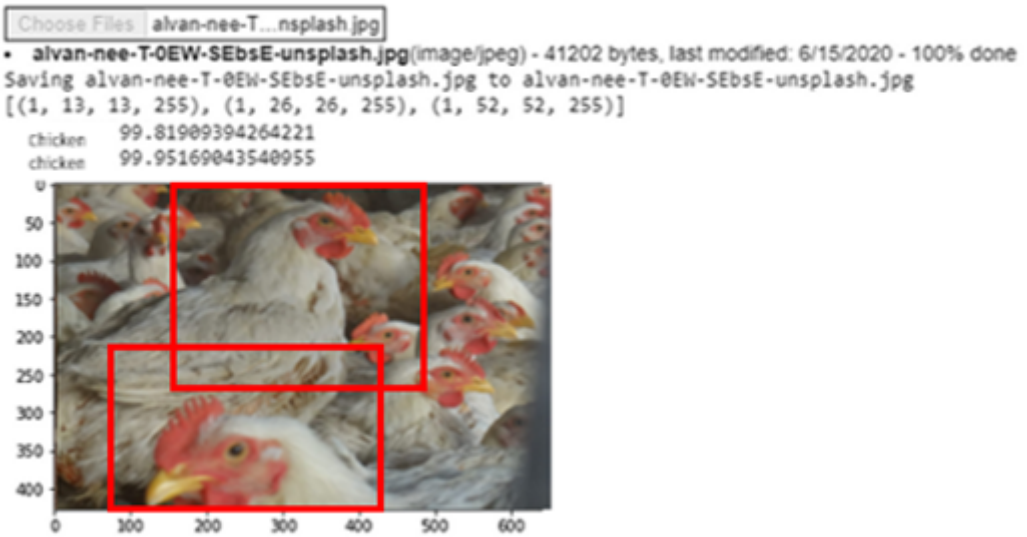

Figura 22 Producción.

Fase 3.

Validar el funcionamiento del prototipo de reconocimiento y conteo automático de pollos utilizando el sistema de visión artificial, en esta fase se procederá a verificar el funcionamiento, para lograr el objetivo principal del proyecto de investigación.

Lo primero que se debe realizar es checar las cámaras de videovigilancia, que estén correctamente instaladas y que puedan transmitir los videos por medio de la computadora, ver figura 23.

Figura 23 Cámaras de videovigilancia.

En la figura 24 se muestra el software de registro de imágenes el cual alberga la data almacenada.

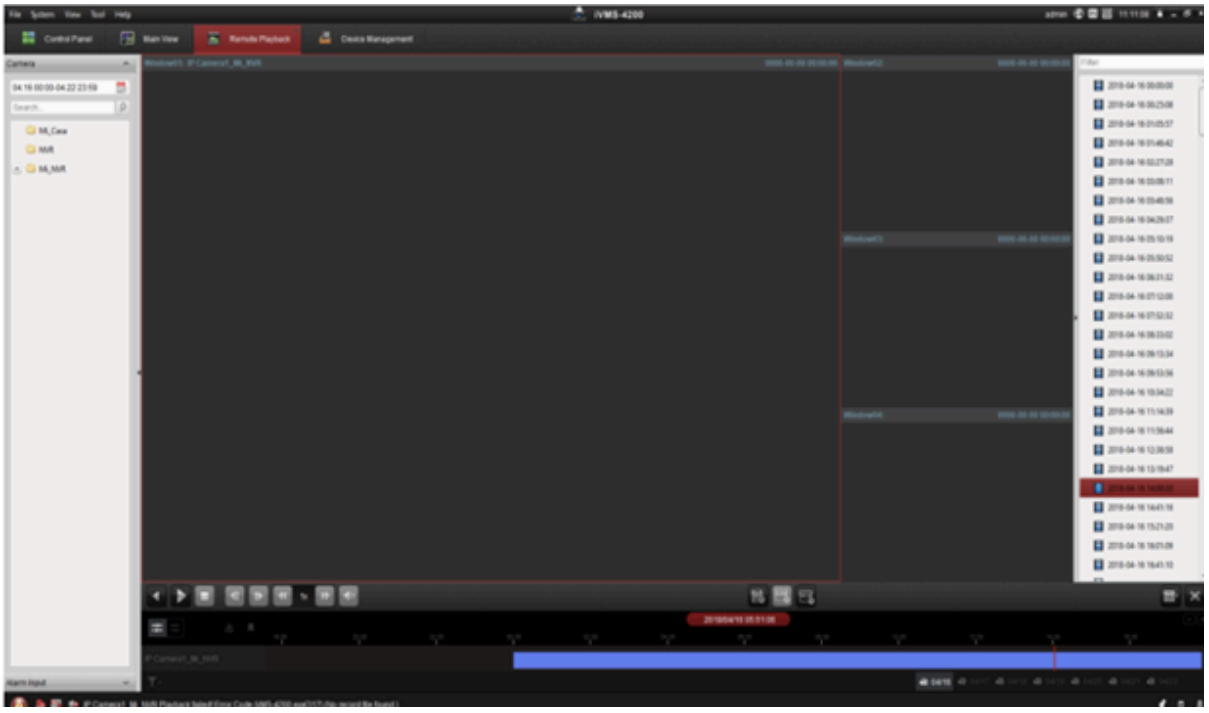

Figura 24 software de registro de imágenes.

Después de comprobar el funcionamiento de las cámaras, el programa YOLOv3 procede a realizar su función, la cual es el detectar los pollos y realizar el conteo, ver figura 25. En la fase 2, se encuentra todo el proceso que realiza YOLOv3 en conjunto con Keras.

Figura 25 Conteo de los pollos mediante YOLOv3.

Tabla 1 Resultado de conteo con YOLOv3

	erdad básica	etectado	Precisión de conteo	ror de conteo	ecisión promedio	Cuadros por segundo
YOLOv3	30	27	0.97	0.3	0.94	5.11

Conclusiones:

El conteo de pollos es una tarea fundamental en la industria avícola, ya que proporciona información valiosa para el seguimiento del crecimiento, la salud y el rendimiento de las aves. En los últimos años, el desarrollo de técnicas de visión por

computadora ha permitido automatizar este proceso, mejorando la eficiencia y precisión del conteo. En particular, el software YOLOv3 ha demostrado ser una herramienta efectiva para contar pollos en imágenes y videos.

Durante el análisis de las imágenes, el software YOLOv3 utiliza una red neuronal convolucional para detectar y localizar objetos. El modelo ha sido entrenado con una amplia variedad de imágenes que incluyen diferentes razas de pollos, poses y condiciones de iluminación. Esto le permite reconocer y contar pollos con una notable precisión, superando en muchos casos la capacidad humana de conteo.

Una de las ventajas más destacadas de utilizar YOLOv3 para el conteo de pollos es su capacidad de procesamiento en tiempo real. Esto significa que el software puede analizar rápidamente grandes volúmenes de imágenes o secuencias de video, lo que resulta en un ahorro significativo de tiempo y recursos para los productores avícolas. Además, al ser una solución basada en software, YOLOv3 es flexible y escalable, lo que permite su implementación en diferentes entornos y sistemas.

La precisión del conteo de pollos con YOLOv3 es impresionante, aunque no está exenta de desafíos. Las imágenes con condiciones de iluminación adversas, o con pollos en posiciones difíciles de detectar, pueden presentar ciertas limitaciones en el desempeño del software. Sin embargo, a medida que se recopila más información y se entrena al modelo con datos específicos de la industria avícola, es probable que estas limitaciones se minimicen.

La adopción del software YOLOv3 para el conteo de pollos ofrece una serie de beneficios tanto para los productores como para las aves. En primer lugar, mejora la eficiencia del proceso de conteo, liberando recursos humanos que pueden ser asignados a otras tareas críticas en la granja. Además, al ser una herramienta automatizada, reduce la posibilidad de errores humanos en el conteo, lo que resulta en datos más precisos y confiables.

Por otro lado, el uso de YOLOv3 también puede mejorar la calidad de vida de los pollos. Un conteo más preciso permite un monitoreo más efectivo de la densidad poblacional en los gallineros, lo que ayuda a garantizar un espacio adecuado para cada ave y a prevenir problemas de salud y estrés. Además, al proporcionar datos precisos sobre el crecimiento y el rendimiento de las aves, el software contribuye a

la toma de decisiones informadas para optimizar el bienestar animal y la productividad.

En conclusión, el conteo de pollos utilizando el software YOLO V3 es una solución altamente efectiva y prometedora en la industria avícola. Su capacidad de procesamiento en tiempo real, precisión y escalabilidad lo convierten en una herramienta valiosa para mejorar la eficiencia operativa, reducir errores y optimizar el bienestar animal. Con el continuo desarrollo y entrenamiento de los modelos de visión por computadora, es probable que veamos aún más avances en el conteo automatizado de pollos y otras aplicaciones en la industria avícola en el futuro.

Referencias:

[1] T. Huscio y R. Trochimczuk, «NOVEL ROPE-FREE MECHATRONIC ELEVATOR SYSTEM TO AUTOMATION OF TRANSPORT IN AGRICULTURAL FARMS,» Bialystok University of Technology, Jelgava, 2016.

[2] C.-H. C. Hung-Wei Liu, Yao-Chuan Tsai y Kuang-Wen Hsieh and Hao-Ting Lin, «MPDI,» MPDI , 21 mayo 2023. [En línea]. Available: https://www.mdpi.com/1424-8220/21/11/3579. [Último acceso: 2 marzo 2023].

[3] L. B. Cao, z. xiao, X. Liao, Y. Yao, K. J. Wu, Jiongmu, j. li y y. pu., «Conteo de pollos automatizado en entornos de cámaras de vigilancia basado en el algoritmo de supervisión de puntos: LC-DenseFCN,» *Agriculture,* vol. 11, nº 6, p. 493, 2021.

[4] H. L. Q. Z. Z. Y. w. x. Yiqinbao, «Sistema de detección de pollos muertos y enfermos en granjas a gran escala basado sobre inteligencia artificial, Yiqinbao1,2,*, Hongbing Lu2, Qiang Zhao3, Zhongxue Yang1y wenbin xu4 13JULIO/2021,» licenciatario AIMS Press., CANADA, 2021.

[5] S. Neethirajan, *ChickTrack – A quantitative tracking tool for measuring chicken activity,* Farmworx, Adaptation Physiology Group, Animal Sciences Department, Wageningen University, The Netherlands: ELSEVIER.

[6] LENKE, «CA-12000 Contador de Aves,» LENKE, 2016. [En línea]. Available: CA-12000 Contador de Aves. [Último acceso: 04 Mayo 2023].

[7] IndustriaAvícola, «Petersime contador de pollitos,» IndustriaAvícola, 2023. [En línea]. Available: https://www.industriaavicola.net/product/petersime-contador-de-pollitos/#:~:text=El%20contador%20de%20pollitos%20de,en%20la%20sala%20de%20extracci%C3%B3n.. [Último acceso: 04 Mayo 2023].

[8] Zoetis, «KL AUTOMATION | CONTADOR DE POLLITOS DE CINCO CARRILES,» Zoetis, 2023. [En línea]. Available: https://www.zoetis.mx/products/avicola/kl-contador-de-pollitos-cinco-carriles.aspx. [Último acceso: 04 Mayo 2023].

[9] S. C. Chinchajoa y J. A. V. Ortiz., *Prototipo de un sistema automático para el reconocimiento y conteo de pollos utilizando visión artificial para la empresa POFRESCOL LTDA,* San Juan de Pasto , p. 25.