UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo
a la obtención del título
de Ingeniero Electrónico

PROYECTO TÉCNICO CON ENFOQUE INVESTIGATIVO:

"DISEÑO DE UN PROTOTIPO DE DETECCIÓN Y RECONOCIMIENTO SEÑALÉTICO DE SEMÁFORO PEATONAL PARA PERSONAS NO VIDENTES HACIENDO USO DE VISIÓN ARTIFICIAL Y SENSORES ULTRASÓNICOS"

AUTORES:

FRANCISCO PATRICIO BRITO CAMPOS

CARLOS NICOLÁS PERALTA MOGROVEJO

TUTOR:

ING. EDUARDO GUILLERMO PINOS VÉLEZ, Ph.D.

CUENCA - ECUADOR

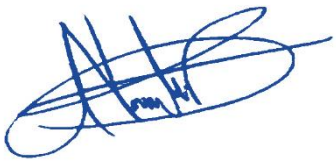
2022

CESIÓN DE DERECHOS DE AUTOR

Nosotros, Francisco Patricio Brito Campos con documento de identificación N° 0105250732 y Carlos Nicolás Peralta Mogrovejo con documento de identificación N° 0106106362, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de titulación: **"DISEÑO DE UN PROTOTIPO DE DETECCIÓN Y RECONOCIMIENTO SEÑALÉTICO DE SEMÁFORO PEATONAL PARA PERSONAS NO VIDENTES HACIENDO USO DE VISIÓN ARTIFICIAL Y SENSORES ULTRASÓNICOS"**, mismo que ha sido desarrollado para optar por el título de: *Ingeniero Electrónico*, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribimos este documento en el momento que hacemos entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, enero de 2022.



Francisco Patricio Brito Campos
C.I. 0105250732

Carlos Nicolás Peralta Mogrovejo
C.I. 0106106362

CERTIFICACIÓN

Yo, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **“DISEÑO DE UN PROTOTIPO DE DETECCIÓN Y RECONOCIMIENTO SEÑALÉTICO DE SEMÁFORO PEATONAL PARA PERSONAS NO VIDENTES HACIENDO USO DE VISIÓN ARTIFICIAL Y SENSORES ULTRASÓNICOS”**, realizado por Francisco Patricio Brito Campos y Carlos Nicolás Peralta Mogrovejo, obteniendo el *Proyecto Técnico con enfoque investigativo* que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana.

Cuenca, enero de 2022.

Ing. Eduardo Guillermo Pinos Vélez, PhD.

C.I. 0102942190

DECLARATORIA DE RESPONSABILIDAD

Nosotros, Francisco Patricio Brito Campos con documento de identificación N° 0105250732 y Carlos Nicolás Peralta Mogrovejo con documento de identificación N° 0106106362, autores del trabajo de titulación: **"DISEÑO DE UN PROTOTIPO DE DETECCIÓN Y RECONOCIMIENTO SEÑALÉTICO DE SEMÁFORO PEATONAL PARA PERSONAS NO VIDENTES HACIENDO USO DE VISIÓN ARTIFICIAL Y SENSORES ULTRASÓNICOS"**, certificamos que el total contenido del *Proyecto Técnico con enfoque investigativo,* es de nuestra exclusiva responsabilidad y autoría.

Cuenca, enero de 2022.

Francisco Patricio Brito Campos
C.I. 0105250732

Carlos Nicolás Peralta Mogrovejo
C.I. 0106106362

AGRADECIMIENTOS

Yo Francisco Brito Campos expreso mi más sincera gratitud a las personas que me apoyaron durante estos 5 años de carrera, de forma especial a mis padres Iván Brito y Eliana Campos, también a mi tía Fabiola Campos y al resto de familiares imprescindibles para mi desarrollo como profesional y persona.

Yo Carlos Nicolás Peralta Mogrovejo expreso mi más sincero agradecimiento a mis padres, hermanos, mi familia y a los maestros de la vida como son mis profesores, ya que todos y cada uno de ellos han formado parte importante en mi vida y en mi formación académica, de esta forma les agradezco por su sincera gratitud y aprecio por ser parte de mi vida.

ÍNDICE

Página

ÍNDICE vi

ÍNDICE DE FIGURAS viii

ÍNDICE DE TABLAS xi

..... xi

Glosario xii

JUSTIFICACIÓN 1

INTRODUCCIÓN 2

1. ESTADO DEL ARTE 3
- **1.1 La Visión** 3
- **1.2 Discapacidad Visual** 3
- **1.3 Datos de discapacidad visual en Ecuador** 4
- **1.4 Visión artificial** 5
- **1.5 Sensores Ultrasónicos** 5
- **1.6 MSCOCO Dataset** 6
- **1.7 CPU** 7
- **1.8 GPU** 7

2. DESARROLLO DE LOS ALGORITMOS DE VISION ARTIFICIAL Y RECONOCIMIENTO SEÑALETICO DE SEMAFORO PEATONAL 7
- **2.1 Herramientas Virtuales** 7
- **2.1.1 Darknet** 7
- **2.1.2 OpenCV** 8
- **2.1.3 CUDA** 8
- **2.1.4 Python** 8
- **2.2 Arquitectura de la Red Neuronal** 9
- **2.3.1 Definición de Número de Clases a detectar** 12
- **2.3.2 Recopilación de imágenes para la base de datos** 12
- **2.3.4 Etiquetado de la base de datos con el uso de labelImg** 13
- **2.3.4.1 Instalación de labelImg** 14
- **2.3.4.2 Tipo de Formato** 15
- **2.3.4.3 Configuración del nombre o tipo de clases dentro de labelImg** 15
- **2.3.4.4 Etiquetado** 15
- **2.3.4.5 Características del "Bounding Box"** 16
- **2.3.5 Contador de imágenes etiquetadas por clase** 17
- **2.3.6 Descarte de imágenes sin etiqueta** 18
- **2.4 Aumento de Datos** 19
- **2.5 Desarrollo del modelo con YOLOv4-Tiny** 22
- **2.5.1 Descarga de Darknet** 22
- **2.5.2 Creación y activación del entorno virtual** 22
- **2.5.3 Instalación y verificación de CUDA (Opcional)** 22

2.5.4 Instalación de CUDNN (Solo si se instaló CUDA) ... 27
2.5.5 Instalación y verificación de OpenCV en Windows utilizando CMake ... 28
2.5.6 Configuración y compilación de Darknet en Windows usando CMake ... 28
2.5.7 Creación del set de entrenamiento y el set de prueba ... 34
2.5.8 Creación del archivo ".names" ... 36
2.5.9 Creación del archivo .data ... 36
2.5.10 Cálculo de los "Anchors" (Opcional) ... 37
2.5.11 Creación y configuración del archivo ".cfg" ... 39
2.5.12 Configuración del guardado automático de pesos ... 41
2.5.13 Entrenamiento de la Red ... 43
2.5.14 Selección de los mejores pesos ... 46
2.5.15 Detección en imágenes de prueba usando los mejores pesos ... 47
2.5.16 Detección en tiempo real usando los mejores pesos ... 50
2.5.17 Aumento del rendimiento con "darknet_video.py" ... 51
2.5.18 Notificación auditiva del estado de señalización del semáforo peatonal ... 54
2.6 Adaptación del modelo en la Raspberry Pi 4 ... 57
2.6.1 Conversión al modelo de Tensorflow ... 58
2.6.2 Conversión al modelo de Tensorflow Lite ... 60
2.6.3 Dictador de clases usando los pines GPIO de la Raspberry Pi 4 ... 63

3. CONTRUCCIÓN DEL DISPOSITIVO ... 65
3.1. Arquitectura del Dispositivo ... 65
3.2. Dispositivos de Hardware ... 66
3.2.1. Microcontrolador Arduino NANO ... 66
3.2.2. Rasberry PI 4 B de 4GB de Memoria RAM ... 66
3.2.3. Sensores Ultrasónicos MaxSonar EZ1: ... 67
3.2.4. Motores de Vibración Ruiling 1034 ... 69
3.3. Diseño y Construcción del Dispositivo en Inventor ... 71
3.3.1. Diseño de Gafas ... 71
3.3.2. Diseño de Carcasa para Controlador ... 74

4. ANÁLISIS DE RESULTADOS ... 76
4.1 Selección de los mejores pesos ... 76
4.2 Detecciones realizadas con Darknet ... 79
4.3 Adaptación del modelo en la Raspberry Pi 4 ... 82
4.4 Conversión al modelo de Tensorflow ... 83
4.5 Conversión al modelo de TensorFlow Lite ... 84
4.6 Detecciones realizadas con TensorFlow Lite ... 84
4.7 Detecciones en imágenes de entrada con "detect.py" ... 88
4.8 Arranque automático del Script "detect.py" en la Raspberry Pi 4 ... 92
4.9 Encendido y Apagado de la Raspberry Pi 4 ... 93

CONCLUSIONES ... 95

RECOMENDACIONES Y TRABAJOS FUTUROS ... 96

REFERENCIAS ... 97

ÍNDICE DE FIGURAS

Figure 1 Porcentaje de datos sobre discapacidades en el Ecuador [7] 5
Figure 2 Sensor de distancia Ultrasónico [9] 6
Figure 3 Detección de Objetos de una imagen de las calles de Cuenca utilizando YOLO en Google Colab [11] [10] 6
Figure 4 Detección de objetos realizado con "Darknet" usando la arquitectura y los pesos pre-entrenados de "YOLO" [11] 8
Figure 5 Grafica comparativa AP (Average Precision) vs FPS de diferentes arquitecturas de red [25] 10
Figure 6 Detección de objetos en tiempo real Yolo v4 [Fuente: Los Autores] 11
Figure 7 Detección de objetos en tiempo real Yolo v4 Tiny [Fuente: Los Autores] 11
Figure 8 Semáforo visto desde una perspectiva más o menos frontal e isométrica [Fuente los Autores] 12
Figure 9 Zonas de la fotografía en las que aparece un semáforo con más recurrencia [Fuente los Autores] 13
Figure 10 Script para el renombramiento de las Imágenes recuperadas [Fuente los Autores] ... 13
Figure 11 Comando para abrir la interfaz gráfica de labelImg [Fuentes los Autores] 14
Figure 12. Interfaz gráfica de labelImg [Fuentes los Autores] 14
Figure 13 Formato YOLO dentro de LabelImg [Fuentes los Autores] 15
Figure 14 Definición del nombre de las clases en labelImg [Fuente Los Autores] 15
Figure 15 Creación del bounding box y asignación de la clase correspondiente [Fuentes Los Autores] 16
Figure 16 Algoritmo para conteo de etiquetas por clase [Fuente: Los Autores] 17
Figure 17 Resultado Obtenido después de ejecutar el algoritmo [Fuentes los Autores] 18
Figure 18 Algoritmo para eliminación de imágenes sin etiqueta [Fuente: Los Autores] 18
Figure 19 Fig. Imagen Original (Izquierda) e Imagen Volteada (Derecha) [Fuente: Los Autores] 19
Figure 20 Fig. Imagen Original (Izquierda) e Imagen Rotada -15° y aplicando el método "reflect" para conservar las dimensiones de la imagen (derecha) [Fuente: Los Autores] 20
Figure 21 Algoritmo para aumento de datos escritos en Python [Fuente: Los Autores] 21
Figure 22 Verificación para saber que GPU se tiene en la computadora [fuente: Los Autores] 23
Figure 23 Verificación de la versión de CUDA instalada y versión de controladores de la tarjeta instalados antes de actualizar los drivers [Fuente: Los Autores] 24
Figure 24 Verificación de la versión de CUDA instalada y versión de controladores de la tarjeta instalados después de actualizar los drivers [Fuente: Los Autores] 24
Figure 25 Verificación de la versión de CUDA [Fuente: Los Autores] 25
Figure 26 Verificación de la detección correcta del hardware NVIDIA por CUDA. [Fuente: Los Autores] 26
Figure 27 Verificación correcta de la comunicación del hardware NVIDIA y el software CUDA [Fuente: Los Autores] 27
Figure 28 Instalación de Numpy en el sistema [Fuente: Los Autores] 28
Figure 29 Verificación de la versión de OpenCV instalada en el sistema [Fuente: Los Autores] 28
Figure 30 Entorno grafico de CMake 3.21.1 GUI [Fuente: Los Autores] 29

Figure 31 Resultados generados después de configurar CMake (marcados en rojo) [Fuente: Los Autores]..... 31
Figure 32 Opción de plataforma de solución en “x64” y configuración de solución en “Release” [Fuente: Los Autores] 31
Figure 33 Resultado devuelto por la terminal de Visual Studio al terminar la compilación. [Fuente: Los Autores] 31
Figure 34 Resultado correcto de la ejecución del comando “darknet.exe” [Fuente: Los Autores] 32
Figure 35 Inicio de ejecución de la detección en tiempo real usando la webcam de la computadora [Fuente: Los Autores]..... 33
Figure 36 Detección en tiempo real usando el modelo YOLOv4 [Fuente: Los Autores]..... 33
Figure 37 Detección en tiempo real usando el modelo YOLOv4-Tiny [Fuente: Los Autores].. 34
Figure 38 Ejecución del script “Split_train_test.py” [Fuente: Los Autores] 35
Figure 39 Contenido de los archivos .txt después de ejecutado el código de Python [Fuente: Los Autores]..... 36
Figure 40 Contenido del archivo “semaforos_peatonales.names” [Fuente: Los Autores]..... 36
Figure 41 Contenido del archivo “semáforos_peatonales.data” [Fuente: Los Autores] 37
Figure 42 Anchors generados para una resolución de 416x416 [Fuente: Los Autores] 38
Figure 43 Anchors generados para una resolución de 320x320 [Fuente: Los Autores] 39
Figure 44 Cambio en la línea 386 del código de “detector.c” [Fuente: Los Autores] 43
Figure 45 Cambio encima de la línea 301 en el código de “detector.c” [Fuente: Los Autores] . 43
Figure 46 8 archivos de pesos generados durante el entrenamiento y por cada 500 iteraciones [Fuente: Los Autores..... 44
Figure 47 Finalización del entrenamiento y generación de los pesos finales [Fuente: Los Autores]..... 44
Figure 48 Métricas calculadas durante una iteración de entrenamiento [Fuente: Los Autores] . 45
Figure 49 Representación gráfica del cálculo de la Intersección sobre Unión (IoU) [25]..... 46
Figure 50 Curvas de entrenamiento y prueba a lo largo de las iteraciones de entrenamiento [25] 46
Figure 51 Grafica obtenida que representa la variación de la pérdida por cada iteración de entrenamiento [Fuente: Los Autores] [25],..... 47
Figure 52 Detección en tiempo real de un mouse que emite luces verdes que es clasificado como semáforo verde [Fuente: Los Autores]..... 50
Figure 53 Detección en tiempo real de un mouse que emite luces rojas que es clasificado como semáforo rojo [Fuente: Los Autores] 50
Figure 54 Detección en tiempo real con un umbral de detección de 0.8 [Fuente: Los Autores] 51
Figure 55 Cambio de la versión de CUDA que se usara en la compilación del proyecto [Fuente: Los Autores]..... 52
Figure 56 Resultado devuelto después de la compilación exitosa [Fuente: Los Autores] 52
Figure 57 Adición de dos líneas de código en caso de error “Not found DLL” [Fuente: Los Autores]..... 53
Figure 58 Detecciones en tiempo real con “darknet_video.py” con un “thresh=0” [Fuente: Los Autores]..... 53
Figure 59 Instalación de “pyttsx3” con pip [Fuente: Los Autores]..... 54
Figure 60 Instalación de “pynput” con pip [Fuente: Los Autores] 54
Figure 61 Adición de tres líneas de código en la función “drawing” para obtener las etiquetas [Fuente: Los Autores] 55
Figure 62 Script para leer la etiqueta en “detecciones.txt” y pasarla a audio [Fuente: Los Autores]..... 56

Figure 63 Ejecución del archivo.bat “Detector_Dictador_python.bat [Fuente: Los Autores]” .. 57
Figure 64 Descarga de darknet [Fuente: Los Autores] 57
Figure 65 Configuración del archivo “Makefile” [Fuente: Los Autores] 58
Figure 66 Nuevo contenido de “semaforos_peatonales.data” [Fuente: Los Autores]..... 58
Figure 67 Configuración de parámetros en “config.py” [Fuente: Los Autores] 59
Figure 68 Conversión de tipo “string” a “int” en la función “VideoCapture” [Fuente: Los Autores]..... 60
Figure 69 Error obtenido al momento de la conversión a Tensorflow Lite [Fuente: Los Autores] 61
Figure 70 Configuración del atributo “experimental_new_converter” del convertidor a “False” [Fuente: Los Autores] 61
Figure 71 Detecciones realizadas con TensorFlow Lite con el intérprete por defecto [Fuente: Los Autores]..... 62
Figure 72 Cambio al nuevo intérprete en el código (línea con el comentario #adicionada) del anterior intérprete (línea con el comentario #original) [Fuente: Los Autores] 63
Figure 73 Script para obtener la salida de audio con la clase detectada mediante el uso de un pulsante en los pines GPIO de la Raspberry Pi 4 [Fuente: Los Autores]..... 64
Figure 74 Arquitectura de Dispositivo (Sistema de Control) [Fuentes los Autores] 65
Figure 75 Arduino NANO [56]..... 66
Figure 76 Rasberry Pi 4 B de 4GB de memoria RAM [57]..... 67
Figure 77 Medidas de Sensor Ultrasónico MaxSonar EZ1 [56] 67
Figure 78 Representación de los haces de luz sensor ultrasónico MaxSonar [59]..... 69
Figure 79 Motores de Vibración Ruiling 1034 [60]..... 69
Figure 80 Modelo 3D y Plano de Cobertura Delantera [Fuente: Los Autores] 71
Figure 81 Modelo 3D y Plano de Cobertura Posterior [Fuente: Los Autores]..... 72
Figure 82 Modelo 3D y Plano de Soporte de Componentes [Fuente: Los Autores]..... 73
Figure 83 Modelo 3D de Ensamble de Piezas para la construcción de las gafas [Fuente: Los Autores]..... 73
Figure 84 Modelo 3D y Plano del Case para el Controlador [Fuente: Los Autores]..... 74
Figure 85 Modelo 3D y Plano de la parte superior del Case para el Controlador [Fuente: Los Autores]..... 75
Figure 86 Modelo 3D de Ensamble de Piezas para la construcción del Case [Fuente: Los Autores]..... 75
Figure 87 Resultados obtenidos para las 500 iteraciones [Fuente: Los Autores] 76
Figure 88 Resultados obtenidos para las 1000 iteraciones [Fuente: Los Autores] 77
Figure 89 Resultados obtenidos para las 1500 iteraciones [Fuente: Los Autores] 77
Figure 90 Resultados obtenidos para las 2000 iteraciones [Fuente: Los Autores] 78
Figure 91 Resultados obtenidos para las 2500 iteraciones [Fuente: Los Autores] 78
Figure 92 Resultados obtenidos para las 3000 iteraciones [Fuente: Los Autores] 78
Figure 93 Resultados obtenidos para las 3500 iteraciones [Fuente: Los Autores] 79
Figure 94 Resultados obtenidos para las 4000 iteraciones [Fuente: Los Autores] 79
Figure 95 Imagen de detección 1 [Fuente: Los Autores]..... 80
Figure 96 Imagen de detección 2 [Fuente: Los Autores] 80
Figure 97 Imagen de detección 3 [Fuente: Los Autores]..... 81
Figure 98 Imagen de detección 5 [Fuente: Los Autores]..... 81
Figure 99 Imagen de detección 6 [Fuente: Los Autores]..... 82
Figure 100 Detecciones realizadas con darknet en la Raspberry Pi 4 [Fuente: Los Autores]..... 82

Figure 101 Detecciones usando Tensorflow con la arquitectura YOLOv4-tiny en Raspberry Pi 4 [Fuente: Los Autores] 83
Figure 102 Detecciones realizadas con el nuevo intérprete [Fuente: Los Autores] 84
Figure 103 Customización de las banderas en el script "detect.py" [Fuente: Los Autores] 85
Figure 104 Detección en la imagen 1 utilizando TensorFlow Lite [Fuente: Los Autores] 85
Figure 105 Detección en la imagen 2 utilizando TensorFlow Lite [Fuente: Los Autores] 86
Figure 106 Detección en la imagen 3 utilizando TensorFlow Lite [Fuente: Los Autores] 87
Figure 107 Detección en la imagen 4 utilizando TensorFlow Lite [Fuente: Los Autores] 87
Figure 108 Detección en la imagen 5 utilizando TensorFlow Lite [Fuente: Los Autores] 88
Figure 109 Importación de la librería dentro de "utils.py" [Fuente: Los Autores] 89
Figure 110 Adición de funciones extras en la función "draw_bbox" [Fuente: Los Autores] 89
Figure 111 Adición de los pares de datos dentro de la lista "clases_puntajes" [Fuente: Los Autores] 90
Figure 112 Condicional para obtener la etiqueta con el mayor valor de confianza [Fuente: Los Autores] 90
Figure 113Detección de dos objetos y selección de la clase con mayor score [Fuente: Los Autores] 91
Figure 114 Detección de dos objetos en una imagen de la misma clase, pero diferente score [Fuente: Los Autores] 91
Figure 115 Fragmento de código para obtener la imagen de la webcam con OpenCV [Fuente: Los Autores] 92
Figure 116 Contenido de "semaforos.desktop" [Fuente: Los Autores] 92
Figure 117 Resumen de banderas en "detect.py" y cambio a ruta absoluta de los pesos de la red [Fuente: Los Autores] 93
Figure 118 Cambio a ruta absoluta en "config.py" [Fuente: Los Autores] 93
Figure 119 Script para el apagado y encendido de una Raspberry Pi 4 [Fuente: Los Autores] .. 94
Figure 120 Contenido de "apagado_semaforos.desktop" [Fuente_ Los Autores] 94

ÍNDICE DE TABLAS

Tabla 1 Causas de la Discapacidad Visual [4] 4
Tabla 2 Personas con tipos de discapacidades en el Ecuador [7] 4
Tabla 3 Tabla de Medidas Sensor Ultrasónico MaxSonar EZI [56] 67

Glosario

CONADIS: Consejo Nacional para la igualdad de discapacidades.

MSCOCO: Microsoft Common Objects in Context.

YOLO: You only look once.

FPS: Frames per second.

MAP: Average mean precision.

CPU: Unidad central de procesamiento

GPU: Unidad de procesamiento gráfico

CUDA: Arquitectura Unificada de Dispositivos de Cómputo

R-CNN: Regions with Convolutional Neural Network feature

JUSTIFICACIÓN

Uno de los mayores retos que afronta una persona con discapacidad visual se presenta al salir al centro de la ciudad e intentar desenvolverse de manera independiente en el tránsito peatonal. Aunque solo un sector de la población tiene acceso a algún tipo de asistencia electrónica, se ven totalmente indefensos ante la situación más básica dentro del tránsito, que es cruzar la calle dependiendo de qué color marque el semáforo peatonal.

La ciudad de Cuenca cuenta con una buena señalización para el tránsito de peatones, así como para el tránsito vehicular. Sin embargo, las personas con ceguera no pueden aprovechar estas señalizaciones públicas y obligadamente llegan a generar un grado de dependencia que puede ser hacia sus parientes o en algunos casos, cuando se valen por sí mismos, hacia la sociedad. Llevándolos de manera necesaria a congeniar con personas desconocidas con el fin de solicitar ayuda para realizar algunas acciones de movilización dentro del tráfico de la ciudad.

El desarrollar un prototipo vinculado con la seguridad vial para los peatones, es de vital importancia, ya que se estaría instaurando una mejor calidad de vida para las personas no videntes. Además, este dispositivo debe tener una velocidad de procesamiento y comunicación eficiente esto con el fin de evitar confusiones o problemas de entendimiento con el usuario.

INTRODUCCIÓN

Actualmente existen alrededor de 481.392 personas que presentan algún tipo de discapacidad en el Ecuador, y de las cuales, casi el 12% poseen discapacidad visual en algún grado, según el número de personas con discapacidad registradas en el Consejo Nacional para la Igualdad de Discapacidades (CONADIS). [1]

Hasta el actual año, se han presentado varios tipos de dispositivos de asistencia electrónica para personas con discapacidad visual con el fin de que progresivamente puedan realizar tareas de forma independiente y segura. Estos dispositivos sirven para que las personas no videntes puedan ubicar, tanto de manera auditiva como sensorial, los objetos que los rodean, siendo esta una de las maneras para que el no vidente pueda desenvolverse mejor en el entorno que lo rodea; sin embargo, gracias a los avances en visión artificial, se pone a disposición una nueva herramienta para poder complementar y mejorar las tecnologías de asistencia para personas no videntes. Existen algunas empresas que ya han empezado hace algunos años a comercializar dispositivos con este tipo de tecnología, como es la empresa española OrCam, con su dispositivo para personas con deficiencia visual llamado “MyEye 2.0” , el cual realiza lectura de textos, reconoce rostros, identifica billetes, reconoce productos, entre otras cosas. [2]

Por esta razón el sistema que se pretende realizar es un dispositivo de uso práctico para personas no videntes que no afecte en su calidad de vida y les permita realizar sus actividades diarias de forma más autónoma.

A continuación, se presenta un nuevo prototipo el cual consta de 2 partes, la primera: la detección y reconocimiento señalético de los semáforos peatonales y la segunda: el sistema de detección de obstáculos que consta de dispositivos de vibración que indicarán al usuario en qué dirección se encuentra un objeto y el nivel de vibración le indicara de forma progresiva que tan cerca está de colisionar la persona con el objeto. Para esta última parte del sistema global se usarán los sensores ultrasónicos y motores de vibración.

CAPITULO 1

1. ESTADO DEL ARTE

En este capítulo se define sobre la visión, visión artificial, los tipos de discapacidad visual y la discapacidad visual en Ecuador. Además de los diferentes componentes que serán usados para la realización de la detección de semáforos y dentro del desarrollo del proyecto.

1.1 La Visión

La visión es el principal medio que tienen los seres humanos para conocer el entorno en el que nos encontramos y así poder relacionarse con nuestros semejantes, el órgano que nos proporciona la visualización es el ojo. El ojo recepta información sobre el color, forma, distancia, posición y movimientos de los objetos que se encuentran en nuestro entorno. [3]

1.2 Discapacidad Visual

Discapacidad visual es la deficiencia en el funcionamiento del ojo humano, que causa deficiencia o pérdida total en el sentido de la visión y existen diferentes tipos de discapacidad visual como se describen a continuación [4]

- **Baja visión leve:** Son aquellas personas que pueden realizar actividades diarias sin inconveniente siempre y cuando se utilizan ayudas ópticas adecuadas, estas personas pueden percibir objetos de tamaños pequeños. [5]
- **Baja visión moderada:** Estas personas pueden percibir objetos con distancias cortas, siempre y cuando los objetos sean grandes y la luz favorezca a percibir objetos, este tipo de personas generalmente necesitan ayuda de lentes de aumento para poder estar correctamente en el medio. [6]
- **Baja visión severa:** Las personas con esta dificultan podrán percibir solo luz y sombra, pero necesitan de ayuda para hacerlo como es el caso de braille para la lectura y escritura, así como lentes y telescopios especiales para poder realizar sus actividades solas.
- **Ceguera Total:** Las personas que padecen de esta dificultad no poseen visión alguna, no pueden percibir ni sombras ni luz, lo que hace que les sea imposible realizar tareas cotidianas, este tipo de personas necesitan de ayuda para leer y escribir además necesitan de una persona para poder movilizarse de mejor manera.

Varias son las causas que provocan estas discapacidades visuales, entre estas tenemos las siguientes:

Tabla 1 Causas de la Discapacidad Visual [4]

Hereditarias	**Congénicas**	**Adquirida/Accidental**	**Tóxico/Tumoral**
Albinismo	Enoftalmia	Avitaminosis	Histoplasmosis
Aniridia	Atrofia de nervio óptico	Cataratas Traumáticas	Infecciones diversas del sistema
Atrofia del nervio óptico	Cataratas Congénicas	Desprendimiento de retina	Meningitis
Cataratas Congénicas	Microftalmia	Diabetes	Neuritis óptica
Coloboma	Rubéola	Estasis papilar	Rubeola
Glaucoma congénito	Toxoplasmosis	Fibroplasia retrolental	Toxoplasmosis
Miopía degenerativa		Glaucoma Adulto	
Queratocono		Hidrocefalia	
Retinitis pigmentaria			

1.3 Datos de discapacidad visual en Ecuador

Datos obtenidos de la Organización Mundial de la Salud (OMS) se dice que alrededor de 285 millones de personas con discapacidad visual en el mundo, donde 39 millones son ciegas y 246 millones presentan baja visión. [4]

En el Ecuador según datos estadísticos por el Consejo Nacional de Igualdad de Discapacidades (CONADIS) existen 54662 personas que tienen discapacidad visual (ver tabla 2). [7]

Esto representa 11,58% de diferentes tipos de discapacidades a nivel Nacional (figura 1), lo cual corresponde a un índice alto en la sociedad. [7]

Tabla 2 Personas con tipos de discapacidades en el Ecuador [7]

TIPO DE DISCAPACIDAD		**PERSONAS**
FISICA		216.479
INTELECTUAL		108.588
AUDITIVA		66.722
VISUAL		54.662
PSICOSOCIAL		25.762
TOTAL	472.213	

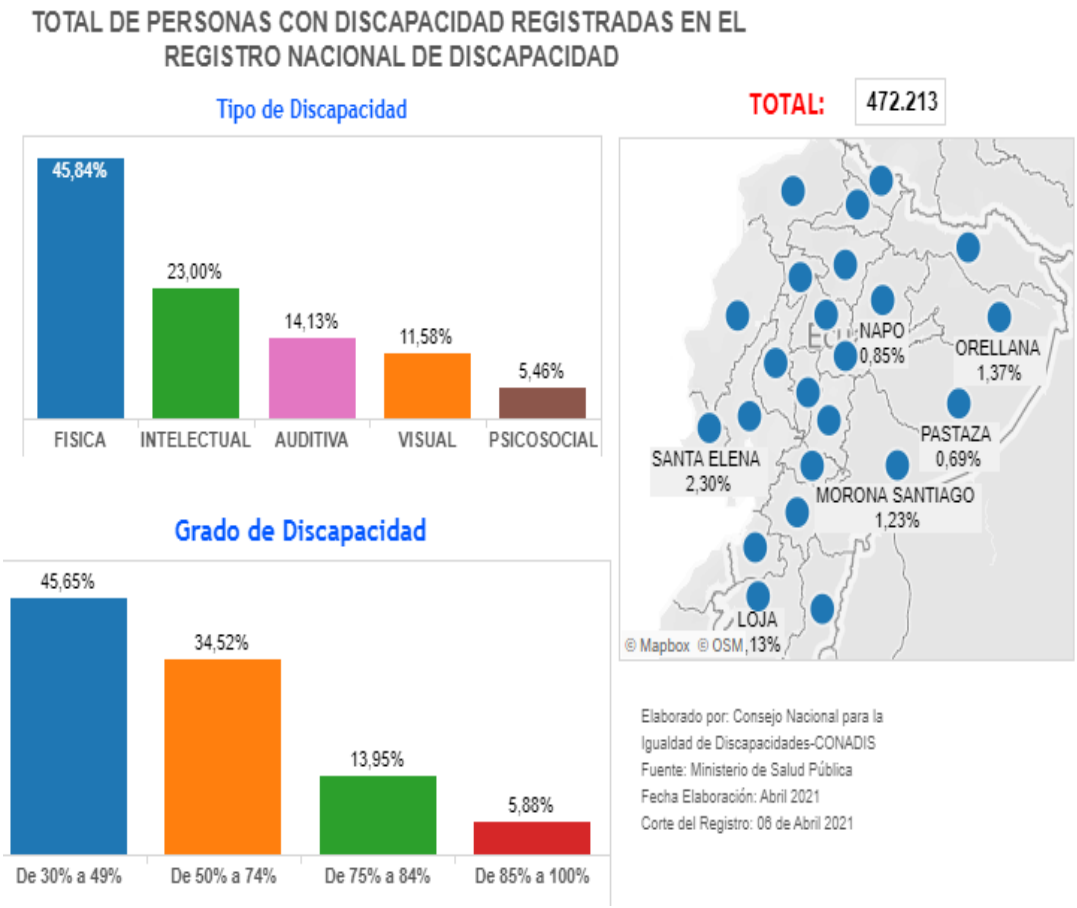


Figure 1 Porcentaje de datos sobre discapacidades en el Ecuador [7]

1.4 Visión artificial

La visión artificial es una disciplina que engloba todos los procesos y elementos que proporcionan ojos a una máquina y se podría decir que, la visión artificial o comprensión de imágenes describe la deducción de la estructura y propiedades de un mundo tridimensional, bien a partir de una o varias imágenes bidimensionales de ese mundo. [8]

1.5 Sensores Ultrasónicos

Los sensores ultrasónicos son interruptores electrónicos que trabajan sin contacto. La parte emisora genera pulsos de sonidos muy fuertes dentro del rango del ultrasonido. Hay sensores ultrasónicos unidireccionales pero la mayoría son bidireccionales. Aquí se aprovecha el efecto del rebote del sonido debido a la presencia del objeto. Además, según el tiempo que transcurre en ir y regresar el pulso sónico, se puede determinar la distancia entre el sensor y el objeto. [9]

Figure 2 Sensor de distancia Ultrasónico [9]

1.6 MSCOCO Dataset

El conjunto de datos de Microsoft Common Objects in Context (MSCOCO) contiene 91 categorías de objetos comunes y 82 de ellas tienen más de 5.000 instancias etiquetadas. Este conjunto de datos contiene 2.500.000 instancias etiquetadas en 328.000 imágenes. [10]

Dentro de estas 91 categorías, podemos encontrar una llamada “semáforos”, que, si lo llevamos a usar en un algoritmo de visión artificial, podemos llegar a identificar semáforos peatonales y vehiculares por igual, además de que, en la categoría antes mencionada, se excluye por completo el estado de señalización del semáforo.

A continuación, se muestra una imagen obtenida en el centro de la ciudad de Cuenca, a la que se le ha aplicado un reconocimiento de objetos utilizando el Dataset MSCOCO y la herramienta darknet, usando el modelo pre-entrenado de YOLO. Como podemos ver, la clase “Traffic light” no distingue entre semáforos peatonales y vehiculares.

Figure 3 Detección de Objetos de una imagen de las calles de Cuenca utilizando YOLO en Google Colab [11] [10]

1.7 CPU

Las CNN (Neural Network Convolutional) no se programan, estas se entrenan y es necesario configurar sus arquitecturas, capas y varios parámetros más que consumen varios períodos de la CPU. [12]

Trabajar solo con lo que es la CPU limita mucho la velocidad de procesamiento ya que esto se basa mucho en el ancho de banda de la memoria [13], Sin embargo al momento que se ejecutan varios procesos con la GPU y se puede obtener valores más precisos y operar en tiempo real a mayor velocidad pero los costos serían más elevados e impensables por lo que estrictamente se debe usar la CPU. [14]

1.8 GPU

Está adaptada al proceso de lo que son cálculos en paralelo. Ayuda a acelerar todo tipo de aplicaciones con un enfoque más profundo que en este caso vendrían a ser las redes neuronales, ya que las aplicaciones serían asignadas a la GPU mientras que la CPU se encarga de los códigos ejecutables. [15]

El uso de la GPU es mucho más complejo ya que su arquitectura es diferente a la CPU por lo que usa recursos de trabajo específicos para los dispositivos de programación, se tiene que considerar un lugar de entorno eficiente como: paralización de un algoritmo, organización y agrupación de subprocesos, uso adecuado de la memoria, comunicación y sincronización correcta de los subprocesos. [16]

CAPITULO 2

2. DESARROLLO DE LOS ALGORITMOS DE VISION ARTIFICIAL Y RECONOCIMIENTO SEÑALETICO DE SEMAFORO PEATONAL

En este capítulo se detalla la metodología empleada para el desarrollo de los algoritmos de detección de semáforos peatonales, además, las herramientas virtuales necesarias para lograrlo. Se desarrollará un modelo de visión artificial con la arquitectura de YOLOv4-tiny.

2.1 Herramientas Virtuales

2.1.1 Darknet

Darknet es un framework de red neuronal de código abierto escrito en C y CUDA. Es rápido, fácil de instalar y admite cálculos de CPU y GPU. [17]

Dentro de Darknet podremos, a partir de un modelo pre-entrenado, crear un nuevo modelo customizado para detectar los objetos en cuestión, que son los semáforos peatonales.

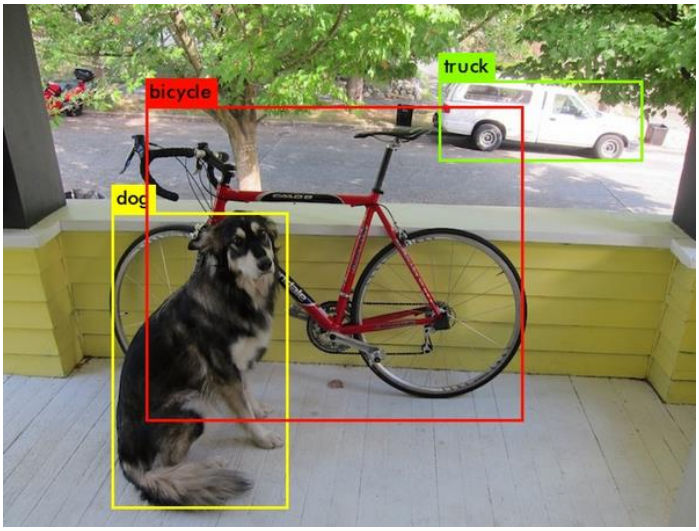


Figure 4 Detección de objetos realizado con "Darknet" usando la arquitectura y los pesos pre-entrenados de "YOLO" [11]

2.1.2 OpenCV

Es una librería de visión por computadora de código abierto que incluye varios cientos de algoritmos de visión por computadora [18]. Para este proyecto es un complemento necesario que utiliza el framework "Darknet" para la detección de objetos en tiempo real.

2.1.3 CUDA

Es una plataforma de computación en paralelo y un modelo de programación desarrollado por NVIDIA para computación general en unidades de procesamiento grafico (GPU). Con CUDA se puede acelerar drásticamente las aplicaciones informáticas aprovechando la potencia de las GPU [19]. En pocas palabras, acelerará el proceso de entrenamiento de nuestra red neuronal al ejecutar las operaciones aprovechando la potencia de nuestra GPU.

2.1.4 Python

Es un lenguaje de programación orientado a objetos (POO), está formado por varias instrucciones que se convierten en operaciones ejecutables para ser interpretadas. [20], se puede crear, configurar y ejecutar scripts, por medio de algunas herramientas y librerías que soporta Python. [21]

2.1.5 TensorFlow

Es una biblioteca de código abierto para desarrollar y entrenar modelos de aprendizaje automático [22], dentro de ella podremos encontrar una gran variedad de funciones que nos permitirán, junto con la API de "Tensorflow Objects Detection", crear los algoritmos para el desarrollo de nuestro modelo de detección de objetos.

2.1.6 LabelImg

Es una herramienta para crear anotaciones a partir de imágenes en varios tipos de formatos como XML PASCAL VOC, YOLO y CreateML. [23]

2.1.7 TensorFlow Lite

Es un framework de código abierto de aprendizaje profundo (Deep Learning) para la inferencia en dispositivos incorporados. Además, permite la inferencia de aprendizaje automático (Machine Learning) en dispositivos con una latencia baja y un tamaño de objeto binario pequeño. [24]

2.2 Arquitectura de la Red Neuronal

Existe una inmensa cantidad de arquitecturas de redes neuronales, en este caso buscamos una arquitectura de red neuronal convolucional (CNN) que pueda llegar a desempeñarse de buena manera en una Raspberry Pi 4 al realizar detecciones en tiempo real, entonces lo que se busca para nuestra aplicación, es un buen balance entre tiempo de respuesta y precisión, siendo el tiempo de respuesta lo más bajo posible en la medida que la precisión no sea tan mala. A continuación, hablaremos un poco de las arquitecturas que planteamos para desarrollar el modelo.

2.2.1 YOLOv4 y YOLOv4 Tiny

Dentro del repositorio de AlexeyAB [32] podemos encontrar una gráfica que compara la precisión y los fotogramas por segundo (FPS) de varias arquitecturas de red que se probaron con el Dataset de MSCOCO y podemos concluir que YOLOv4 tiene más precisión que YOLOv4 – Tiny pero la versión Tiny es casi ocho veces más rápida que YOLOv4, esto es debido a que YOLOv4 tiene 110 capas convolucionales a comparación

de las 21 que posee la versión Tiny, es decir, requiere de un mayor calculo computacional en toda la red debido a que posee una arquitectura más compleja.

Para algunos ordenadores con pocos recursos la versión Tiny podría ser una buena opción ya que no demanda que soporte una gran carga computacional, pero como veremos más adelante, las detecciones en tiempo real se verán muy limitadas en cuanto a velocidad de inferencia en la Raspberry Pi 4.

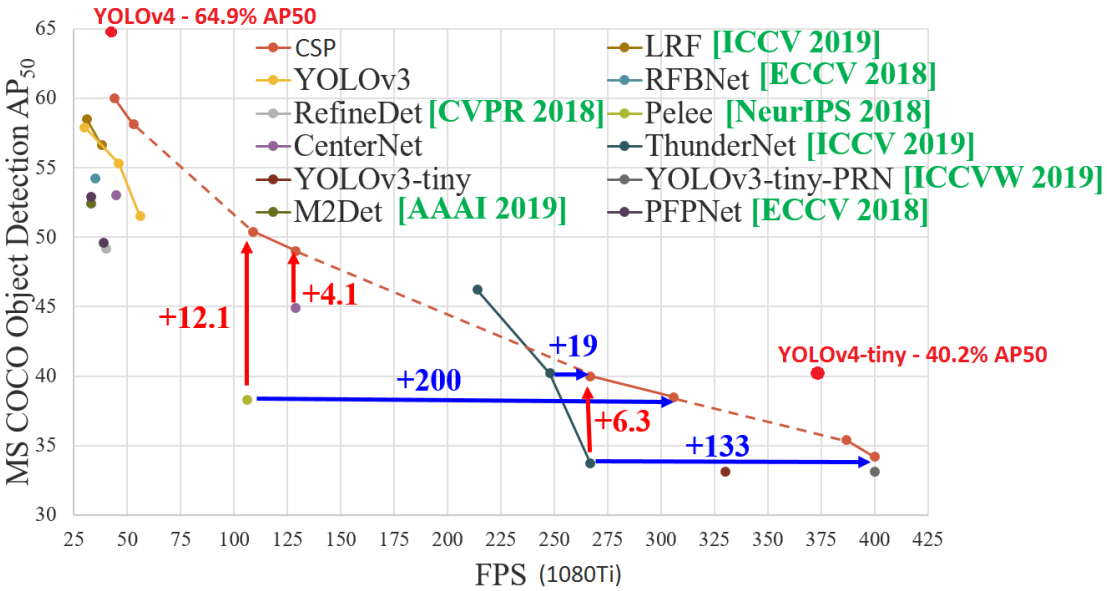


Figure 5 Grafica comparativa AP (Average Precision) vs FPS de diferentes arquitecturas de red [25]

Podemos analizar cada arquitectura disponible en el framework de "Darknet" dentro de la carpeta "cfg" del repositorio de Alex y adaptarlo a nuestras necesidades. [25]

Para tener una idea de la velocidad y precisión que se tiene con ambas versiones de YOLO, a continuación, se muestran los resultados obtenidos en una computadora con un procesador Intel(R) Core (TM) i7-5500U CPU @ 2.40GHz 2.40 GHz, una RAM de 16GB y con Sistema Operativo Windows 10 de 64 Bits.

Utilizando la arquitectura de red de YOLOv4 y corriendo detecciones en tiempo real con una webcam de 640x480 Pixeles, obtenemos una velocidad de detección promedio de 1.3FPS, por lo cual, se descarta esta arquitectura debido a que es muy pesada para una computadora de las especificaciones mencionadas, es decir, en una Raspberry Pi podría tener un desempeño desastroso al momento de detectar objetos en tiempo real, su tiempo de respuesta sería demasiado elevado. Para ejecutar esta prueba se usaron los pesos de YOLOv4 [26].

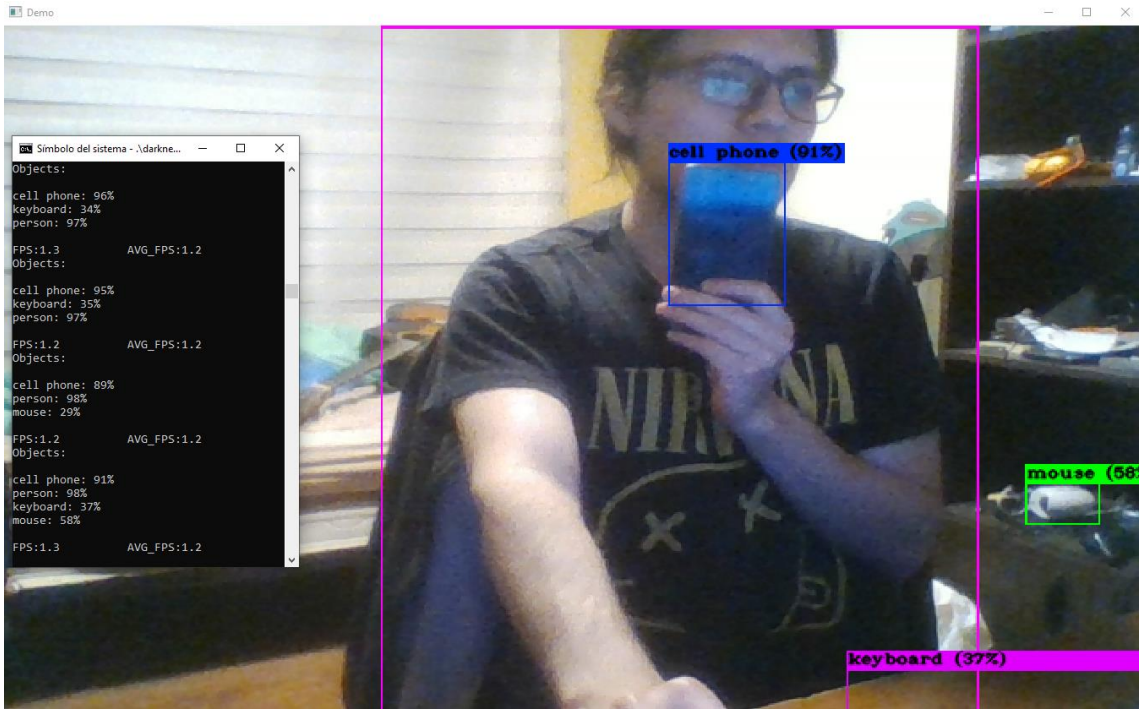


Figure 6 Detección de objetos en tiempo real Yolo v4 [Fuente: Los Autores]

Utilizando la arquitectura de red de YOLOv4-tiny y corriendo detecciones en tiempo real con la misma webcam mencionada anteriormente, se obtiene una velocidad de detección promedio de 13.3FPS, es más rápida y la razón es simplemente porque su arquitectura es más pequeña, en este caso la precisión promedio desciende, pero aun así es bastante aceptable. Los pesos para YOLOv4-tiny los podemos encontrar en [27].

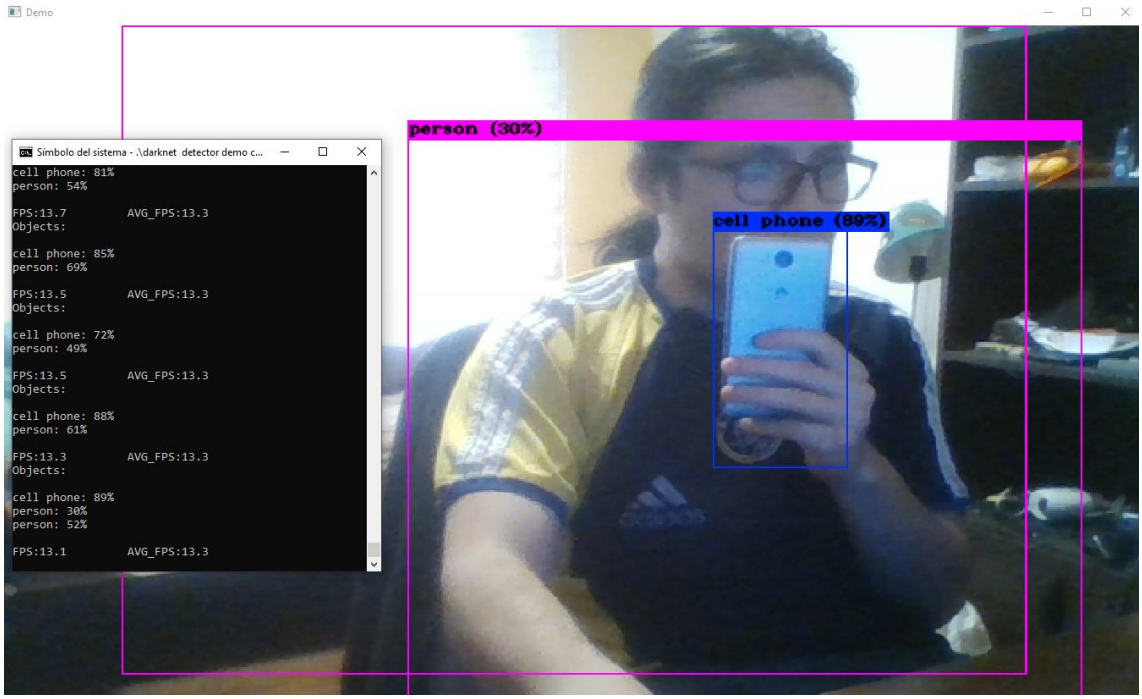


Figure 7 Detección de objetos en tiempo real Yolo v4 Tiny [Fuente: Los Autores]

En conclusión, para el desarrollo del modelo se usará la arquitectura de YOLOv4-tiny y se evaluaran los resultados en la Raspberry Pi 4.

2.3 Creación de la Base de Datos

2.3.1 Definición de Número de Clases a detectar

Debido a que el objetivo es detectar exclusivamente semáforos peatonales, entonces el número de clases que se detectarán serán 2, en este caso vendrán a ser Semáforo Peatonal Rojo y Semáforo Peatonal Verde, no se implementaron más clases ya que los semáforos peatonales solo cuentan con dos estados.

2.3.2 Recopilación de imágenes para la base de datos

Dentro de la web no existe una base de datos de imágenes de semáforos peatonales, es por este motivo se creó una base de datos de semáforos peatonales a partir de la obtención de fotografías de este objeto en varios puntos de la ciudad de Cuenca.

Se realizaron las fotografías con una cámara de celular que poseía una resolución de 5MP (2592x1936).

Para la toma de fotografías, no se especificó algún patrón especifico, pero en la mayoría de los casos dejar el semáforo peatonal en algunas de las zonas a continuación mostradas, además de capturar el semáforo desde una vista más o menos frontal hacia una vista más o menos isométrica, esta última en algunos casos.

Figure 8 Semáforo visto desde una perspectiva más o menos frontal e isométrica [Fuente los Autores]

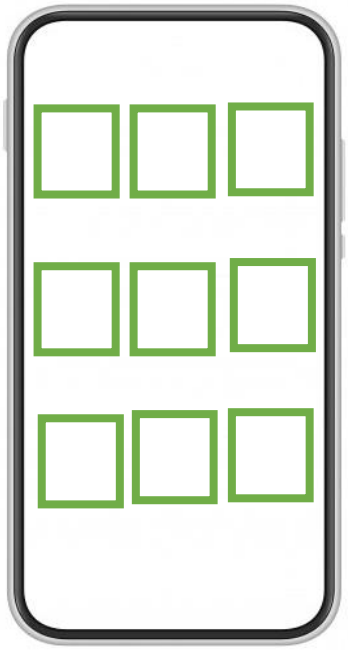


Figure 9 Zonas de la fotografía en las que aparece un semáforo con más recurrencia [Fuente los Autores]

2.3.3 Renombramiento de las imágenes recopiladas

Con el código de programación realizado en Python se cambia el nombre de las imágenes con la intención de organizar la información de forma ordenada, numérica y simple de entender.

```
import os
from glob import glob

path = "C:/Users/Francisco/Desktop/imagenes_tesis_nuevas_2"
inicio = 1

for item in glob(os.path.join(path, '*')):
    os.rename(item, os.path.join(path, "imagen_" +
str(inicio) + ".jpg"))
    inicio += 1
```

Figure 10 Script para el renombramiento de las Imágenes recuperadas [Fuente los Autores]

La variable “path” es la dirección en la que se encuentran las imágenes y la variable “inicio” es un contador que simplemente ira incrementando por cada imagen en la dirección de la variable path.

2.3.4 Etiquetado de la base de datos con el uso de labelImg

2.3.4.1 Instalación de labelImg

El labelImg tiene una interfaz gráfica completa y está escrita en Python y su información puede ser escrita en archivos XML [28]. Para la instalación se busca dentro de una carpeta llamada “labelImg” que se encontrará junto con las paqueterías necesarias para su correcto funcionamiento. Las instrucciones de código y el orden de ejecución de los mismos para instalar esta herramienta son los siguientes [29]:

```
Sudo apt-get installs pyqt5-dev-tools
Sudo pip3 install -r requirements/requirements-linux-python3.txt
Make qt5py3
python3 labelImg.py
```

Al ejecutar la última línea de este código, se abrirá una interfaz gráfica del labelImg.

Cabe mencionar que también existe una versión para Windows que es más sencilla de usar, simplemente se descarga labelImg para Windows del repositorio de Github y se ejecuta, pero se debe tener instalado Python en el sistema debido a que como ya mencionamos antes es un programa escrito en este lenguaje de programación. Se usaron ambas versiones y el resultado no difiere en lo absoluto.

```
francisco@francisco-Inspiron-5558:~/Escritorio/LabelImg/labelImg$ python3 labelImg.py
```

Figure 11 Comando para abrir la interfaz gráfica de labelImg [Fuentes los Autores]

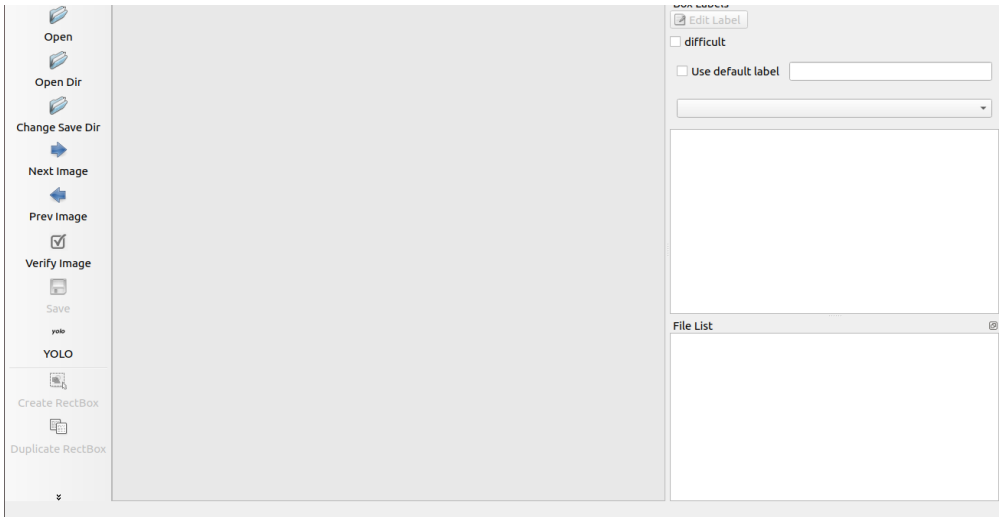


Figure 12. Interfaz gráfica de labelImg [Fuentes los Autores]

2.3.4.2 Tipo de Formato

En principio se crea las etiquetas para el primer modelo, que como se mencionó al inicio del capítulo, será el modelo basado en la arquitectura de YOLOv4-Tiny.

Es importante especificar el tipo de formato en el que se desea que se genere el archivo de salida, porque como ya se mencionamos antes, labelImg puede generar archivos XML que son incompatibles para el formato que necesita YOLO. Es por esto que se debe asegurar tener activado este formato en la barra lateral del programa.

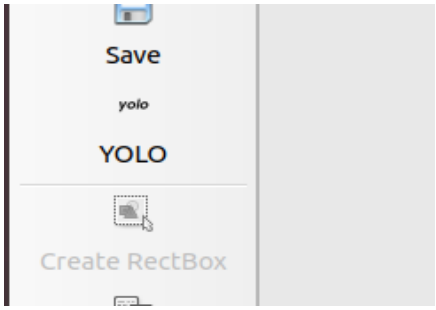


Figure 13 Formato YOLO dentro de LabelImg [Fuentes los Autores]

2.3.4.3 Configuración del nombre o tipo de clases dentro de labelImg

En la carpeta "data" dentro de "labelImg", se busca el "predefined_classes.txt". Dentro del .txt, se remplazan los valores que vengan por defecto por el nombre de las clases que desea usarse. Esta carpeta también se la puede encontrar en la versión para Windows con el mismo nombre y el mismo contenido.

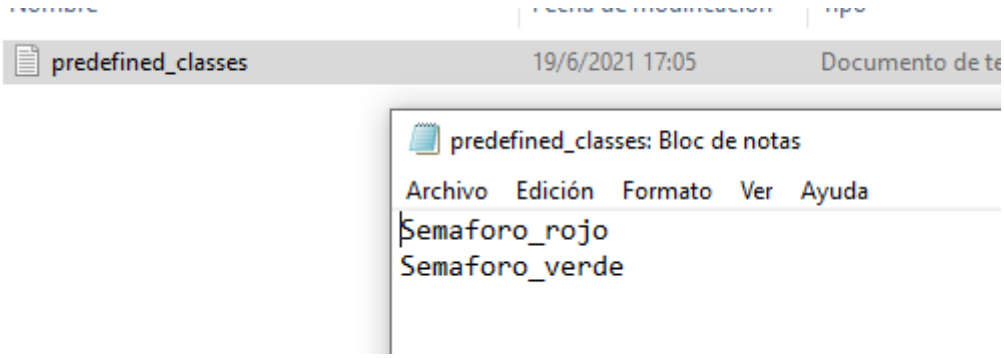


Figure 14 Definición del nombre de las clases en labelImg [Fuente Los Autores]

2.3.4.4 Etiquetado

Se abre el directorio en donde se ubica la base de datos obtenida de la recopilación de imágenes del apartado 2.3.4 y se procede a etiquetar dicha base para que la red pueda saber en qué posición se encuentra el objeto a detectar dentro de la imagen. Este proceso puede ser tedioso y más si se tiene una base de datos robusta, debido a que el etiquetado se realizara imagen por imagen de forma manual.

Para este procedimiento se usa la opción “Create Rectbox”, que se encuentra dentro de la interfaz gráfica de “labelImg”. Esto permitirá trazar el “bounding box” para obtener los valores de la zona de interés dentro de la imagen, es decir, nos permitirá aislar el objeto del resto de la imagen. Una vez que se crea el “bounding box”, se procede a asignar una clase de las opciones que “labelImg” nos presenta y que serán las que configuramos en el apartado anterior, seleccionamos la que corresponda con la imagen.

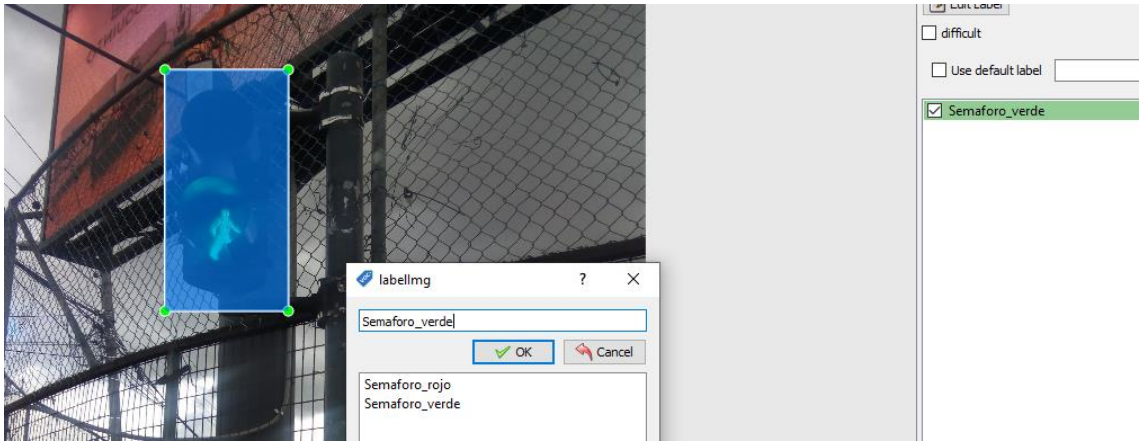


Figure 15 Creación del bounding box y asignación de la clase correspondiente [Fuentes Los Autores]

La herramienta “labelImg”, permitirá ir generando un archivo con el mismo nombre de la imagen, pero con extensión “.txt” debido a que estamos usando el formato de salida para YOLO que tendrá un contenido parecido al siguiente formato:

Id_clasificador x y w h

En donde Id_clasificador será el identificador de la clase comenzando por 0 de lo que representa el área delimitada o zona de interés. Las variables x, y, w, h las veremos en el siguiente apartado como características del “bounding box” o cuadro delimitador.

2.3.4.5 Características del “Bounding Box”

X, Y, W y H son las coordenadas dentro del “bounding box” (x, y), el ancho (w) y el alto (h) del “bounding box”. Es importante aclarar que estos valores vienen en porcentajes, técnicamente son valores que varían entre 0 y 1, debido a que se encuentran proporcionadas en función del ancho y alto de la imagen con el fin de que esas coordenadas sean siempre las mismas sin importar que el tamaño de la imagen cambie.

2.3.5 Contador de imágenes etiquetadas por clase

No todas las imágenes recopiladas fueron etiquetadas por varias razones, algunas imágenes se encontraban demasiado borrosas debido a un mal enfoque o por un mal contraste de la foto, en muchas imágenes no se apreciaba el semáforo peatonal porque estaba demasiado oscuro o porque el brillo del sol opacaba el semáforo peatonal.

Para llevar un mejor control sobre los datos que se posee, se creó un pequeño algoritmo para contar la cantidad de etiquetas por clase que se tenía hasta ese momento.

```
import os
from glob import glob

path = "C:/Users/Francisco/Desktop/DECIMO
CICLO/TESIS/ETIQUETAS_TOTALES_FINALES"
path_labels_si = os.path.join(path, 'imagen_*')
contador = 0
contador_rojos = 0
contador_verdes = 0

for element in glob(path_labels_si):
    f = open(element, "r")
    color = f.read(1)
    if color == "0":
        contador_rojos += 1
    elif color == "1":
        contador_verdes += 1
    else:
        print("No es 0 ni 1")
        print(element)
    contador += 1

print("Total: %s \nSemaforos verdes: %s \nSemaforos rojos:
%s " % (contador, contador_verdes, contador_rojos))
```

Figure 16 Algoritmo para conteo de etiquetas por clase [Fuente: Los Autores]

El algoritmo es simple y hace lectura de la primera línea y la primera columna de los archivos ".txt" generados en labelImg que si bien recordamos tenían en esta posición el "Id_identificador" que como sabemos es el número de la clase. Entonces lee cada archivo ".txt" encontrado en la variable "path" y aumenta el contador, ya sea para semáforos rojos o verdes, dependiendo de qué valor tengan estos en la primera línea y la primera columna de su contenido. La sentencia "else" nos sirve para ver que archivo ".txt" pudo haber pasado sin los datos del bounding box y de la clase.

Al final se obtuvo un total de 2065 etiquetas que corresponden a las 2065 imágenes de las cuales 958 son de semáforos verdes y 1107 son de semáforos rojos.

```
Run:  main
C:\Users\Francisco\AppData\Local\Progra
Total: 2065
Semaforos verdes: 958
Semaforos rojos: 1107

Process finished with exit code 0
```

Figure 17 Resultado Obtenido después de ejecutar el algoritmo [Fuentes los Autores]

2.3.6 Descarte de imágenes sin etiqueta

Como se mencionó anteriormente, no todas las imágenes recopiladas fueron etiquetadas. Para capturar las 2065 imágenes que si tienen etiqueta se ejecuta el siguiente algoritmo creado con el propósito de separar las imágenes en otro directorio, es simple de entender y bastante practico.

```
from glob import glob
import shutil, os
from pathlib import Path

path_imagenes = "C:/Users/Francisco/Desktop/imagenes_tesis_nuevas_2"
path_labels = "C:/Users/Francisco/Desktop/DECIMO CICLO/TESIS/labels_si_3"
path_imagenes_destino = "C:/Users/Francisco/Desktop/DECIMO
CICLO/TESIS/imagenes_tesis para el data aumenting"
array = []

for elemento in glob(os.path.join(path_labels, "imagen_*")):
    array.append(Path(elemento).stem)

print(array)

for element in glob(os.path.join(path_imagenes, "imagen_*")):
    print(element)
    nombre = Path(element).stem
    if nombre in array:
        shutil.move(element, path_imagenes_destino)
```

Figure 18 Algoritmo para eliminación de imágenes sin etiqueta [Fuente: Los Autores]

La librería “pathlib” tiene el propósito de usar la función “stem” que esta posee para obtener el nombre de un archivo sin su extensión. Se crea una lista vacía y la llenaremos con el nombre de las etiquetas que se encuentren en la dirección definida por la variable “path_labels” y moverá aquellas imágenes (dirección definida en variable “path_imagenes”) que tengan un nombre igual al de la lista creada a una dirección destino definido en la variable “path_imagenes_destino”.

El resultado será una carpeta con las 2065 imágenes que tienen una correspondiente etiqueta.

2.4 Aumento de Datos

Como recomendación, en el repositorio de Github de Alex [25]se especifica que para llegar a tener una buena detección de objetos se debe tener al menos 2000 diferentes imágenes por clase, si es que se va a utilizar alguna arquitectura de YOLO.

Ya que el número de imágenes disponibles en este punto es de 2065, lo cual es preocupante debido a que se necesitaría al menos 2000 imágenes por clase. Partiendo de este punto es en donde surge la necesidad de usar "data augmentation" o aumento de datos, en español, para incrementar el tamaño de nuestra base de datos sin la necesidad de salir a fotografiar más semáforos peatonales por toda la ciudad.

Se pueden utilizar varias técnicas para el aumento de datos, en este caso, optamos por dos de las más sencillas, Flip y Rotación.

Flip

Consiste en voltear las imágenes horizontalmente y verticalmente, para esta operación utilizaremos OpenCV. [30]

Figure 19 Fig. Imagen Original (Izquierda) e Imagen Volteada (Derecha) [Fuente: Los Autores]

Rotación

Esta operación es más compleja que la anterior porque tiene un punto clave a tomar en cuenta y es que si se rota la imagen en ángulos rectos, siempre se preservara las dimensiones de la imagen, mientras que, si se rota la imagen en ángulos más finos, por ejemplo 45° o 2° es muy posible que veamos espacios "huecos" rellenados con un color negro en la imagen resultante. [30]

Para evitar este inconveniente y que la imagen trate de salir lo más natural posible después de la rotación, usaremos un método de rellenado en la imagen para los espacios "huecos" de color negro que se formaran después de la rotación. Puede existir varios métodos de rellenado y en nuestro caso usaremos el método "Reflect".

Antes de pasar a revisar el método "Reflect" cabe mencionar que los ángulos de rotación aplicados a las imágenes oscilan entre -15° y 15°, es decir, existe una brecha de 30 grados que irán iterando por cada imagen que entre al algoritmo de data augmentation.

Reflect

Este método consta en reflejar los valores de los pixeles de la imagen a lo largo del límite de la imagen, es decir, el límite que separa la imagen de estos espacios "huecos" de color negro. [30]

El resultado de utilizar rotación con el método "reflect" fue satisfactorio y se observa en la siguiente imagen.

Figure 20 Fig. Imagen Original (Izquierda) e Imagen Rotada -15° y aplicando el método "reflect" para conservar las dimensiones de la imagen (derecha) [Fuente: Los Autores]

Algoritmo para el aumento de datos escrito en Python

```
import os
import cv2
import skimage.transform
from skimage import io
from glob import glob

images_path= 'C:/Users/Francisco/Desktop/DECIMO CICLO/TESIS/imagenes_tesis para el data
aumenting'
images_path_destino= 'C:/Users/Francisco/Desktop/DECIMO CICLO/TESIS/DATA AUMENTING'
angulo = -15

# Giro y Rotacion -- Triplica la data (x3)
for im in os.listdir(images_path):
    if angulo > 15:
        angulo = -15
    # Parametros
    path = os.path.join(images_path + '/' + im)
    name = os.path.splitext(im)[0]
    direccion_flip = os.path.join(images_path_destino + '/' + name + '_flip' + '.jpg')
    direccion_rot = os.path.join(images_path_destino + '/' + name + '_rot' + '_' + str(angulo) + '.jpg')

    # Procesos
    imagen = cv2.imread(path)
    imagen_2 = io.imread(path)
    flip = cv2.flip(imagen, 1)
    # cv2.imwrite(direccion_flip,flip)
    ancho = imagen.shape[1]  # columnas
    alto = imagen.shape[0]  # filas
    centro = (ancho // 2, alto // 2)
    # M=cv2.getRotationMatrix2D(centro,10,1)
    # rot=cv2.warpAffine(imagen,M,(ancho,alto))
    rot = skimage.transform.rotate(imagen_2, angle=angulo, center=centro, mode='reflect')  # rotacion
antihoraria
    cv2.imwrite(direccion_flip, flip)
    io.imsave(direccion_rot, rot)
    print(angulo)
    angulo += 1
    # cv2.imwrite(direccion_rot,rot)
```

Figure 21 Algoritmo para aumento de datos escritos en Python [Fuente: Los Autores]

El algoritmo utiliza OpenCV para el volteo de las imágenes y la librería skimage para la rotación de las mismas, esta librería se utilizó por ser muy práctica y facilitar los métodos de rellenado, en este caso, el método “reflect”

Como se puede observar en el algoritmo, el ángulo va iterando conforme avanza la lectura de imágenes y esto puede ayudar a generar una base de datos aumentada con mayor flexibilidad a detectar los objetos a diferentes ángulos de rotación de la cámara, siempre y cuando estos no sean excesivos.

El resultado será un total de 3982 imágenes nuevas a las que se les deberá realizar el mismo proceso antes visto en el apartado 2.3.4.4 hasta el apartado 2.3.6, y se incluye el apartado 2.3.6 (Descarte de imágenes sin etiqueta) porque debido al proceso de rotación, es posible que el objeto salga de la imagen o parte del objeto y no sea una buena opción a etiquetar, entonces será necesario un descarte posterior.

Una vez después de etiquetar las imágenes se deberán colocar las 3982 imágenes nuevas junto con las 2065 imágenes obtenidas en el primer proceso de etiquetado para sumar un total de 6047 imágenes, y de igual forma con las etiquetas, deberán ser un total de 6047 etiquetas.

2.5 Desarrollo del modelo con YOLOv4-Tiny

2.5.1 Descarga de Darknet

El primer paso antes de empezar con la preparación de los archivos es crear una carpeta con el nombre "modelo1YOLOv4-Tiny" (puede llevar cualquier nombre) que contendrá todos los archivos necesarios para construir nuestro modelo.

Como estamos usando Windows, deberemos descargar la versión de darknet de la fuente [31], descomprimimos el archivo y obtendremos una carpeta con el nombre "darknet-master", por conveniencia le cambiaremos el nombre a simplemente "darknet".

2.5.2 Creación y activación del entorno virtual

Una buena práctica es crear un entorno virtual en el que vayamos a instalar todas nuestras dependencias y librerías para nuestro proyecto, de forma que en un futuro no exista un problema de conflicto de versiones o por la actualización de algún paquete el programa deje de funcionar.

Empezamos creando el entorno virtual con Python:

```
Python -m venv vemodelo1YOLOv4-Tiny
```

Ahora activaremos el entorno virtual para proceder con la instalación del resto de librerías y dependencias necesarias, para activarlo usar el comando (Windows):

```
.\vemodelo1YOLOv4-Tiny\Scripts\activate
```

Si deseáramos desactivarlo por alguna razón solo debemos usar el "deactivate":

```
.\vemodelo1YOLOv4-Tiny\Scripts\deactivate
```

2.5.3 Instalación y verificación de CUDA (Opcional)

Darknet en la CPU es rápido, pero en una GPU es mucho más rápido, para esto se necesita tener una GPU NVIDIA y tener instalado CUDA. [11]

Se utiliza una NVIDIA Geforce 920M con una capacidad de 4GB. A continuación, se muestra el procedimiento para la instalación y la verificación de la correcta instalación de CUDA.

Recalcando, este paso se lo realiza debido a que se dispone de una GPU NVIDIA a la cual se le puede sacar mucho provecho en el entrenamiento de la red, en caso de no disponer de una GPU NVIDIA, se omite este paso.

1. Se verifica que la computadora tenga integrada una GPU NVIDIA consultando en la sección "Adaptadores de Pantalla" de Administrador de dispositivos.

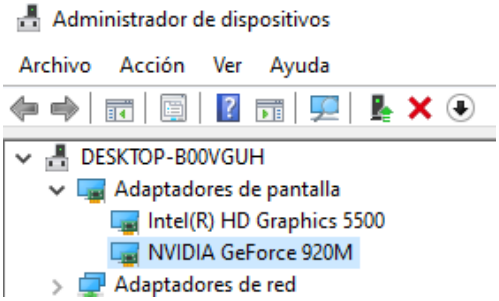


Figure 22 Verificación para saber que GPU se tiene en la computadora [fuente: Los Autores]

2. Es importante conocer cuál es la última versión del driver de nuestra GPU, es nuestro caso es la versión 425.31, compatible con la versión de CUDA 10.1, para verificar esto podemos escribir en la consola de Windows el comando "nvidia-smi". Para ejecutar este comando se debe tener agregado la dirección en donde se encuentre el ejecutable "nvidia-smi.exe", en nuestro caso es "C:\Program Files\NVIDIA Corporation\NVSMI".

```
C:\Users\fbrit>nvidia-smi
Wed Jul 28 16:42:12 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 425.31       Driver Version: 425.31       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce 920M       WDDM  | 00000000:08:00.0 N/A |                  N/A |
| N/A   44C    P8    N/A /  N/A |     41MiB /  4096MiB |     N/A      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0                    Not Supported                                       |
+-----------------------------------------------------------------------------+
```

Figure 23 Verificación de la versión de CUDA instalada y versión de controladores de la tarjeta instalados antes de actualizar los drivers [Fuente: Los Autores]

Para asegurar no instalar la versión de CUDA incorrecta se puede revisar la tabla que NVIDIA ofrece en la fuente [32]. En este caso debido a que la GPU no dispone de un driver de mayor versión que soporte CUDA 11.x, se opta por trabajar con la versión de CUDA 10.1.

3. Descargamos e instalamos la versión de CUDA 10.1 Update 2 [33], para obtener más detalles sobre los pasos para la instalación consultar [34]. NOTA: Esta versión de CUDA nos actualizara los drivers de la GPU de la versión 425.31 a la versión 426.0

```
C:\Users\fbrit>nvidia-smi
Wed Jul 28 21:41:07 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 426.00       Driver Version: 426.00       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce 920M       WDDM  | 00000000:08:00.0 N/A |                  N/A |
| N/A   43C    P8    N/A /  N/A |     41MiB /  4096MiB |     N/A      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0                    Not Supported                                       |
+-----------------------------------------------------------------------------+
```

Figure 24 Verificación de la versión de CUDA instalada y versión de controladores de la tarjeta instalados después de actualizar los drivers [Fuente: Los Autores]

4. Finalizada la instalación, ejecutamos el comando “nvcc -V” y verificamos la versión de CUDA instalada.

```
C:\Users\fbrit>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Apr_24_19:11:20_Pacific_Daylight_Time_2019
Cuda compilation tools, release 10.1, V10.1.168
```

Figure 25 Verificación de la versión de CUDA [Fuente: Los Autores]

5. Abrimos el proyecto de prueba “deviceQuery_vs2019.vcxproj” dentro del directorio “C:\ProgramData\NVIDIA Corporation\CUDA Samples\v10.1\1_Utilities\deviceQuery” que nos dejara la instalacion de CUDA. En este caso se utiliza Microsoft Visual Studio 2019 IDE para abrir el proyecto y se procede a compilarlo y ejecutarlo. Si todo se ha ejecutado correctamente se abre una ventana como la que se muestra a continuacion, que indica que se logrado una correcta configuracion entre el hardware y el software. Lo importante es observar que se ha detectado la tarjeta y que se ha pasado la prueba (Result = PASS).

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v10.1\bin\win64\Debug\deviceQuery.exe Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce 920M"
  CUDA Driver Version / Runtime Version          10.1 / 10.1
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 4096 MBytes (4294967296 bytes)
  ( 2) Multiprocessors, (192) CUDA Cores/MP:     384 CUDA Cores
  GPU Max Clock rate:                            954 MHz (0.95 GHz)
  Memory Clock rate:                             900 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 524288 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 8 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.1, CUDA Runtime Version = 10.1, NumDevs = 1
Result = PASS
```

Figure 26 Verificación de la detección correcta del hardware NVIDIA por CUDA. [Fuente: Los Autores]

6. Para verificar que exista una correcta comunicación entre el sistema y el dispositivo compatible con CUDA, se ejecuta una prueba más, esta se la puede encontrar con el nombre "bandwidthTest_vs2019.vcxproj" en la dirección "C:\ProgramData\NVIDIA Corporation\CUDA Samples\v10.1\1_Utilities\bandwidthTest", de igual forma que la prueba anterior, se abre el proyecto, se compila y se ejecuta. Si todo se ejecuta correctamente se obtiene una ventana como la que se muestra a continuación.

```
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce 920M
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     1.6

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     1.7

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     12.8

Result = PASS
```

Figure 27 Verificación correcta de la comunicación del hardware NVIDIA y el software CUDA [Fuente: Los Autores]

2.5.4 Instalación de CUDNN (Solo si se instaló CUDA)

Se instala cuDNN v7.6.5 (Noviembre 5th, 2019), para CUDA 10.1. Actualmente se tiene versiones nuevas de cuDNN para la versión de CUDA, pero se utiliza está en especial debido a que forma parte de la combinación de configuraciones de compilación probadas por TensorFlow [35]. Esto será relevante cuando se haga la conversión de los pesos a TensorFlow.

1. Descargar la versión de cuDNN que mencionamos antes de la fuente [36].
2. Descomprimir el archivo y copiamos los archivos que se encuentran dentro de las carpetas “bin”, “include” y “lib” a las carpetas con el mismo nombre en la dirección “C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1”.
3. Agregar a la variable de entorno “Path” las siguientes direcciones, suponiendo que <INSTALL_PATH> = < C:\Program Files>:
   - <INSTALL_PATH> \NVIDIA GPU Computing Toolkit\CUDA\v10.1\bin
   - <INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\libnvvp
   - <INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\extras\CUPTI\lib64

2.5.5 Instalación y verificación de OpenCV en Windows utilizando CMake

YOLO utiliza herramientas de OpenCV para realizar detecciones de objetos en tiempo real a través de una cámara, de forma que es indispensable para este proyecto tenerla instalada.

Debido a que más adelante se debe compilar "Darknet" en Windows se necesita proporcionarle a esta compilación algunos archivos que generaremos ahora con las versiones de fuente de OpenCV. Para esta instalación se desactivo el entorno virtual, es decir, se instala OpenCV directamente en el sistema, al igual que Numpy.

1. Instalar CMake, en este caso instalaremos la versión 3.21.1 para Windows "cmake-3.21.1-windows-x86_64.msi" [37], CMake GUI.
2. Descargar la librería OpenCV de fuente (Sources) [38] dentro de la carpeta "modelo1YOLOv4-Tiny" y extraer los documentos en la misma dirección.
3. Dentro de la carpeta "opencv-4.5.3" crear una nueva carpeta llamada "build".
4. Siguiendo los pasos recomendados por OpenCV. [39], es importante que para continuar tener el módulo de Python "Numpy" instalado, pero en el entorno principal del sistema.

```
C:\Users\fbrit>pip install numpy
Collecting numpy
  Using cached numpy-1.21.1-cp39-cp39-win_amd64.whl (14.0 MB)
Installing collected packages: numpy
Successfully installed numpy-1.21.1
WARNING: You are using pip version 21.1.3; however, version 21.2.1 is available.
You should consider upgrading via the 'c:\users\fbrit\appdata\local\programs\pyt
-upgrade pip' command.
```

Figure 28 Instalación de Numpy en el sistema [Fuente: Los Autores]

```
Símbolo del sistema - python
Microsoft Windows [Versión 10.0.19043.1110]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\fbrit>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> import cv2
>>>
>>>
>>> print(cv2.__version__)
4.5.3
```

Figure 29 Verificación de la versión de OpenCV instalada en el sistema [Fuente: Los Autores]

2.5.6 Configuración y compilación de Darknet en Windows usando CMake

Existe varias formas de compilar "Darknet" y esto depende principalmente del sistema operativo que se esté usando, en este caso, para el entrenamiento del modelo se

optara por usar Windows y por ende los pasos a seguir para compilar este framework los podemos encontrar en [25], en la sección “How to compile on Windows (using CMake)”.

El primer paso antes de empezar con la preparación de los archivos es crear una carpeta con el nombre “modelo1YOLOv4-Tiny” (puede llevar cualquier nombre) que contendrá todos los archivos necesarios para construir nuestro modelo.

Como se utiliza Windows, hay que descargar la versión de darknet de la fuente [31], descomprimir el archivo y obtener una carpeta con el nombre “darknet-master”, por simple conveniencia se le cambia el nombre a simplemente “darknet”.

Dentro de esta carpeta se ejecuta los siguientes pasos:

1. Descargar e Instalar Microsoft Visual Studio, en esta ocasión se estará utilizando la versión 2019.
2. Abrir CMake GUI y ejecutar el programa en el siguiente orden indicado

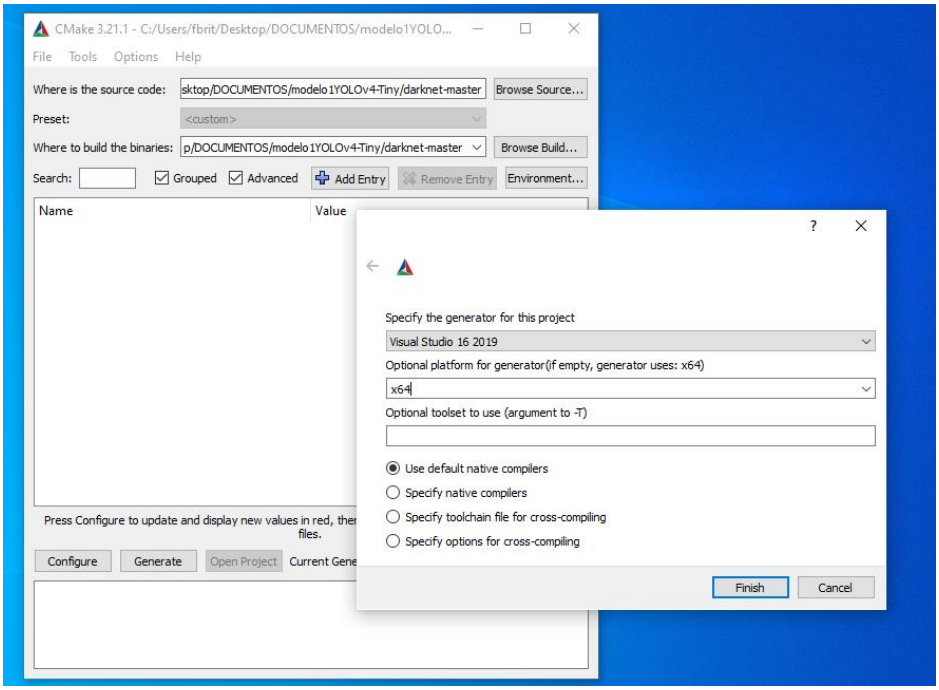


Figure 30 Entorno grafico de CMake 3.21.1 GUI [Fuente: Los Autores]

- Paso 1: Cargar la dirección en donde se encuentra la fuente “darknet-master” y subir la dirección de salida en donde se construirán los binarios, en este caso es la misma dirección
- Paso 2: Se da un click en el botón “Configure”

- Paso 3: Especificar el generador para el proyecto, en nuestro caso será Microsoft Visual Studio 16 2019 y opcionalmente poder especificar el tipo de plataforma con el que generara, en este caso es x64.
- Paso 4: Se da un click en “Finish”.
- Paso 5: Se da un click en “Generate”
- Paso 6: Se da un click en “Open Project”

Si arroja un error diciendo que no puede encontrar el archivo “OpenCVConfig.cmake” lo que se debe hacer es abrir el archivo “CMakeLists” que se encuentra dentro de la carpeta “darknet” y escribir la siguiente línea de código

Set (CMAKE_PREFIX_PATH "C:/Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/opencv-4.5.3/build")

Con esto se indica dónde buscar el archivo “OpenCVConfig.cmake” que necesita para compilar de forma exitosa. Esto se lo hace debido a que Darknet no trae ese archivo por defecto, así que se debe indicar dónde buscarlo. Como se puede apreciar este archivo es parte del resultado cuando se compila e instala OpenCV.

3. Terminado el proceso de configuración, se genera dando en la opción “Generate” y luego al botón “Open Project”. Se debe verificar que en la sección “ENABLE” se encuentren activadas las siguientes opciones, esto si se está usando CUDA y OPENCV.

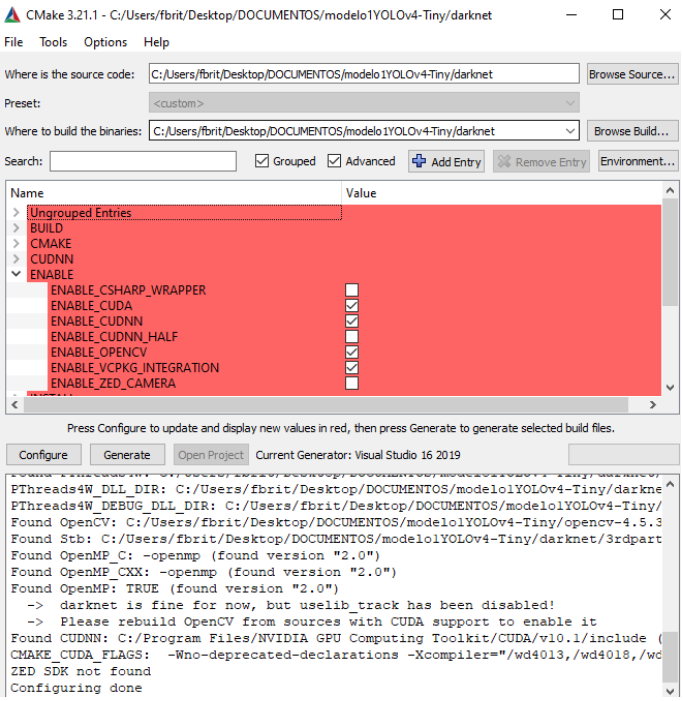


Figure 31 Resultados generados después de configurar CMake (marcados en rojo) [Fuente: Los Autores]

4. Abierto el Proyecto se debe asegurar de tener las siguientes opciones activadas

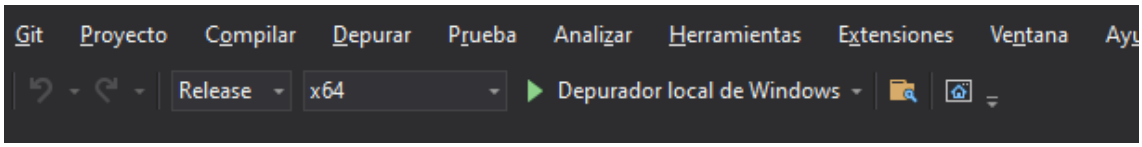


Figure 32 Opción de plataforma de solución en "x64" y configuración de solución en "Release" [Fuente: Los Autores]

5. En la pestaña "Compilar" se selecciona la acción "Compilar solución", esto ejecutara la compilación de todos los 6 proyectos, y cuando termine dejará al final en la ventana de salida lo siguiente:

```
4>cl /c /I"C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\include" /I"C:\Us
4>yolo_console_dll.cpp
4>uselib.vcxproj -> C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\Release\
5>------ Operación Compilar iniciada: proyecto: ALL_BUILD, configuración: Release x64 ---
5>Building Custom Rule C:/Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/darknet/CMake
6>------ Operación Compilar omitida: proyecto: INSTALL, configuración: Release x64 ------
6>Proyecto no seleccionado para compilarse para esta configuración de solución
========== Compilar: 5 correctos, 0 incorrectos, 0 actualizados, 1 omitidos ==========
```

Figure 33 Resultado devuelto por la terminal de Visual Studio al terminar la compilación. [Fuente: Los Autores]

Lo importante es encontrar el ejecutable "darknet.exe", en este caso, se encuentra dentro de la dirección "darknet/Release" que se crea después de haber realizado los pasos anteriores, con algunos archivos adicionales.

Se deberán copiar todos estos archivos que se encuentran en la carpeta “Release” y se los copiara en la dirección “darknet\build\darknet\x64” y desde esa dirección se ejecutara “darknet.exe”. [40]

Además, se tuvo que agregar una dirección a la variable de entorno “Path” debido a que al ejecutar el “darknet.exe” genera errores de DLL’s, al parecer “Darknet” no podía encontrar algunos “.dll” de OpenCV. Para esto se agrega la siguiente dirección a la variable de entorno “C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\opencv-4.5.3\build\bin\Release”, de esta forma, Darknet podrá encontrar los DLL que solicita para ejecutar “darknet.exe”.

Otro error que genera es la falta de otro DLL que no tenía relación con OpenCV, este dll fue “pthreadVC2.dll” y se utiliza la solución propuesta por AlexeyAB para erradicar estas dificultades [41], que es simplemente pasar los DLL’s de la dirección “C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\3rdparty\pthreads\bin” a la dirección “C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet”

Para comprobar que Darknet se encuentra correctamente instalado se ejecuta el comando “.\darknet.exe” dentro de la consola de Windows en la dirección “darknet\build\darknet\x64”, y el resultado deberá ser el siguiente, según [42]

```
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>darknet.exe
usage: darknet.exe <function>
```

Figure 34 Resultado correcto de la ejecución del comando “darknet.exe” [Fuente: Los Autores]

A continuación, se debe ejecutar una prueba para verificar que hasta el momento todo esté funcionando de forma correcta, para ello se requiere descargar el archivo de pesos para YOLOv4 y se coloca dentro de la carpeta “x64”.

Se realizan detecciones en tiempo real para verificar que no exista ningún problema, usamos el comando:

darknet.exe detector demo cfg\coco.data cfg\yolov4.cfg yolov4.weights

```
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>darknet.exe detector demo cfg\coco.data c
fg\yolov4.cfg yolov4.weights
 CUDA-version: 10010 (10010), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 4.5.3
Demo
 0 : compute_capability = 350, cudnn_half = 0, GPU: GeForce 920M
net.optimized_memory = 0
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    608 x 608 x   3 ->  608 x 608 x  32 0.639 BF
   1 conv     64       3 x 3/ 2    608 x 608 x  32 ->  304 x 304 x  64 3.407 BF
   2 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   3 route  1                                     ->  304 x 304 x  64
   4 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   5 conv     32       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  32 0.379 BF
```

Figure 35 Inicio de ejecución de la detección en tiempo real usando la webcam de la computadora [Fuente: Los Autores]

Se detecta la GPU, la versión de CUDA, cuDNN, OpenCV instaladas en apartados anteriores. Adicionalmente se debe ejecutar YOLOv4 solo para comprobar que todo estuviera en orden y como dato importante a resaltar son los FPS Promedio (AVG_FPS) de 1.3. Para verificar el desempeño de la GPU se verifica en el administrador de tareas y en la pestaña "Rendimiento" se observa el trabajo de la GPU mientras realiza las detecciones con YOLOv4, un modelo que, para esta GPU, resulta muy pesado.

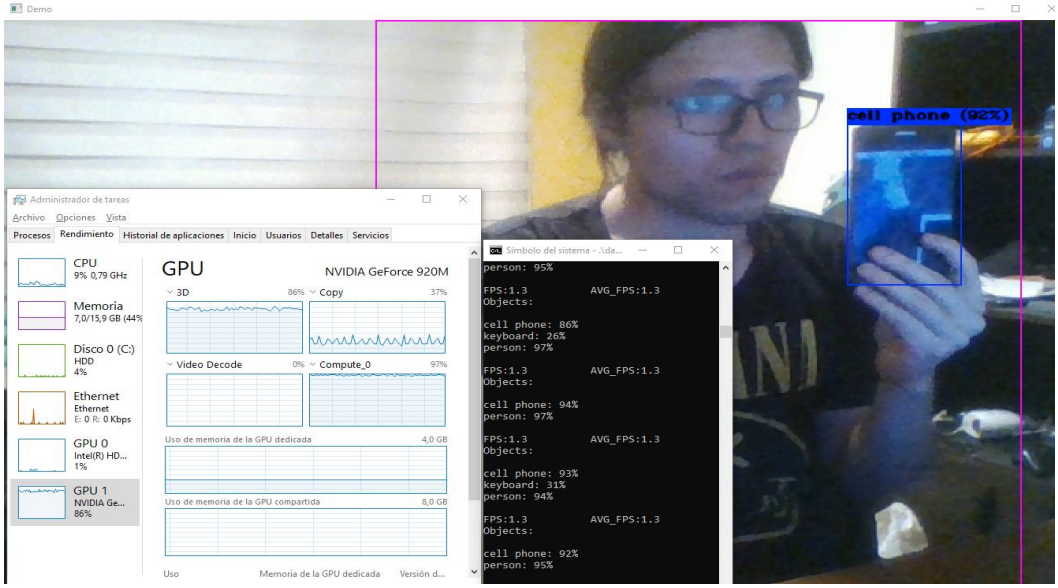


Figure 36 Detección en tiempo real usando el modelo YOLOv4 [Fuente: Los Autores]

Realizando el mismo paso anterior con la arquitectura de YOLOv4-tiny y usando el comando:

darknet.exe detector demo cfg\coco.data cfg\yolov4-tiny.cfg yolov4-tiny.weights

El trabajo de la GPU en las detecciones en tiempo real con esta arquitectura que es más liviana y por esta razón es preferible esta configuración para el desarrollo del primer modelo.

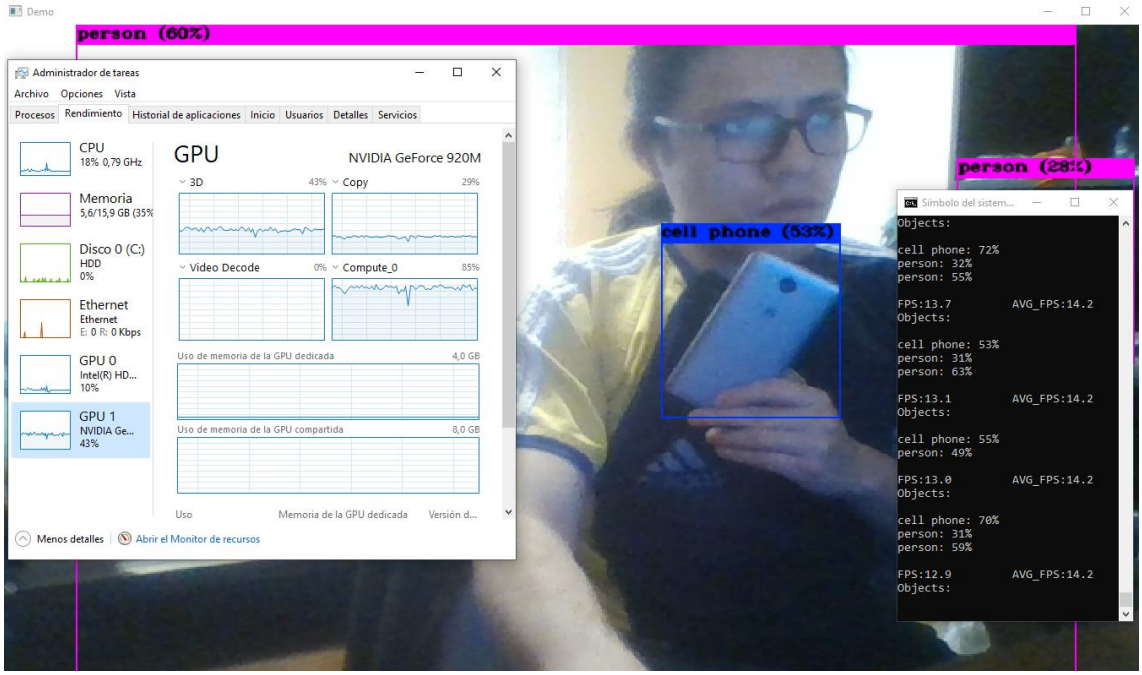


Figure 37 Detección en tiempo real usando el modelo YOLOv4-Tiny [Fuente: Los Autores]

2.5.7 Creación del set de entrenamiento y el set de prueba

Se procede a dividir el conjunto de datos total obtenido hasta el momento, es decir las 6047 imágenes con sus etiquetas correspondientes, en dos conjuntos de datos, el conjunto de datos de entrenamiento y de prueba.

Se destinará el 80% del conjunto de datos total al conjunto de datos de entrenamiento y el 20% restante al conjunto de datos de prueba.

Crear un algoritmo escrito en Python para realizar dicha operación usando especialmente la función “train_test_split” de la librería sklearn.model_selection.

El objetivo es crear dos archivos de texto (train.txt y test.txt) que serán los encargados de contener las direcciones de las imágenes tanto de entrenamiento como las de prueba. Esto es debido a que YOLO solicitara estas direcciones para el entrenamiento y evaluación del modelo. [25]

Para utilizar el siguiente algoritmo es necesario crear una carpeta con el nombre “custom” dentro de la dirección “C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data”. Dentro de la carpeta “custom” hay que dirigir la

carpeta que contiene las 6047 imágenes y se cambia el nombre de la carpeta a “images” para evitar tener que cambiar las direcciones en el código.

El siguiente código se debe ejecutar con el archivo .py dentro de la dirección “C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet” (ruta en donde se encuentra la carpeta “darknet”), para que pueda trabajar con las direcciones de forma correcta y crear los archivos “.txt” de forma satisfactoria. Para ejecutar el código simplemente se dirige a una terminal de Windows y se ejecuta el comando:

```
(vemodelo1YOLOv4-Tiny) C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet>python Split_train_test.py

(vemodelo1YOLOv4-Tiny) C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet>
```

Figure 38 Ejecución del script “Split_train_test.py” [Fuente: Los Autores]

Es importante tener el entorno virtual activado, caso contrario podría generar errores por ausencia de alguna librería.

```
Import os
From sklearn.model_selection import train_test_split
Savepath = 'build/darknet/x64/data/custom/'
path_img= 'build/darknet/x64/data/custom/images'
lst_img=os.listdir (path_img)
train_img, test_img=train_test_split (lst_img, test_size=0.2, random_state=3)
bloc_file_train= savepath+'/train.txt'
bloc_file_test= savepath+'/test.txt'
#ENTRENAMIENTO
With open (bloc_file_train,'w') as f:
   For item in train_img:
      f.write (path_img+'/'+item+'\n')
   f.close ()

# VALIDACION

With open (bloc_file_test,'w') as f:
   For item in test_img:
      f.write (path_img+'/'+item+'\n')
   f.close ()
```

test: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
data/custom/images/imagen_1179.jpg
data/custom/images/imagen_413_rot_-7.jpg
data/custom/images/imagen_1016_flip.jpg
data/custom/images/imagen_3792_flip.jpg
data/custom/images/imagen_3752_flip.jpg
```

train: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
data/custom/images/imagen_3388_rot_-7.jpg
data/custom/images/imagen_3101_flip.jpg
data/custom/images/imagen_4224_rot_-4.jpg
data/custom/images/imagen_557_flip.jpg
data/custom/images/imagen_522_rot_14.jpg
data/custom/images/imagen_2761_flip.jpg
data/custom/images/imagen_3341.jpg
```

Figure 39 Contenido de los archivos .txt después de ejecutado el código de Python [Fuente: Los Autores]

2.5.8 Creación del archivo ".names"

Dentro del archivo ".names" estará escrita el nombre de la clase que se desea que nos etiquete YOLO cuando haya realizado la detección, es decir, al momento de detectar un objeto escriba una etiqueta encima del "bounding box", este escribirá lo que se encuentre en el archivo ".names", siendo la primera línea para la clase 0 y de aquí en adelante para el resto de las clases.

Este archivo se crea en la dirección "'build/darknet/x64/data" con el nombre "semaforos_peatonales.names" con las clases que nos impuso en el principio.

semaforos_peatonales.names ×

C: > Users > fbrit > Desktop > DOCUMENTOS > modelo1YOLOv4-Tiny > darknet > build > darknet > x64 > data > semaforos_peatonales.names

```
1  Semaforo_rojo
2  Semaforo_verde
```

Figure 40 Contenido del archivo "semaforos_peatonales.names" [Fuente: Los Autores]

2.5.9 Creación del archivo .data

Se crea un nuevo archivo ".data" en la dirección "'build/darknet/x64/data" al cual se lo llama de una manera que se pueda identificar, en este caso "semaforos_peatonales.data" dentro de este archivo se va a configurar los valores de clases y direcciones de entrenamientos que son: train, labels; valid y backup.

- El parámetro "classes" hace referencia a cuantas clases se van a detectar.
- El parámetro "train" es la dirección absoluta del archivo ".txt" que contiene las direcciones de las imágenes de entrenamiento.
- El parámetro "valid" es la dirección absoluta del archivo ".txt" que contiene las direcciones de las imágenes de prueba o validación.

- El parámetro “labels” es la dirección absoluta del archivo “.names”
- El parámetro “backup” es la dirección absoluta de la carpeta “backup” (Respaldos del entrenamiento)

```
semaforos_peatonales.data ×
C: > Users > fbrit > Desktop > DOCUMENTOS > modelo1YOLOv4-Tiny > darknet > build > darknet > x64 > data > semaforos_peatonales.data
1  classes= 2
2  train  = data/custom/train.txt
3  valid  = data/custom/test.txt
4  names = data/semaforos_peatonales.names
5  backup = backup/
```

Figure 41 Contenido del archivo “semáforos_peatonales.data” [Fuente: Los Autores]

2.5.10 Cálculo de los “Anchors” (Opcional)

Este parámetro se lo calcula antes de entrenar la Red e indica el tamaño de los cuadros delimitadores (bounding boxes) para la detección de los objetos del conjunto de datos a entrenar. El recálculo de los Anchors lo realizamos para tratar de obtener el mejor resultado en el entrenamiento de la red [25], si no se desea realizar, se puede omitir este paso y utilizar los anchors que vienen por defecto en “.cfg” de YOLO.

Para conseguir con efectividad la detección se necesita copiar las etiquetas totales (archivos .txt) que se obtienen en los anteriores apartados a la carpeta “images” que se encuentra en la dirección “"build/darknet/x64/data/custom/images”, esto es debido a que el programa espera encontrar las etiquetas en la misma carpeta de las imágenes.

Se ejecuta el siguiente comando en la dirección “"build/darknet/x64” para calcular los “anchors” de nuestro conjunto de imágenes [43]:

Darknet.exe detector calc_anchors data/semaforos_peatonales.data -num_of_clusters 6 -width 416 -height 416 -show

- El parámetro “data/semaforos_peatonales.data” es la dirección de nuestro archivo “.data”

- En el parámetro “-num_of_clusters” se especifica el número de “Anchors” que se desea obtener, en el caso de YOLOv4-tiny este número es 6. [25]
- En los parámetros “-width” y “-height”, se especifica el ancho y alto de la imagen de entrada a la red [25]. Las imágenes se redimensionarán al valor de estos dos parámetros cuando ejecutemos nuestra red, por supuesto, estos valores de redimensionamiento se los configura más adelante en el archivo “.cfg”.

```
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>.\darknet detector calc_anchors
data/semaforos_peatonales.data -num_of_clusters 6 -width 416 -height 416 -show
CUDA-version: 10010 (10010), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 4.5.3

 num_of_clusters = 6, width = 416, height = 416
 read labels from 4837 images
 loaded          image: 4837     box: 4863
 all loaded.

 calculating k-means++ ...

 iterations = 37


counters_per_class = 2601, 2262

 avg IoU = 80.12 %

Saving anchors to the file: anchors.txt
anchors =  20, 28,  28, 35,  36, 52,  53, 68,  74, 88, 119,130
```

Figure 42 Anchors generados para una resolución de 416x416 [Fuente: Los Autores]

Se obtendrá un archivo con estos anchors llamado “anchors.txt” en el directorio actual, más adelante en la configuración de la arquitectura de la red veremos donde colocar estos valores.

Anchors = 20, 28, 28, 35, 36, 52, 53, 68, 74, 88, 119,130

Si se cambian los valores de width y height a 320 y 320 se obtienen otros valores, es posible que la red se termine entrenando con esta resolución debido a que después puede llegar a ser un problema para la Raspberry tratar de realizar detecciones con la resolución que viene por defecto en YOLOv4-tiny que es de 416x416. Si se entrena la red con 416x416 para el redimensionamiento de las imágenes que entran a la red se puede obtener una buena precisión, pero al momento de ejecutar las detecciones en tiempo real puede que para reducir el tiempo de respuesta (aumentar FPS), se necesita alterar el valor de redimensionamiento de la red y esto perjudicara a la precisión con la que se entrenó, es decir, se perderá precisión al detectar objetos.

```
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>.\darknet detector calc_anchors
data/semaforos_peatonales.data -num_of_clusters 6 -width 320 -height 320 -show
 CUDA-version: 10010 (10010), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 4.5.3

 num_of_clusters = 6, width = 320, height = 320
 read labels from 4837 images
 loaded          image: 4837     box: 4863
 all loaded.

 calculating k-means++ ...

 iterations = 48


counters_per_class = 2601, 2262

 avg IoU = 80.40 %

Saving anchors to the file: anchors.txt
anchors =  15, 20,  22, 27,  17, 39,  30, 40,  44, 55,  66, 78
```

Figure 43 Anchors generados para una resolución de 320x320 [Fuente: Los Autores]

anchors = 15, 20, 22, 27, 17, 39, 30, 40, 44, 55, 66, 78

2.5.11 Creación y configuración del archivo ".cfg"

En la carpeta "cfg" en la dirección "darknet\build\darknet\x64\cfg" encontraremos el archivo "yolov4-tiny-custom.cfg", el cual comenzaremos a editar para conseguir el modelo que necesitamos para entrenar la red.

Dentro de este archivo se encuentran los hiper-parámetros de nuestra red, junto con su arquitectura, aquí se cambian algunos hiper-parámetros con el objetivo de adaptar la red a nuestra capacidad de GPU, número de clases, pasos de entrenamiento, etc. Podemos consultar [44] para información más detallada de los parámetros de la red de YOLO.

- Los parámetros "width" y "height" especifican el tamaño de re dimensionado de la imagen., como dijimos antes, para entrenar la red, se utiliza el tamaño 320x320, aunque se pueden colocar otros valores, pero deberán ser múltiplos de 32 [25].
- El parámetro "batch" indica el valor de imágenes que pasaran por la red en una iteración de entrenamiento, por defecto vienen en 64. Entonces un batch es igual a 64 imágenes.
- El valor N del parámetro "subdivisión" sirve para dividir el "batch" en N mini-lotes que pasaran por la red en una iteración de entrenamiento, por defecto viene en "1" así que lo cambiaremos a "4", entonces en cada subdivisión pasaran (64/4=16) imágenes por toda la red. La iteración de entrenamiento tendrá 4 grupos de 16 imágenes. [45]. Este parámetro se lo puede ir ajustando a más o menos, dependiendo de la capacidad de la GPU que se disponga.

- El parámetro “max_batches” es igual, según la documentación de darknet Yolo, el número de clases por 2000, pero no menor que el número de imágenes de entrenamiento y no menor que 6000. En nuestro caso disponemos de 2 clases y 4837 imágenes de entrenamiento. En el primer entrenamiento que se realizó de forma experimental se observó que 6000 iteraciones era un valor demasiado alto debido a que la red termino sobre-entrenándose, pero para evitar esto adelante se detallará la forma en la que se obtienen los mejores pesos para la detección. En conclusión, el valor que se colocará en este parámetro será de 4000. [25]
- El parámetro “steps” se debe cambiar a los valores correspondientes al 80% y 90% del valor colocado en “max_batches”, en este caso los valores que irán será 3200 y 3600.
- El parámetro “random=1” que incrementara la precisión al entrenar YOLO para diferentes resoluciones, este parámetro se lo encuentra dentro de las dos últimas capas de la red [yolo].
- El parámetro “classes” ira con el número de clases a identificar que en este caso es 2.
- El parámetro “anchors” que indica el tamaño de los cuadros delimitadores (bounding boxes), en este parámetro se duplica los anchors obtenidos para una resolución de 320x320 obtenidos en la sección 2.5.10.
- En la capa “[convolutional]” que esta antes de cada capa [yolo] se modifica el parámetro “filters”. La manera de calcular el número es la siguiente:

  Filters= (#número de clases + 5) x <num_of_mask>

  El parámetro <num_of_mask> es por defecto 3, pero esto puede cambiar y más si se recalcula los anchors para un conjunto de imágenes personalizado, en este caso se dejará en 3 debido a que capa [yolo] predice para 3 anchors diferentes, es decir la primera capa predice con los anchors (3, 4,5), que son los más grandes y la segunda capa predice con los anchors (0, 1,2) que serían los de menor tamaño, enfocados en objetos más pequeños. [25]

- Se agrega el parámetro “max_chart_loss=5”, no se encuentra declarado en la sección [net] porque por defecto viene en 20. Este parámetro es el valor máximo

en la escala de la perdida que se podrá visualizar en el "chart.png", generado durante el entrenamiento de la red. Le damos el valor de 5 porque en el primer entrenamiento se vio que la perdida terminaba por debajo de 1, así que 20 es un valor muy grande, se puede omitir este parámetro.

Al modelo le tomara aproximadamente 76 iteraciones completar una época, es decir, cuando el conjunto de datos pase de adelante hacia atrás por toda la red, debido a que se tiene:

$$\#\ iteraciones\ para\ completar\ una\ epoca = \frac{\#\ imagenes_entrenamiento}{batch_size}$$

$$\#\ iteraciones\ para\ completar\ una\ epoca = \frac{4837}{64}$$

$$\#\ iteraciones\ para\ completar\ una\ epoca \approx 76$$

Otro parámetro que se encuentra intrínseco es el número de épocas, valor que se obtiene por la ecuación [46]:

$$\text{epocas} = \frac{\max_batches\ *\ batch_size}{\#\ images_entrenamiento}$$

$$\text{epocas} = \frac{4000\ *\ 64}{4837}$$

$$epocas \approx 53$$

2.5.12 Configuración del guardado automático de pesos

Es importante ir guardando pesos de entrenamiento durante el entrenamiento para después escoger los que le den a la red una mejor precisión en los datos de prueba. Por defecto Darknet crea y actualiza un archivo de pesos cada 100 iteraciones y cada 1000 iteraciones crea un archivo único de pesos con el sub nombre de esa iteración. En el caso de requerir más de 6000 iteraciones esto puede ser suficiente, es decir, se genera 6 archivos de pesos diferentes y se puede escoger los mejores, pero si se quisiera obtener

más archivos de pesos todavía, como en este caso, se debe cambiar una línea de código en el archivo “detector.c” ubicado en “C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\src”.

En la línea 386 en donde se encuentra el siguiente fragmento de sentencia if:

```
(Iteration >= (iter_save + 1000) || iteration % 1000 == 0) && net.max_batches < 10000)
```

En realidad, la sentencia completa luce así:

```
if ((iteration >= (iter_save + 10000) || iteration % 10000 == 0) ||

(iteration >= (iter_save + 1000) || iteration % 1000 == 0) && net.max_batches < 10000)

{
…. resto del código
```

Una de las operaciones que se lleva a cabo dentro de esta sentencia, y es la que nos interesa, es la creación de un nuevo archivo de pesos único. Se cambia la sentencia y se comanda a que también genere este archivo ya no en cada 1000 iteraciones si no en cada 500, que es un valor que en total nos entregaría 8 archivos de pesos para analizar después del entrenamiento. La línea 386 del código quedaría:

```
(Iteration >= (iter_save + 1000) || Iteration % 500 == 0) && net.max_batches < 10000)
```

Con esto se logra que cada vez que llegue a iteraciones múltiplos de 500, genere un archivo de pesos, entre una de las varias condiciones que tiene esta sentencia.

```
//if (i % 1000 == 0 || (i < 1000 && i % 100 == 0)) {
//if (i % 100 == 0) {
if ((iteration >= (iter_save + 10000) || iteration % 10000 == 0) ||
    (iteration >= (iter_save + 1000) || iteration % 500 == 0) && net.max_batches < 10000)
{
    iter_save = iteration;
f GPU
    if (ngpus != 1) sync_nets(nets, ngpus, 0);
f
    char buff[256];
    sprintf(buff, "%s/%s_%d.weights", backup_directory, base, iteration);
    save_weights(net, buff);
}
```

Figure 44 Cambio en la línea 386 del código de "detector.c" [Fuente: Los Autores]

Opcionalmente se puede agregar la siguiente línea de código encima de la línea 301 para visualizar el número de imágenes de entrenamiento que están presentes, esto solamente se lo realizo para verificar que sean 4837, y así fue.

Printf ("\n (Imágenes de entrenamiento = %d) ", train_images_num);

```
const int iteration = get_current_iteration(net);
//i = get_current_batch(net);
printf("\n (Imagenes de entrenamiento = %d ) ", train_images_num);
int calc_map_for_each = 4 * train_images_num / (net.batch * net.subdivisions);  // calculate mAP for each 4 Epochs
//printf("\n (Calculo de map para cada = %d ) ", calc_map_for_each);
calc_map_for_each = fmax(calc_map_for_each, 100);
int next_map_calc = iter_map + calc_map_for_each;
next_map_calc = fmax(next_map_calc, net.burn_in);
//next_map_calc = fmax(next_map_calc, 400);
```

Figure 45 Cambio encima de la línea 301 en el código de "detector.c" [Fuente: Los Autores]

Realizados estos cambios, se debe compilar darknet nuevamente siguiendo los mismos pasos vistos en la sección 2.5.6 y de igual forma pasar los archivos de la carpeta "Release" a la carpeta "x64". Es importante realizar la compilación, caso contrario, los cambios no surgirán efecto, aunque esta vez se demorara menos debido a que solo necesita volver a compilar "detector.c" de la carpeta "src".

2.5.13 Entrenamiento de la Red

Principalmente, se descargan las primeras 29 capas convolucionales para yolov4-tiny disponibles en [47]. Este archivo se lo colocará en la dirección "darknet\build\darknet\x64" antes de comenzar el entrenamiento. Este archivo es importante debido a que de esta forma no empieza un entrenamiento desde cero, si no con valores de pesos que servirán para entrenar la red más rápido.

Una vez que todos los pasos se ejecutan correctamente solo queda entrenar la red con el siguiente comando:

darknet.exe detector train data/semaforos_peatonales.data cfg/yolov4-tiny-custom.cfg yolov4-tiny.conv.29

Después de que el entrenamiento este completo debemos revisar los pesos generados en la carpeta “backup”. Cada 100 iteraciones podemos parar y volver a empezar el entrenamiento desde ese punto, solo que en vez de cargar el comando con el archivo “yolov4-tiny.conv.29”, lo hacemos con el archivo de pesos (*.weights) que termine en “_last.weights”. [25]

Nombre	Fecha de modificación	Tipo	Tamaño
tmp	29/7/2021 20:05	Documento de te...	0 KB
yolov4-tiny-custom_500.weights	31/7/2021 17:18	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_1000.weights	31/7/2021 18:38	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_1500.weights	31/7/2021 20:02	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_2000.weights	31/7/2021 22:34	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_2500.weights	31/7/2021 23:51	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_3000.weights	1/8/2021 1:08	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_3500.weights	1/8/2021 2:29	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_4000.weights	1/8/2021 3:58	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_final.weights	1/8/2021 3:58	Archivo WEIGHTS	22.980 KB
yolov4-tiny-custom_last.weights	1/8/2021 3:58	Archivo WEIGHTS	22.980 KB

Figure 46 8 archivos de pesos generados durante el entrenamiento y por cada 500 iteraciones [Fuente: Los Autores

El entrenamiento se completará cuando llegue a las 4000 iteraciones especificadas con la siguiente ventana final:

```
 (Imagenes de entrenamiento = 4837 )
 4000: 0.165071, 0.139147 avg loss, 0.000026 rate, 13.672000 seconds, 256000 images, 0.280508 hours left
Saving weights to backup//yolov4-tiny-custom_4000.weights
Saving weights to backup//yolov4-tiny-custom_last.weights
Saving weights to backup//yolov4-tiny-custom_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

Figure 47 Finalización del entrenamiento y generación de los pesos finales [Fuente: Los Autores]

El entrenamiento finalizo con una pérdida promedio de 0.139147 y tiempo de muestreo de alrededor de 11 horas, aproximadamente.

Durante la ejecución del entrenamiento podemos ir revisando el valor de la perdida vs las iteraciones, tanto en la consola como gráficamente en el “chart.png” que se genera automáticamente.

```
 (Imagenes de entrenamiento = 4837 )
 596: 0.538197, 0.527717 avg loss, 0.000329 rate, 13.072000 seconds, 38144 images, 11.491012 hours left
Loaded: 0.000000 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.416830), count: 11, class_loss =
 0.657245, iou_loss = 0.517167, total_loss = 1.174412
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.495351), count: 4, class_loss =
0.327093, iou_loss = 1.496711, total_loss = 1.823805
 total_bbox = 33266, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.439484), count: 9, class_loss =
0.631231, iou_loss = 0.475696, total_loss = 1.106928
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.506177), count: 6, class_loss =
0.455350, iou_loss = 1.624887, total_loss = 2.080236
 total_bbox = 33281, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.425330), count: 7, class_loss =
0.601940, iou_loss = 0.391115, total_loss = 0.993055
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.440154), count: 5, class_loss =
0.423704, iou_loss = 1.430554, total_loss = 1.854258
 total_bbox = 33293, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.344139), count: 9, class_loss =
0.546793, iou_loss = 0.371936, total_loss = 0.918729
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.377195), count: 5, class_loss =
0.368441, iou_loss = 0.765700, total_loss = 1.134141
 total_bbox = 33307, rewritten_bbox = 0.000000 %
```

Figure 48 Métricas calculadas durante una iteración de entrenamiento [Fuente: Los Autores]

Si se analizan estas iteraciones se pueden identificar algunas métricas y valores importantes:

- 596: Numero de iteraciones.
- 0.538197: Valor de la pérdida total.
- 0.527717 avg los: Valor de la pérdida promedio.
- 0.000329 rate: Valor del actual “learning rate”
- 13.072000 seconds: Tiempo de procesamiento del batch/iteración.
- 38144 images: El número de imágenes procesadas hasta ese momento, este valor es simplemente $(Numero\ de\ iteraciones\ x\ batch_size)$ que en este caso seria $596 * 64 = 38144$. Lo cual tiene sentido porque como se observa cada batch/iteración que pasa por la red lleva 64 imágenes en 4 grupos de 16 imágenes.

En cada uno de estos 4 grupos se puede ver 2 valores de la métrica conocida como Intersección sobre Unión (IoU) para dos diferentes capas de la red (Región 30 y Región 37). Esta métrica sirve para determinar qué tan preciso nuestro modelo detecto nuestro objeto, siendo 1 el 100%, es decir, detección perfecta. Para obtener este valor se divide el área formada por la intersección del objeto y el cuadro delimitador para el área formada por la unión del objeto y el cuadro delimitador, entonces aquí el valor más alto, ósea 1, será como tal cuando las dos áreas sean iguales. [28]

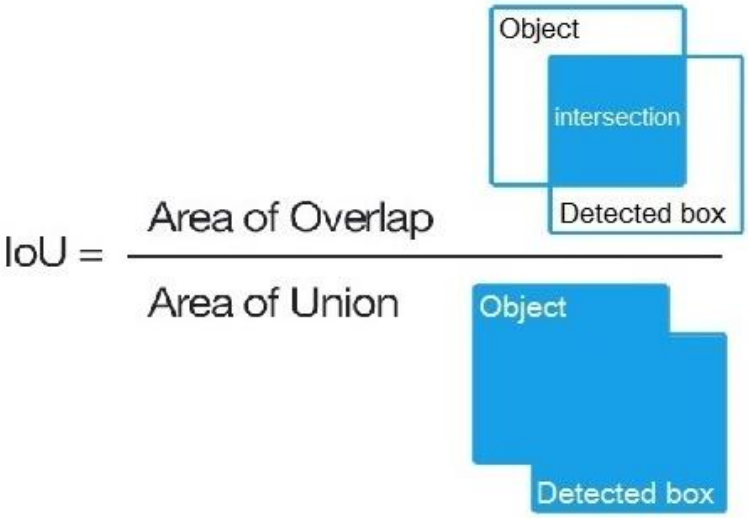


Figure 49 Representación gráfica del cálculo de la Intersección sobre Unión (IoU) [25]

En este caso en cada grupo/subdivisión devuelve el Avg (IoU), es decir el promedio IoU de cada imagen en la subdivisión actual, que recordemos eran 4 en total.

2.5.14 Selección de los mejores pesos

Para escoger los mejores pesos se utiliza como principal métrica la mAP (Precisión promedio media) y se calcula en las imágenes de prueba, ya que esta precisión promedio media es la de importancia para saber qué tan robusto es el modelo. Puede darse el caso en el que los mejores pesos obtenidos se hayan generado antes de completar el total de iteraciones, esto es debido a que el modelo después de obtener los mejores pesos para la red haya empezado a sobre entrenarse y llegara a terminar el entrenamiento en un estado de "Over-fitting", es por eso que se debe buscar siempre el punto de equilibrio o como se encuentra en la siguiente imagen, el "Early stopping point".

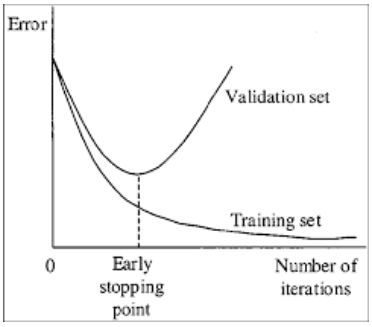


Figure 50 Curvas de entrenamiento y prueba a lo largo de las iteraciones de entrenamiento [25]

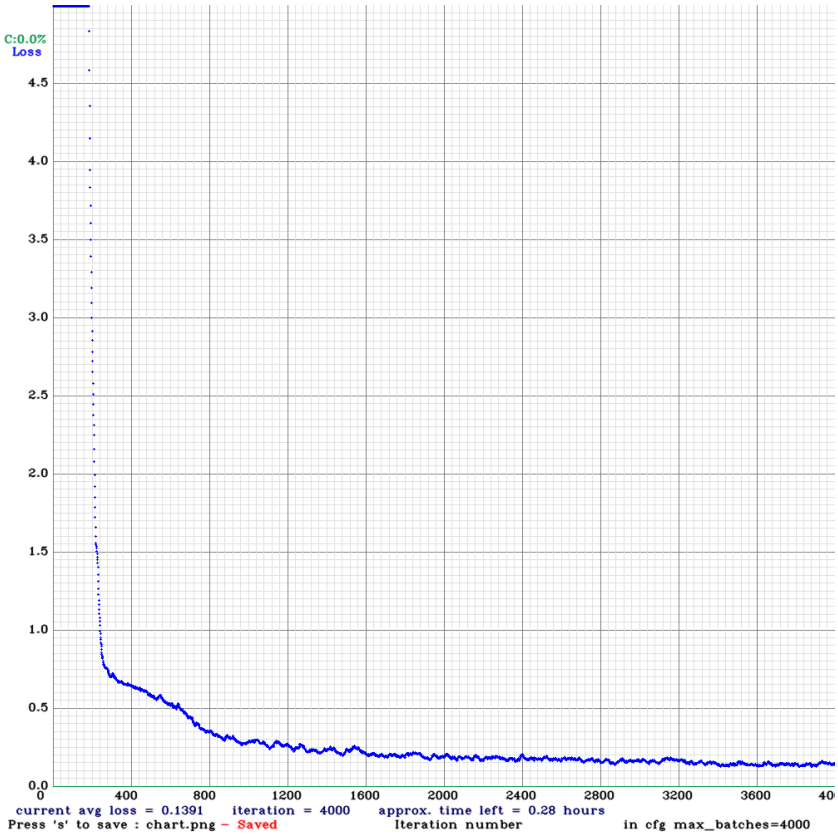


Figure 51 Grafica obtenida que representa la variación de la pérdida por cada iteración de entrenamiento [Fuente: Los Autores] [25],

2.5.15 Detección en imágenes de prueba usando los mejores pesos

Darknet da la facilidad de realizar detecciones en imágenes usando el archivo de pesos que se deseen, en este caso solo para un modo de prueba, se escogen los mejores pesos obtenidos en el apartado anterior, que son los pesos que se crearon después de las 4000 iteraciones, y se va a detectar 5 imágenes del set de prueba.

Se usa el comando:

darknet.exe detector test data\semaforos_peatonales.data cfg\yolov4-tiny-custom.cfg backup\yolov4-tiny-custom_4000.weights

La consola pedirá ingresar la dirección de la imagen que se desea detectar; ahora, todas las imágenes están en una misma carpeta llamada “images” ya que no se separó las imágenes, solo se crea dos archivos “.txt” que contenían las imágenes de prueba y entrenamiento. Entonces se obtiene dos soluciones, abrir el archivo “test.txt” para saber que imágenes buscar en la carpeta “images” o separar en dos carpetas las imágenes de entrenamiento y de prueba. En este caso se desarrolla un algoritmo en Python para

ejecutar esta tarea, entonces lo primero que se hace es crear dos carpetas dentro de la carpeta “custom”, llamadas “train” y “test” y se ejecuta el siguiente algoritmo dentro de la dirección en la que se encuentran estas carpetas.

```
Import os

From glob import glob

Import shutil

#DIRECCIONES PARCIALES, USAR ESTAS SI SE ESTA EJECUTANDO EL
#SCRIPT DESDE LA CARPETA "custom"

#train_txt = "train.txt"

#test_txt = "test.txt"

#direccion_images = "images"

#train_folder = "train"

#test_folder = "test"

#DIRECCIONES ABSOLUTAS

train_txt = "C:/Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/darknet/build/darknet/x64/data/custom/train.txt"

test_txt = "C:/Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/darknet/build/darknet/x64/data/custom/test.txt"

direccion_images = "C: /Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/darknet/build/darknet/x64/data/custom/images"

train_folder = "C: /Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/darknet/build/darknet/x64/data/custom/train"

test_folder = "C: /Users/fbrit/Desktop/DOCUMENTOS/modelo1YOLOv4-Tiny/darknet/build/darknet/x64/data/custom/test"

#LISTA DE IMAGENES DE ENTRENAMIENTO

lista_imagenes_train = []

With open (train_txt,'r') as t:
```

```
    For linea in t:
        Imagen = linea.split ("/") [3].split (".")[0]
        #print (imagen)
        lista_imagenes_train.append (imagen)
    t.close ()
#LISTA DE IMAGENES DE PRUEBA
lista_imagenes_test = []
With open (test_txt,'r') as t:
    For linea in t:
        Imagen = linea.split ("/") [3].split (".")[0]
        #print (imagen)
        lista_imagenes_test.append (imagen)
    t.close ()
#SELECCION DE IMAGENES Y COPIA A OTRA DIRECCION
For imagen in glob (os.path.join (direccion_images,'*')):
    nombre_archivo = imagen.split ("/") [12].split ("\\") [1].split (".")[0]
    If nombre_archivo in lista_imagenes_train:
        shutil.copy2 (imagen, train_folder)
    Elif nombre_archivo in lista_imagenes_test:
        shutil.copy2 (imagen, test_folder)
    Else:
        Print ("Archivo omitido (¿Error?): {}".format (imagen))
    #print (nombre_archivo)
```

2.5.16 Detección en tiempo real usando los mejores pesos

Esta deteccion se la realiza con la webcam de la computadora en tiempo real con el siguiente comando:

darknet.exe detector demo data\semaforos_peatonales.data cfg\yolov4-tiny-custom.cfg backup/yolov4-tiny-custom_4000.weights

Como se ve en las dos siguientes figuras, la deteccion parece no tener sentido debido a que detecta como semaforo peatonal a un mouse que emite un brillo de un color similar a los que la red aprendio a detectar, esto es porque por defecto cuando se realiza las detecciones en tiempo real, el parametro “thresh” o umbral de deteccion viene en 0, es decir, detecta todos los objetos que puedan llegar a parecerse al objeto con el que entrenamos la red.

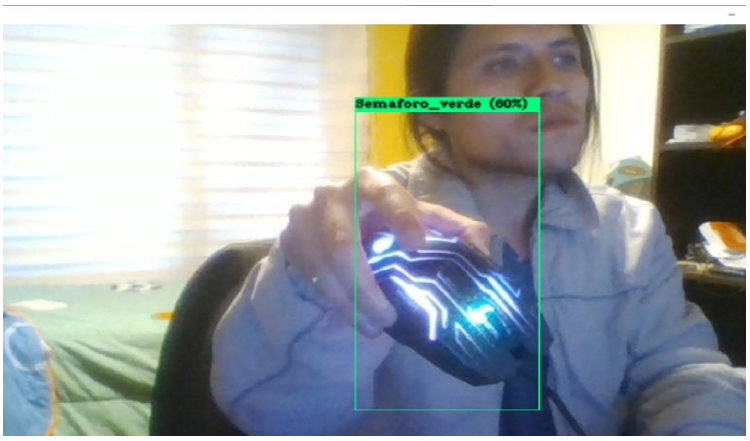


Figure 52 Detección en tiempo real de un mouse que emite luces verdes que es clasificado como semáforo verde [Fuente: Los Autores]

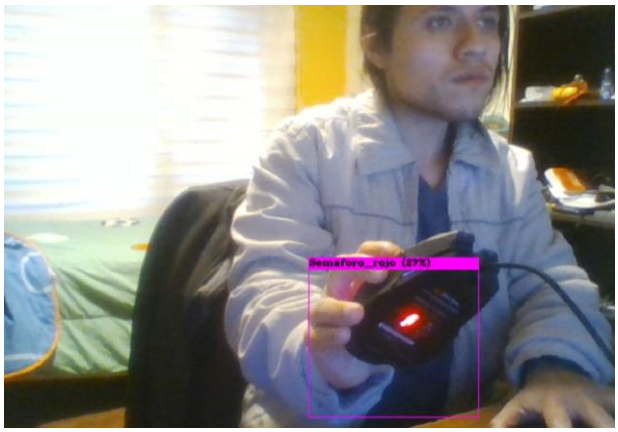


Figure 53 Detección en tiempo real de un mouse que emite luces rojas que es clasificado como semáforo rojo [Fuente: Los Autores]

Debido a que la precisión promedio para ambas clases está por encima del 98%, asignaremos un valor de 0.8 al umbral de detección, de esta forma el modelo detectará solo objetos que tengan un valor de confianza mayor o igual a 0.8, aunque también existe el riesgo de que algunos de los objetos que sean realmente la clase que se quiera detectar queden excluidos de la detección por el elevado umbral.

darknet.exe detector demo data\semaforos_peatonales.data cfg\yolov4-tiny-custom.cfg backup/yolov4-tiny-custom_4000.weights -thresh 0.8

Figure 54 Detección en tiempo real con un umbral de detección de 0.8 [Fuente: Los Autores]

2.5.17 Aumento del rendimiento con "darknet_video.py"

Hasta el momento se realiza detecciones en tiempo real a un alrededor de 14.2 FPS (Fotogramas por segundo), pero al pasar a detectar objetos en un procesador de recursos más escasos esto puede llegar a ser un problema. Darknet provee de un script desarrollado en Python para realizar las tareas de detección con un menor consumo de recursos y que por ende acelerara las detecciones por segundo en nuestra computadora, este script se llama "darknet_video.py".

Este script importa "darknet.py" que a su vez necesita de un DLL específico y que nos pide compilar con Visual Studio 2015 V140, aunque al final se opta por la versión 2019 V142 sin problema [25]. El proyecto que se debe compilar se llama "yolo_cpp_dll.sln", que es la versión del DLL que trabaja con la GPU, dentro de la dirección "/build/darknet" y para hacerlo se deben cambiar dos líneas de código. Se procede a abrir el "yolo_cpp_dll.vcxproj", en este caso con el bloc de notas y corrigiendo las dos líneas que se mencionaron anteriormente en la versión de CUDA, debido a que por defecto esta viene declarada con la versión 11.1 y se procederá a trabajar con la versión 10.1.

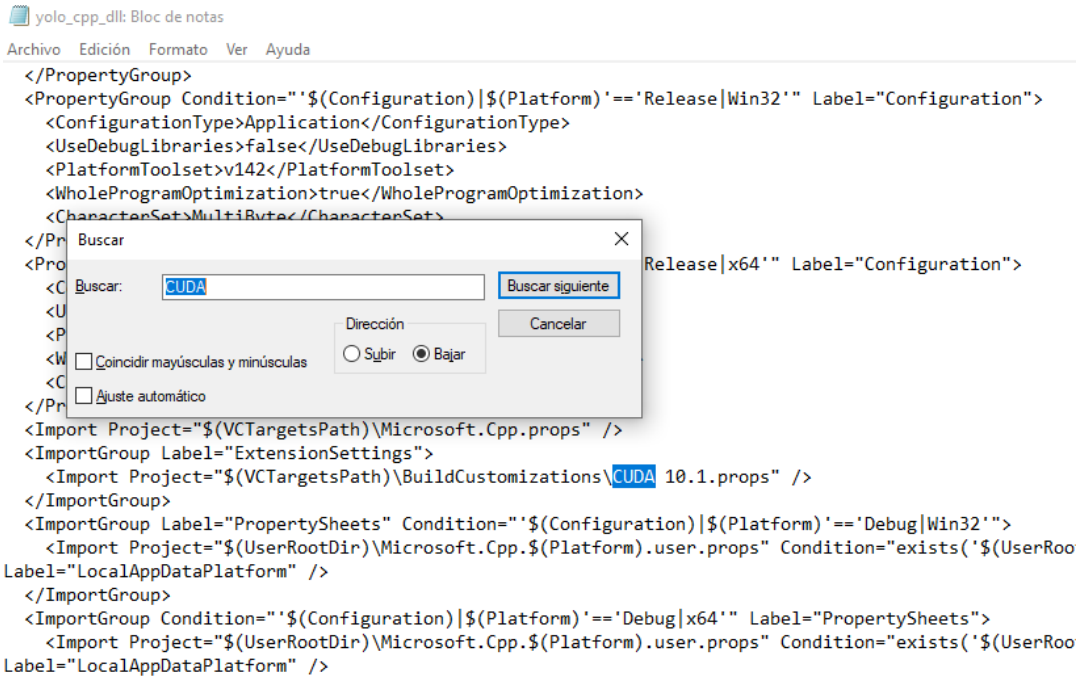


Figure 55 Cambio de la versión de CUDA que se usara en la compilación del proyecto [Fuente: Los Autores]

Hecho el cambio, se abre la solucion en MSVS 2019 V140 y se compila el proyecto, en “Release x64” y se obtendrá el DLL dentro de la carpeta “x64”.

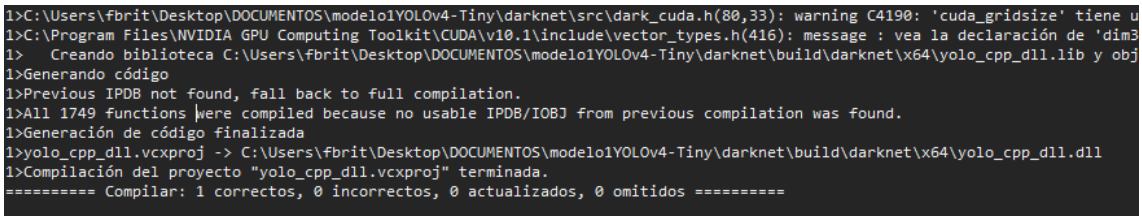


Figure 56 Resultado devuelto después de la compilación exitosa [Fuente: Los Autores]

Se ejecutan unas detecciones de prueba para observar el aumento de los fotogramas por segundo (FPS) y en caso de obtener algún error sobre la carga del DLL creado con anterioridad en la línea “lib = CDLL (winGPUdll, RTLD_GLOBAL)”, agregar estas dos líneas de código (marcadas en plomo) que le permitirán a “darknet.py” encontrar los DLL que está buscando:

```python
#!python3
"""
Python 3 wrapper for identifying objects in images

Requires DLL compilation

Both the GPU and no-GPU version should be compiled; the no-GPU version should be renamed "yolo_cpp_dll_nogpu.dll".

On a GPU system, you can force CPU evaluation by any of:

- Set global variable DARKNET_FORCE_CPU to True
- Set environment variable CUDA_VISIBLE_DEVICES to -1
- Set environment variable "FORCE_CPU" to "true"

Directly viewing or returning bounding-boxed images requires scikit-image to be installed (`pip install scikit-image`)

Original *nix 2.7: https://github.com/pjreddie/darknet/blob/0f110834f4e18b30d5f101bf8f1724c34b7b83db/python/darknet.py
Windows Python 2.7 version: https://github.com/AlexeyAB/darknet/blob/fc496d52bf22a0bb257300d3c79be9cd80e722cb/build/darknet/x64/darknet.py

@author: Philip Kahn
@date: 20180503
"""
from ctypes import *
import math
import random
import os

os.add_dll_directory('C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v10.1/bin')
os.add_dll_directory(os.path.dirname(__file__))

print("Run: darknet_images.py or:\n")
print("python.exe darknet_video.py --data_file cfg/coco.data --config_file cfg/yolov4.cfg --weights yolov4.weights --input test.mp4 \n")
```

Figure 57 Adición de dos líneas de código en caso de error "Not found DLL" [Fuente: Los Autores]

Se ejecuta el siguiente comando y automáticamente se abrirá una ventana con la imagen de la webcam y con las detecciones.

python.exe darknet_video.py --data_file data\semaforos_peatonales.data --config_file cfg\yolov4-tiny-custom.cfg --weights backup\yolov4-tiny-custom_4000.weights --input 0

Figure 58 Detecciones en tiempo real con "darknet_video.py" con un "thresh=0" [Fuente: Los Autores]

Como se puede observar, existe un aumento de los fotogramas por segundo al ejecutar las detecciones en nuestra computadora, se mantiene en una media por encima de 20 más o menos, en este caso detecta la parte inferior del mouse como un semáforo rojo, pero con un valor de confianza bajo, como pasaba al ejecutar "darknet.exe", estos

errores de detección se solucionan una vez más aumentando el “thresh”, haciendo que las detecciones sean más estrictas.

2.5.18 Notificación auditiva del estado de señalización del semáforo peatonal

Para dar a conocer el estado de señalización del semáforo peatonal, se utiliza la librería “pyttsx3” [48], que básicamente es una librería “Text To Speech” que ayudará a convertir las clases detectadas a un formato mp3 que la computadora podrá reproducir con o sin la necesidad de crearlo, básicamente.

```
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>pip install pyttsx3
Collecting pyttsx3
  Downloading pyttsx3-2.90-py3-none-any.whl (39 kB)
Collecting pypiwin32
  Downloading pypiwin32-223-py3-none-any.whl (1.7 kB)
Collecting comtypes
  Downloading comtypes-1.1.10.tar.gz (145 kB)
     |████████████████████████████████| 145 kB 726 kB/s
Collecting pywin32
  Using cached pywin32-301-cp39-cp39-win_amd64.whl (9.3 MB)
Using legacy 'setup.py install' for comtypes, since package 'wheel' is not installed.
Installing collected packages: pywin32, pypiwin32, comtypes, pyttsx3
    Running setup.py install for comtypes ... done
Successfully installed comtypes-1.1.10 pypiwin32-223 pyttsx3-2.90 pywin32-301
WARNING: You are using pip version 21.1.3; however, version 21.2.1 is available.
You should consider upgrading via the 'c:\users\fbrit\appdata\local\programs\python\python39\python.exe -m pip install
-upgrade pip' command.

C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>
```

Figure 59 Instalación de “pyttsx3” con pip [Fuente: Los Autores]

Para el control de la salida de audio se utiliza la librería “pynput” [49], aunque esta librería se usará solo dentro de Windows.

```
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>pip install pynput
Collecting pynput
  Downloading pynput-1.7.3-py2.py3-none-any.whl (99 kB)
     |████████████████████████████████| 99 kB 526 kB/s
Collecting six
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, pynput
Successfully installed pynput-1.7.3 six-1.16.0
WARNING: You are using pip version 21.1.3; however, version 21.2.1 is available.
You should consider upgrading via the 'c:\users\fbrit\appdata\local\programs\python\python39\python.exe -m pip install -
-upgrade pip' command.

C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64>
```

Figure 60 Instalación de “pynput” con pip [Fuente: Los Autores]

Dentro del script “darknet_video.py” se encuentran varias funciones que permiten trabajar al algoritmo, para hacer detecciones, cargar la arquitectura, llenar los pesos, etc. Al final del código se puede encontrar la parte principal, el “main”, y se encuentra que el programa ejecuta 3 funciones con procesos de multi-hilo con la función “Thread” de la librera “threading”. Estas funciones son: video_capture, inference y drawing. La función “drawing” será la única que se edita para crear un archivo “.txt” que ira escribiendo en la primera línea, la clase que se está detectando en ese momento.

```
def drawing(frame_queue, detections_queue, fps_queue):
    random.seed(3)  # deterministic bbox colors
    video = set_saved_video(cap, args.out_filename, (video_width, video_height))
    while cap.isOpened():
        label=str("")
        frame = frame_queue.get()
        detections = detections_queue.get()
        fps = fps_queue.get()
        detections_adjusted = []
        if frame is not None:
            for label, confidence, bbox in detections:
                bbox_adjusted = convert2original(frame, bbox)
                detections_adjusted.append((str(label), confidence, bbox_adjusted))
            image = darknet.draw_boxes(detections_adjusted, frame, class_colors)
            with open("detecciones.txt", "w") as f:
                f.write(label)
                f.close()
            if not args.dont_show:
                cv2.imshow('Inference', image)
            if args.out_filename is not None:
                video.write(image)
            if cv2.waitKey(fps) == 27:
                break

    cap.release()
    video.release()
    cv2.destroyAllWindows()
```

Figure 61 Adición de tres líneas de código en la función "drawing" para obtener las etiquetas [Fuente: Los Autores]

Ahora esto se hace con un simple propósito, evitar tener que ejecutar la reproducción del audio dentro del script "darknet_video.py" debido a que la ejecución de este proceso bloquearía el programa hasta que termine y esto puede llegar a generar problemas en las detecciones, como atrasos o paradas de programa inesperadas.

Para convertir las etiquetas de las detecciones en audio, usamos el siguiente script:

dictador_clases.py - C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\b...

File Edit Format Run Options Window Help

```python
from pynput import keyboard as kb
import pyttsx3

def operacion():
    engine = pyttsx3.init()
    with open("detecciones.txt","r") as f:
        label = f.readline()
        if label == "Semaforo_rojo":
            engine.say("Semaforo rojo")
            if engine.isBusy():
                print("Reproduciendo...")
                engine.runAndWait()

        if label == "Semaforo_verde":
            engine.say("Semaforo verde")
            if engine.isBusy():
                print("Reproduciendo...")
                engine.runAndWait()


def pulsa(tecla):
    operacion()

def suelta(tecla):
    operacion()


with kb.Listener(pulsa,suelta) as escuchador:
    escuchador.join()
```

Figure 62 Script para leer la etiqueta en "detecciones.txt" y pasarla a audio [Fuente: Los Autores]

El funcionamiento del llamado "dictador_clases.py" es muy sencillo, lee la primera y única línea que tendrá el archivo "detecciones.txt" y dependiendo de qué etiqueta sea, reproducirá el audio "Semáforo verde" o el audio "Semáforo rojo". Ahora el control de la salida del audio se lo realiza con "pynput", de forma que solo reproducirá audio bajo la primera condición de que se presione o se suelte cualquier tecla del teclado y después considerará las demás condicionales que mencionamos antes.

Con la creación de un archivo ".txt" por parte de "darknet_video.py" y a parte la lectura del mismo archivo para posteriormente pasarlo a formato mp3, no se interrumpen los procesos de darknet y el programa puede seguir funcionando con normalidad.

Para ejecutar estos dos scripts al mismo tiempo, se crea un archivo.bat en el cual se colocan las siguientes líneas de código:

```
@echo off

Start python dictador_clases.py

Start python darknet_video.py --data_file data\semaforos_peatonales.data --config_file cfg\yolov4-tiny-custom.cfg --weights backup\yolov4-tiny-custom_4000.weights --input 0
```

Con la finalidad de arrancar dos consolas para los dos scripts, el archivo .bat se llama “Detector_Dictador_python.bat”, así que ahora simplemente se deberá ejecutar este .bat para arrancar los dos scripts simultáneamente.

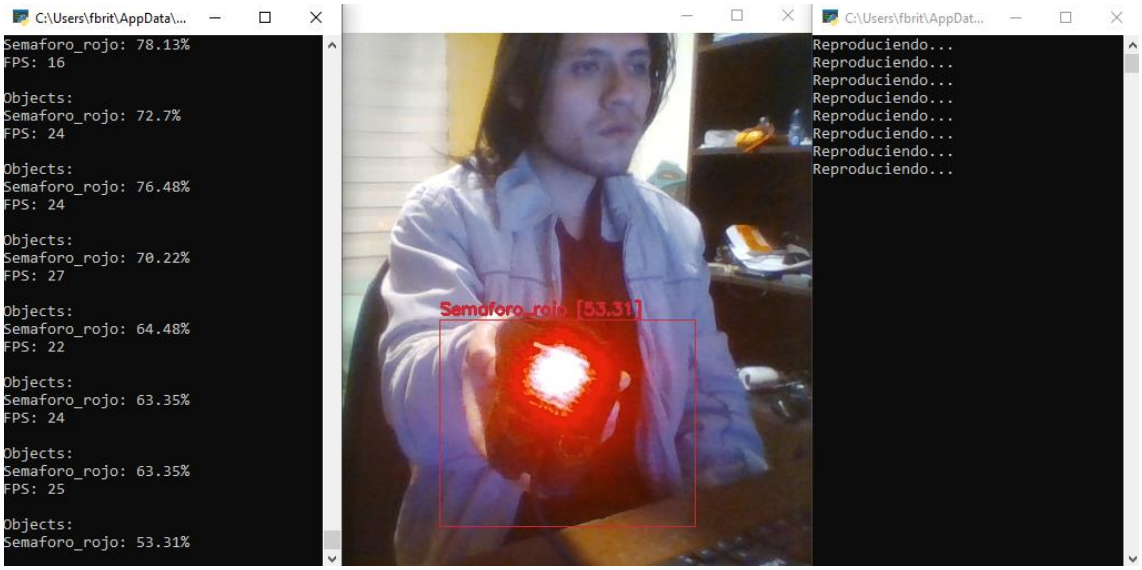


Figure 63 Ejecución del archivo.bat “Detector_Dictador_python.bat [Fuente: Los Autores]”

2.6 Adaptación del modelo en la Raspberry Pi 4

La Raspberry Pi 4 tiene muchos menos recursos de los que posee la computadora en donde se entrenó el modelo, así que los resultados en cuanto a la velocidad de inferencia serán menores, es decir, la velocidad de las detecciones será más lenta.

Lo primero es descargar, compilar e instalar OpenCV, en este caso es preferible la versión 4.1.1 [50], de igual forma que en Windows se utiliza un entorno virtual llamado “vemodeloYOLOtinyv4”.

Se debe descargar Darknet del repositorio de Github [25] y editar el archivo “Makefile” a la siguiente configuración, que, a diferencia de la usada en Windows, no posee GPU ni soporte CUDA, solo se habilita OpenCV.

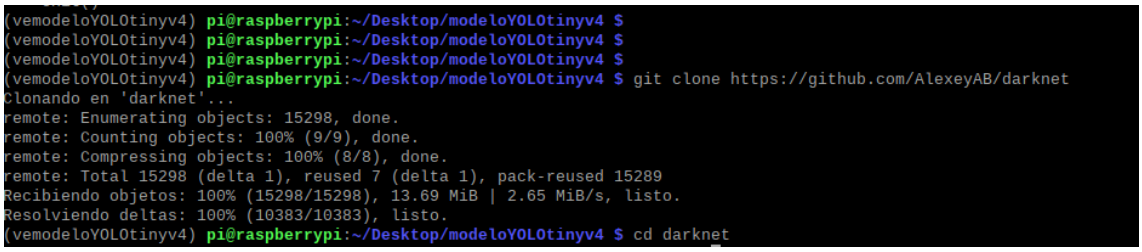


Figure 64 Descarga de darknet [Fuente: Los Autores]

```
GPU=0
CUDNN=0
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=0
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
```

Figure 65 Configuración del archivo "Makefile" [Fuente: Los Autores]

Para compilar Darknet en Linux, y en este caso estamos se utiliza Raspbian, además, también se utiliza simplemente "make", los pasos y otras formas de compilación con toda la información se las puede encontrar en el repositorio de Github de AlexeyAB [25].

Hay que franquear los pesos obtenidos con el entrenamiento (yolov4-tiny-custom_4000.weights) a la dirección "/darknet", la arquitectura de red "yolov4-tiny-custom.cfg" la pasaremos a la dirección "/darknet/cfg" y finalmente los archivos "semaforos_peatonales.names" y "semaforos_peatonales.data" se pasan a la dirección "/darknet/data". Se edita el contenido del ".data" debido a que en la Raspberry no se necesita validar el modelo o entrenarlo, ese proceso ya lo realizamos en Windows con la computadora.

```
classes= 2
names = data/semaforos_peatonales.names
```

Figure 66 Nuevo contenido de "semaforos_peatonales.data" [Fuente: Los Autores]

2.6.1 Conversión al modelo de Tensorflow

Las detenciones realizadas en el framework "Darknet" no parecen ser del todo satisfactorias en la Raspberry Pi 4, así que se utilizara otro framework, que además de ser recomendado por "Darknet", ayudará a incrementar la velocidad de los procesos de inferencia en la Raspberry.

Instalar Tensorflow en su versión 2.3.0 [51] [52] y descargaremos el repositorio de Github [53] con los algoritmos para realizar la conversión del modelo de pesos que se obtuvo en darknet al modelo de TensorFlow.

Se pasa el archivo de pesos a la carpeta “data” que se obtiene del repositorio anteriormente descargado y en la carpeta “core” editaremos el “config.py” con la dirección de nuestro archivo “. names”, que se pasa a la dirección “data/classes” y el arreglo de anchors obtenidos cuando desarrollábamos el modelo en Windows, estos valores se los asigna a la opción “__C.YOLO.ANCHORS_TINY”, debido a que solo se utiliza esta versión d YOLO en específico.

```
#! /usr/bin/env python
# coding=utf-8
from easydict import EasyDict as edict


__C                             = edict()
# Consumers can get config by: from config import cfg

cfg                             = __C

# YOLO options
__C.YOLO                        = edict()

__C.YOLO.CLASSES                = "./data/classes/semaforos_peatonales.names" #coco.names
__C.YOLO.ANCHORS                = [12,16, 19,36, 40,28, 36,75, 76,55, 72,146, 142,110, 192,243, 459,401]
__C.YOLO.ANCHORS_V3             = [10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326]
#__C.YOLO.ANCHORS_TINY           = [23,27, 37,58, 81,82, 81,82, 135,169, 344,319]
__C.YOLO.ANCHORS_TINY           = [15,20, 22,27, 17,39, 30,40, 44,55, 66,78]
__C.YOLO.STRIDES                = [8, 16, 32]
__C.YOLO.STRIDES_TINY           = [16, 32]
__C.YOLO.XYSCALE                = [1.2, 1.1, 1.05]
__C.YOLO.XYSCALE_TINY           = [1.05, 1.05]
__C.YOLO.ANCHOR_PER_SCALE       = 3
__C.YOLO.IOU_LOSS_THRESH        = 0.5
```

Figure 67 Configuración de parámetros en “config.py” [Fuente: Los Autores]

En el repositorio de estos códigos se puede encontrar la secuencia de comandos a ejecutar dependiendo de lo que se busca, en este caso se realiza la conversión al modelo Tensorflow con el siguiente comando:

Python save_model.py --weights ./data/yolov4-tiny-custom_4000.weights --output ./checkpoints/yolov4-tiny --input_size 320 --model yolov4 --tiny

Esto dejará una carpeta con las conversiones realizadas por el script “save_model.py”, que ahora usaremos para realizar detecciones con la webcam.

Es importante mencionar que se edita una línea de código en el script “detectvideo.py” debido a que pasa un parámetro de tipo “string” a la función “VideoCapture” en vez de un tipo “int”, esto con la intención de que a “VideoCapture” se le pase el valor de 0 para que pueda abrir la webcam.

```
py ✖ detectvideo.py ✖
flags.DEFINE_boolean('dis_cv2_window', False, 'disable cv2 window during the process') # this is good

def main(_argv):
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)
    STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)
    input_size = FLAGS.size
    video_path = FLAGS.video

    print("Video from: ", video_path )
    vid = cv2.VideoCapture(int(video_path)) #Linea editada, convirtiendo a int porque viene en str
```

Figure 68 Conversión de tipo "string" a "int" en la función "VideoCapture" [Fuente: Los Autores]

2.6.2 Conversión al modelo de Tensorflow Lite

En busca de tratar de mejorar en lo máximo que se pueda la velocidad de inferencia del modelo al igual que el arranque del programa, se convierten los pesos obtenidos por darknet al modelo Tensorflow Lite, que como dice en su propia documentación [54], es la mejor versión de Tensorflow para ejecutar modelos de TensorFlow en dispositivos incorporados, móviles o de IoT.

Antes de empezar la conversión, es importante tener en cuenta dos componentes:

- El intérprete de Tensorflow Lite, que ejecuta modelos optimizados en varios tipos de hardware como celulares o dispositivos Linux.
- El conversor de Tensorflow Lite, que convierte los modelos de Tensorflow en un formato eficiente para que los use el intérprete.

Para realizar la conversión, en el repositorio que descargamos anteriormente en la sección 2.6.1 [53], primero debemos ejecutar el siguiente comando:

Python save_model.py --weights ./data/yolov4-tiny-custom_4000.weights --output ./checkpoints/yolov4-tiny-tflite --input_size 320 --model yolov4 --framework tflite

Una vez terminado, se obtiene de igual forma una carpeta con los archivos resultantes de la conversión. Ahora para obtener el archivo ".tflite" se ejecuta el siguiente comando:

Python convert_tflite.py --weights ./checkpoints/yolov4-tiny-tflite --output ./checkpoints/yolov4-tiny.tflite

El script "convert_tflite.py" recibe un parámetro llamado "quantize_mode" que por defecto viene en "float32", es por esa razón que no se lo coloca en el comando

anterior, aunque en la siguiente sección se verá como optimizar más el modelo cambiando este parámetro.

Si se llega a recibir algún error como el que se muestra en la Fig. , se debe añadir una línea de código para evitar la “violación se segmento”, se debe agregar la línea “converter.experimental_new_converter=False”

```
2021-08-05 12:23:44.536903: I tensorflow/core/grappler/devices.cc:78] Number of eligible GPUs (core count >= 8, compute capability >= 0.0): 0 (Note: TensorFlow was not compiled with CUDA
port)
2021-08-05 12:23:44.537193: I tensorflow/core/grappler/clusters/single_machine.cc:356] Starting new session
2021-08-05 12:23:45.000097: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816] Optimization results for grappler item: graph_to_optimize
2021-08-05 12:23:45.000202: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:818]   function_optimizer: Graph size after: 647 nodes (544), 657 edges (553), time = 83.7640076ms.
2021-08-05 12:23:45.000237: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:818]   function_optimizer: function_optimizer did nothing. time = 1.21ms.
I0805 12:23:52.092086 3069815504 lite.py:624] Using experimental converter: If you encountered a problem please file a bug. You can opt-out by setting experimental_new_converter=False
2021-08-05 12:23:52.622374: W tensorflow/compiler/mlir/lite/python/tf_tfl_flatbuffer_helpers.cc:313] Ignored output_format.
2021-08-05 12:23:52.622462: W tensorflow/compiler/mlir/lite/python/tf_tfl_flatbuffer_helpers.cc:316] Ignored drop_control_dependency.
Fatal Python error: Segmentation fault

Current thread 0xb6f9aad0 (most recent call first):
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/tensorflow/lite/python/wrap_toco.py", line 38 in wrapped_toco_convert
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/tensorflow/lite/python/convert.py", line 199 in toco_convert_protos
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/tensorflow/lite/python/convert.py", line 574 in toco_convert_impl
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/tensorflow/lite/python/lite.py", line 633 in convert
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/tensorflow/lite/python/lite.py", line 900 in convert
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/tensorflow/lite/python/lite.py", line 1076 in convert
  File "convert_tflite.py", line 45 in save_tflite
  File "convert_tflite.py", line 71 in main
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/absl/app.py", line 258 in _run_main
  File "/home/pi/.virtualenvs/vemodeloYOLOtinyv4/lib/python3.7/site-packages/absl/app.py", line 312 in run
  File "convert_tflite.py", line 76 in <module>
Violación de segmento
```

Figure 69 Error obtenido al momento de la conversión a Tensorflow Lite [Fuente: Los Autores]

```python
def save_tflite():
  converter = tf.lite.TFLiteConverter.from_saved_model(FLAGS.weights)

  if FLAGS.quantize_mode == 'float16':
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    converter.target_spec.supported_types = [tf.compat.v1.lite.constants.FLOAT16]
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS, tf.lite.OpsSet.SELECT_TF_OPS]
    converter.allow_custom_ops = True
  elif FLAGS.quantize_mode == 'int8':
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS, tf.lite.OpsSet.SELECT_TF_OPS]
    converter.allow_custom_ops = True
    converter.representative_dataset = representative_data_gen

  converter.experimental_new_converter=False
  tflite_model = converter.convert()
  open(FLAGS.output, 'wb').write(tflite_model)

  logging.info("model saved to: {}".format(FLAGS.output))
```

Figure 70 Configuración del atributo “experimental_new_converter” del convertidor a “False” [Fuente: Los Autores]

Comando para realizar las detecciones:

Python detectvideo.py --weights ./checkpoints/yolov4-tiny-tflite --size 320 --model yolov4 --video 0 --framework tflite --tiny

Realizadas las conversiones, si se empieza a realizar detecciones el programa después de casi 5 segundos, a diferencia de los 40 segundos de espera que teníamos con

Tensorflow normal, pero la velocidad de inferencia bajo, esto se debió en nuestro caso a que la librería “lite” del Tensorflow que se instaló no era la más óptima para realizar las detecciones en tiempo de ejecución, para resolver este problema se siguen las instrucciones dadas por Tensorflow sobre el paquete “Tensorflow Lite runtime”, que como dice en su documentación, es una fracción del paquete total de Tensorflow que incluye el código mínimo para ejecutar inferencias con Tensorflow Lite, principalmente la clase “Interpreter”. [55]

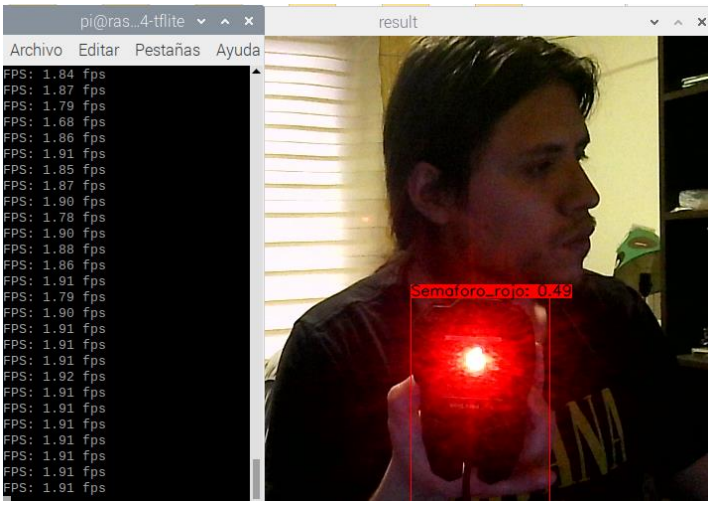


Figure 71 Detecciones realizadas con TensorFlow Lite con el intérprete por defecto [Fuente: Los Autores]

Una vez realizados los pasos [55], se debe importar la librería en el script “detectvideo.py” y comentar la línea de código en donde se esté configurando al interprete y cambiarla por la que se instala, tal y como lo dice la documentación.

```
def main(_argv):
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)
    STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)
    input_size = FLAGS.size
    video_path = FLAGS.video

    print("Video from: ", video_path )
    vid = cv2.VideoCapture(int(video_path)) #Linea editada, convirtiendo a int porque viene en str

    if FLAGS.framework == 'tflite':
        #interpreter = tf.lite.Interpreter(model_path=FLAGS.weights)  #original
        interpreter = tflite.Interpreter(model_path=FLAGS.weights)   #adicionada
        interpreter.allocate_tensors()
        input_details = interpreter.get_input_details()
        output_details = interpreter.get_output_details()
        print(input_details)
        print(output_details)
    else:
        saved_model_loaded = tf.saved_model.load(FLAGS.weights, tags=[tag_constants.SERVING])
        infer = saved_model_loaded.signatures['serving_default']
```

Figure 72 Cambio al nuevo intérprete en el código (línea con el comentario #adicionada) del anterior intérprete (línea con el comentario #original) [Fuente: Los Autores]

2.6.3 Dictador de clases usando los pines GPIO de la Raspberry Pi 4

Para obtener la clase almacenada en el archivo “detecciones.txt” usando los pines GPIO de la Raspberry se usará la librería “gpiozero”.

```
import pyttsx3
from gpiozero import Button
import time
boton = Button(2)

def operacion():

    for t in range(500):
        engine = pyttsx3.init()
        engine.setProperty('voice',"spanish+f4")
        engine.setProperty('rate',150)
        with open("detecciones.txt","r") as f:
            label = f.readline()
            if label == "Semaforo_rojo":
                engine.say("Semaforo rojo")
                if engine.isBusy():
                    print("Reproduciendo...")
                    engine.runAndWait()
                    break

            if label == "Semaforo_verde":
                engine.say("Semaforo verde")
                if engine.isBusy():
                    print("Reproduciendo...")
                    engine.runAndWait()
                    break
        time.sleep(0.01)

while True:
    boton.wait_for_press()
    #print("Se aplasto")
    operacion()
```

Figure 73 Script para obtener la salida de audio con la clase detectada mediante el uso de un pulsante en los pines GPIO de la Raspberry Pi 4 [Fuente: Los Autores]

El algoritmo funciona de la siguiente manera, al aplastar el pulsante, que estará conectado entre el pin GPIO2 y el pin GND de la Raspberry, se enviara la señal para que arranque la función "operación", que ejecutara su contenido por 500 iteraciones en lapsos de 10ms.

CAPITULO 3

3. CONTRUCCIÓN DEL DISPOSITIVO

En este capítulo se explicará cómo es la construcción del dispositivo desde su parte electrónica hasta su parte física que será realizada a través del programa Inventor y la programación que pasará desde Ubuntu por medio de comunicación serial al Arduino mini.

3.1. Arquitectura del Dispositivo

En el sistema de control se encuentra el microcontrolador que para este caso es un arduino Pro mini, conformado por dos sensores ultrasónicos que serán los encargados de obtener las señales de distancia, estos enviarán una señal a los motores de vibración indicando a que dirección debe moverse el usuario.

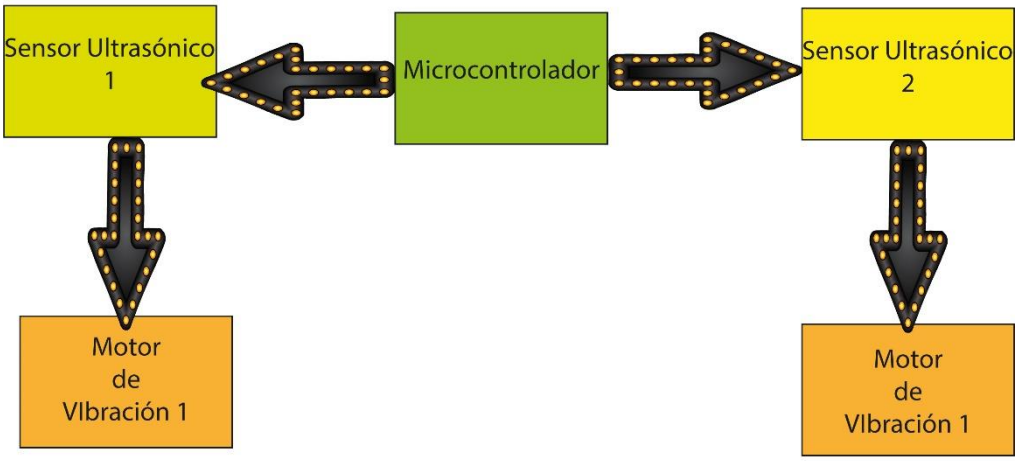


Figure 74 Arquitectura de Dispositivo (Sistema de Control) [Fuentes los Autores]

3.2. Dispositivos de Hardware

3.2.1. Microcontrolador Arduino NANO

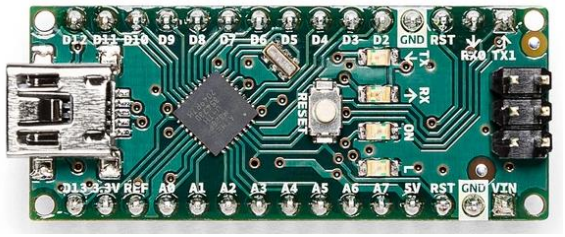


Figure 75 Arduino NANO [56]

El Arduino nano es un microcontrolador que posee una entrada de Mini-B USB con una fuente de alimentación de 5V, contiene 14 pines digitales y 8 entradas analógicas, posee características que permiten hacer controles de PWM, MISO, MOSI, SCK, controles seriales y de I2C, puede ser programado con su propio software llamado Arduino y escogiendo la opción de placa llamada ATmega328, generalmente es usado por su maleabilidad y pequeño tamaño ya que consta de unas medidas de 44cm de largo con 18 cm de alto, lo cual lo vuelven un microcontrolador de gran uso y para espacios reducidos. [56]

3.2.2. Rasberry PI 4 B de 4GB de Memoria RAM

La Rasberry Pi 4 B es una especie de mini Ordenador de bajo coste en el cual se pueden una variedad de cosas entre ellas está el entrenamiento de redes neuronales, servidores de archivos, consolas de juegos retro, enrutadores, bloqueadores de red entre otros. [57]

Especificaciones de la Raspberry Pi 4 modelo B [57]

- Sistema en un chip: Broadcom BCM2711
- CPU: Procesador de cuatro núcleos a 1,5 GHz con brazo Cortex-A72
- GPU: Video Core VI
- Memoria: 1/2/4GB LPDDR4 RAM
- Conectividad: 802.11ac Wi-Fi / Bluetooth 5.0, Gigabit Ethernet
- Vídeo y sonido: 2 x puertos micro-HDMI que admiten pantallas de 4K@60Hz a través de HDMI 2.0, puerto de pantalla MIPI DSI, puerto de cámara MIPI CSI, salida estéreo de 4 polos y puerto de vídeo compuesto.
- Puertos: 2 x USB 3.0, 2 x USB 2.0
- Alimentación: 5V/3A vía USB-C, 5V vía cabezal GPIO
- Expansión: Cabezal GPIO de 40 pines

Figure 76 Rasberry Pi 4 B de 4GB de memoria RAM [57]

3.2.3. Sensores Ultrasónicos MaxSonar EZ1:

Es un sensor ultrasónico capaz de detectar y medir rangos desde distancias cortas a largas con unas medidas que van desde 0 y 6.54metros, Estos sensores tienen integrados en la misma placa el circuito emisor y receptor, por lo que sus dimensiones son menores y facilita el uso de estos dispositivos. [56]

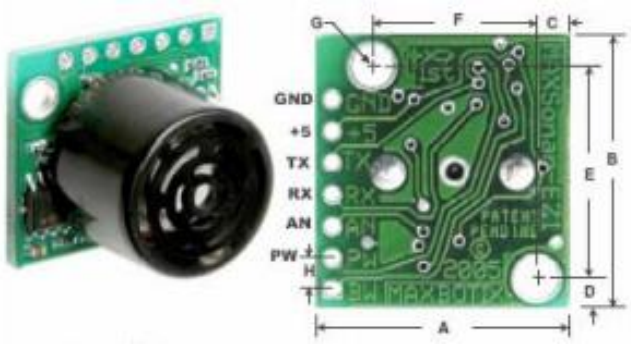


Figure 77 Medidas de Sensor Ultrasónico MaxSonar EZ1 [56]

A	**19,9mm**	**F**	**12,6mm**
B	21,1mm	G	3,1mm
C	2,54mm	H	2,54mm
D	2,54mm	J	16,4mm
E	17,0mm	K	15,5mm

Tabla 3 Tabla de Medidas Sensor Ultrasónico MaxSonar EZI [56]

- **Características** [58]**:**

  a) Frecuencia: 42KHz
  b) Rango detección: 10cm-6m
  c) Tres opciones de toma de datos: Con voltaje Análogo, de forma serial, o por ancho de pulso.
  d) Resolución: 2.5cm
  e) Rango voltajes: 2.5-5.5V
  f) Consumo corriente promedio: 2mA
  g) Dimensiones: 19.9x22.1x15.5mm
  h) Peso: 4.3 gramos

- **Ventajas** [58]**:**
  - Su tamaño es el más pequeño dentro de su categoría
  - Las zonas muertas del sensor desaparecen virtualmente
  - No hay puntos ciegos centrales
  - Haces de gran calidad
  - Orificios de montaje incluidos en la placa de circuito impreso
  - Sensor de distancia de baja potencia, ideal para los sistemas formados por varios sensores alimentados por baterías
  - Activación externa o interna
  - El sensor genera las lecturas directamente, liberando al procesador de la carga de trabajo
  - Ciclos de medición rápidos
  - El usuario puede elegir cualquiera de las 3 salidas del sensor

Características del haz

Las siguientes figuras reflejan los resultados de muestra para los cálculos de patrones de haces en cuadrículas de 30 cm. Se muestra el patrón de detección para [59]:

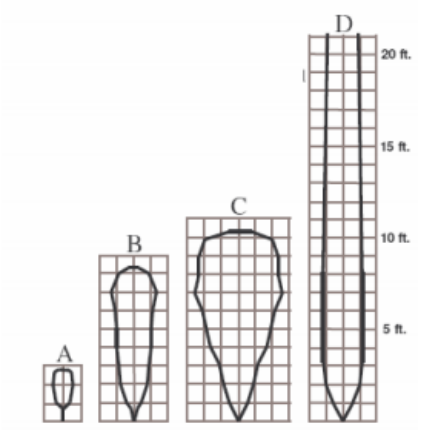


Figure 78 Representación de los haces de luz sensor ultrasónico MaxSonar [59]

A: Cilindro de 0,635cm de diámetro. Tienen un haz muy estrecho con objetos pequeños.

B: Cilindro de 2.54cm de diámetro. Patrón de detección largo y estrecho.

C: Barra de 8.255cm de diámetro. Patrón de detección largo y controlado.

D. Placa de 27.94cm de ancho. Gran capacidad de detección del sensor ultrasónico

3.2.4. Motores de Vibración Ruiling 1034

Generalmente son usados como para artículos telefónicos, Tablet entre otros productos por su uso práctico uso y por el pequeño espacio que ocupan ademas de brindar una comodidad en el la instalación.

Figure 79 Motores de Vibración Ruiling 1034 [60]

- **Especificaciones** [60] **:**

- **Tipo:** micro motor de vibración.
- **Tamaño:** 0.394 x 0.134 in.
- **Tensión nominal:** 3 V CC.
- **Corriente nominal:** 80 mA
- **Velocidad:** 12000 rpm.
- **Conexión del cable:** rojo (polo positivo), azul (polo negativo).

- **Características** [60]**:**
  - Rendimiento estable, potencia fuerte, larga vida útil.
  - Amplia aplicación: se utiliza en teléfono móvil, Bluetooth, mini ventilador y otros dispositivos electrónicos.

3.3. Diseño y Construcción del Dispositivo en Inventor

3.3.1. Diseño de Gafas

- **Cobertura Delantera**

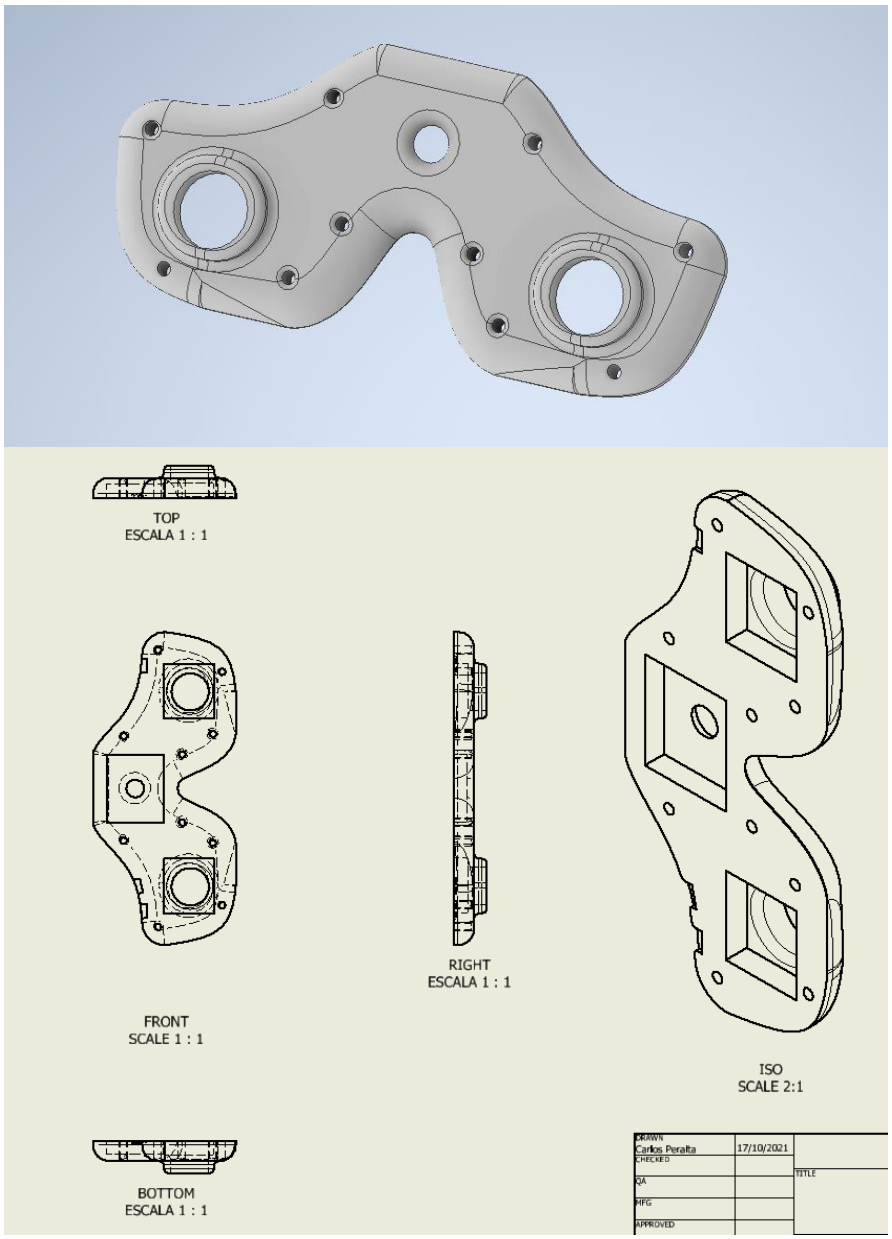


Figure 80 Modelo 3D y Plano de Cobertura Delantera [Fuente: Los Autores]

- **Cobertura Posterior**

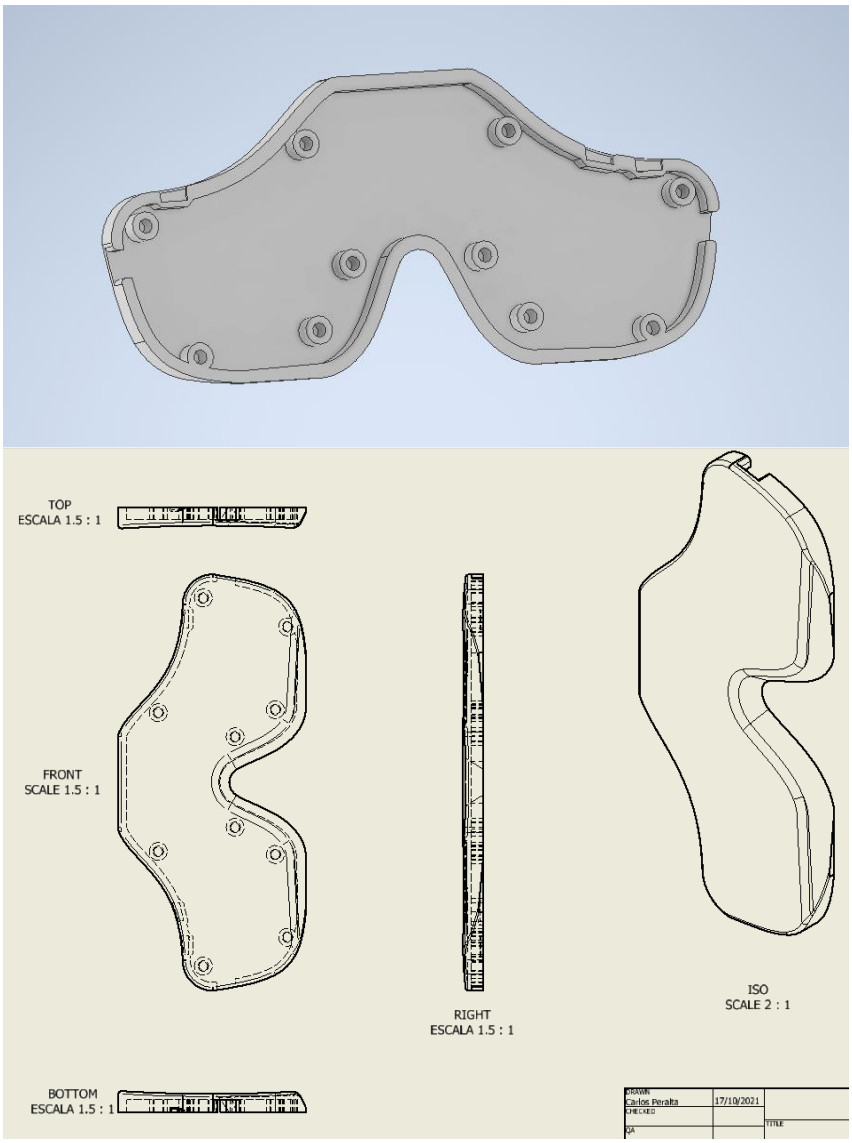


Figure 81 Modelo 3D y Plano de Cobertura Posterior [Fuente: Los Autores]

- **Soporte de Componentes**

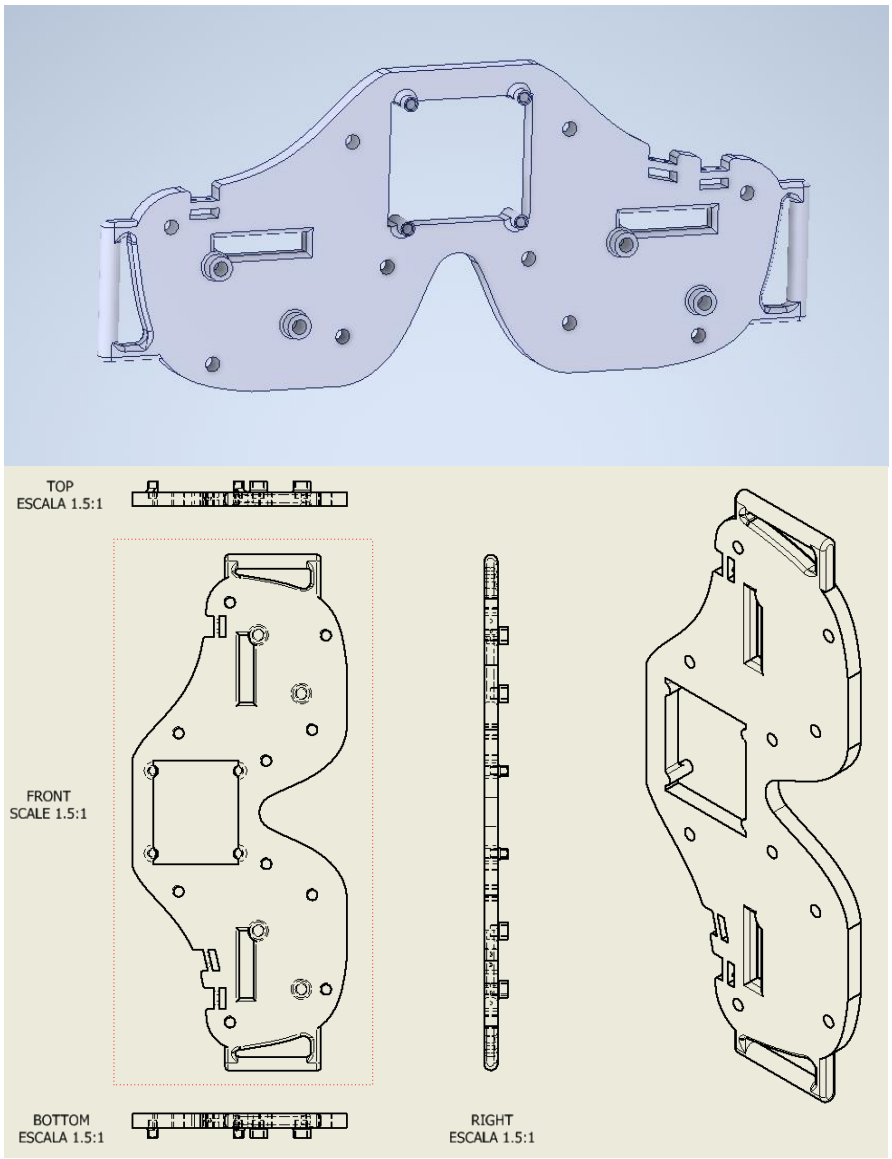


Figure 82 Modelo 3D y Plano de Soporte de Componentes [Fuente: Los Autores]

- **Ensamble de Piezas para las Gafas**

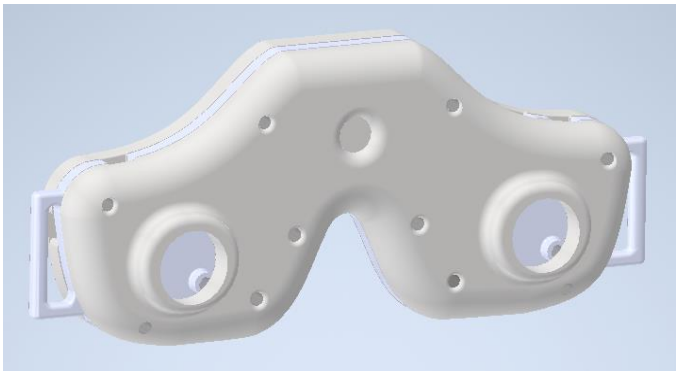


Figure 83 Modelo 3D de Ensamble de Piezas para la construcción de las gafas [Fuente: Los Autores]

3.3.2. Diseño de Carcasa para Controlador

- **Case para el Controlador**

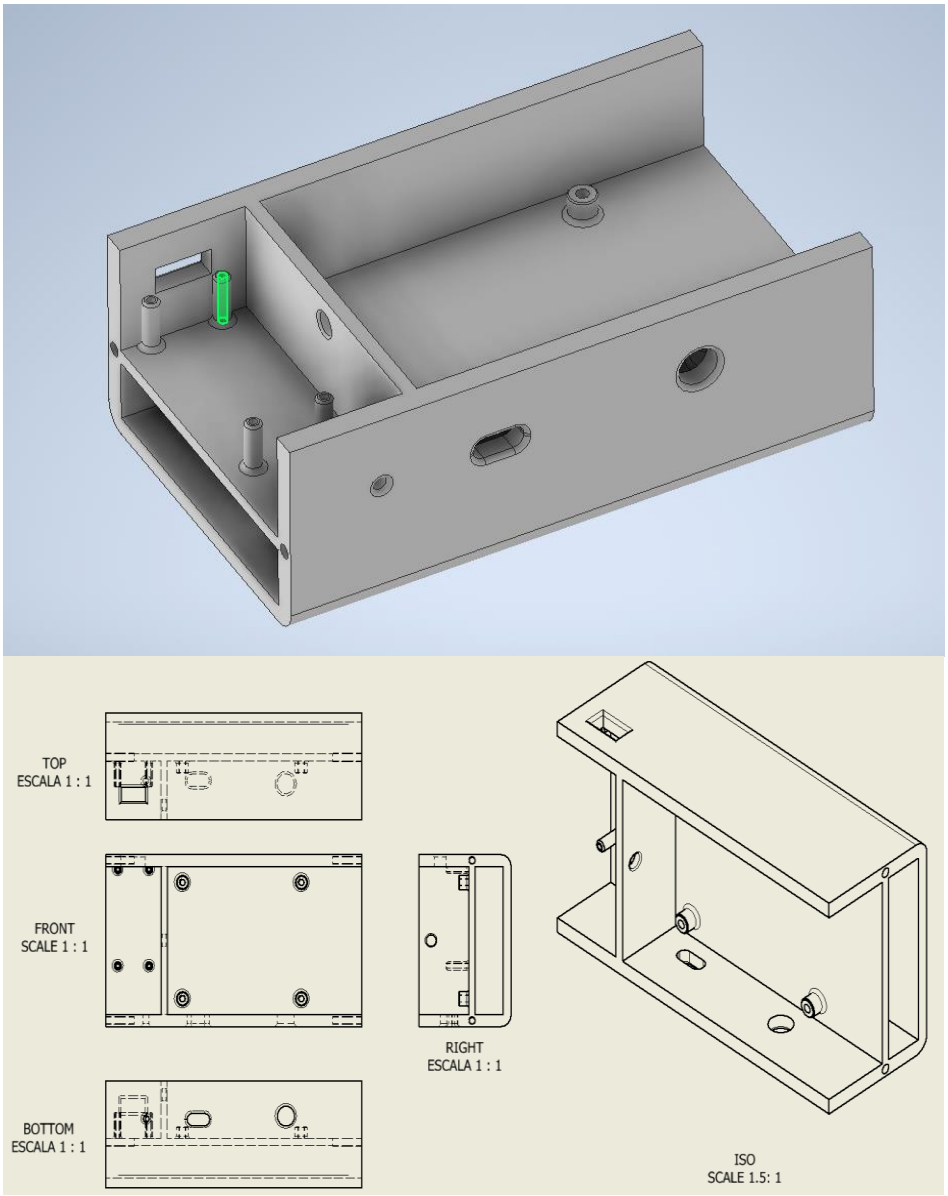


Figure 84 Modelo 3D y Plano del Case para el Controlador [Fuente: Los Autores]

- **Parte superior del Case para el Controlador**

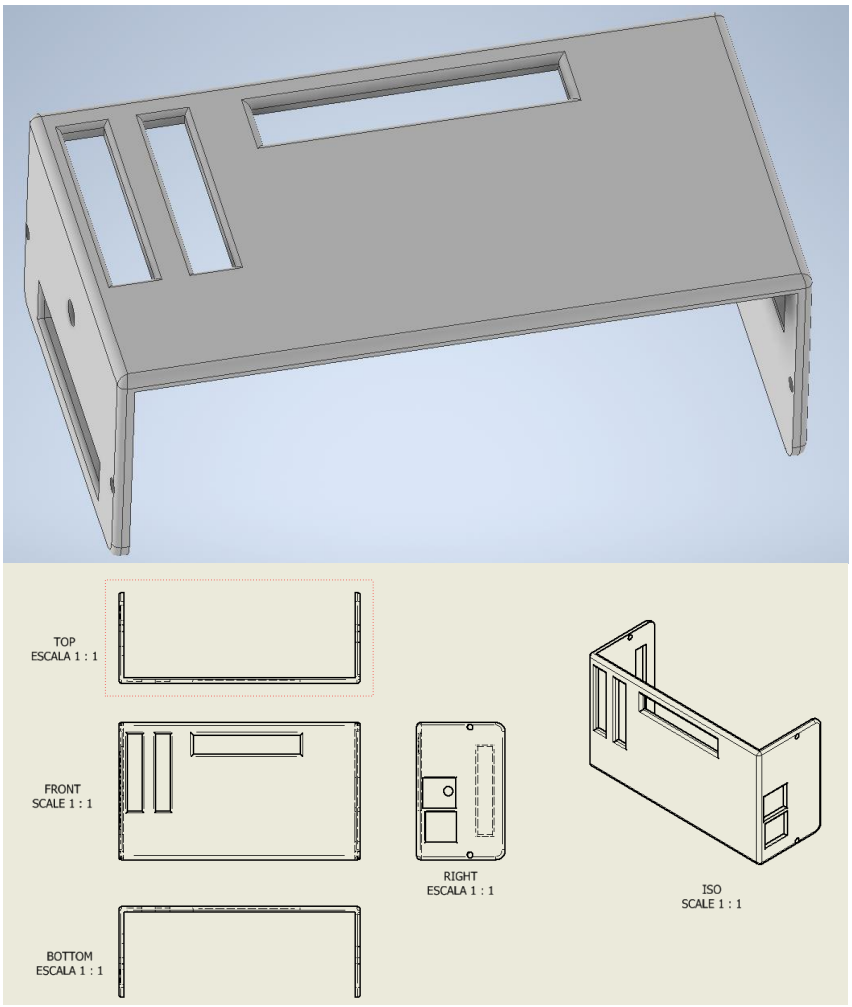


Figure 85 Modelo 3D y Plano de la parte superior del Case para el Controlador [Fuente: Los Autores]

- **Ensamble de Piezas para el Case**

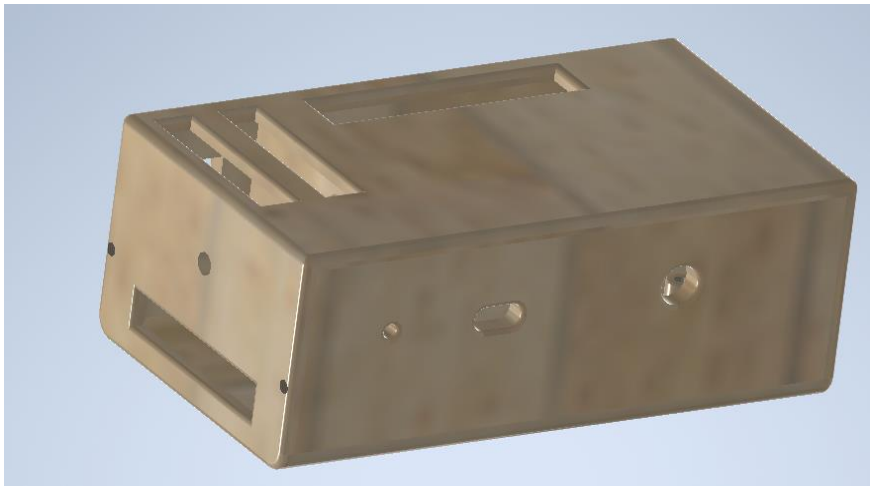


Figure 86 Modelo 3D de Ensamble de Piezas para la construcción del Case [Fuente: Los Autores]

CAPITULO 4

4. ANÁLISIS DE RESULTADOS

4.1 Selección de los mejores pesos

Darknet facilita el cálculo de la precisión promedio media (mAP) con el siguiente comando, ejemplificado para este caso y con el primer archivo de pesos generado:

darknet.exe detector map data\semaforos_peatonales.data cfg\yolov4-tiny-custom.cfg backup\yolov4-tiny-custom_500.weights

Revisando el resultado entregado por darknet se observa algunas métricas y valores que calcula [25], a continuación, se explican las más importantes:

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 30306, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 7.32%          (TP = 59, FP = 175)
class_id = 1, name = Semaforo_verde, ap = 2.93%         (TP = 16, FP = 144)

 for conf_thresh = 0.25, precision = 0.19, recall = 0.06, F1-score = 0.09
 for conf_thresh = 0.25, TP = 75, FP = 319, FN = 1139, average IoU = 11.12 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.051250, or 5.12 %
Total Detection Time: 141 Seconds
```

Figure 87 Resultados obtenidos para las 500 iteraciones [Fuente: Los Autores]

- 1212: Numero de imágenes de prueba analizadas para determinar el valor de las métricas.
- TP: Verdaderos positivos
- FP: Falsos positivos
- FN: Falsos Negativos
- Average IoU: Intersección sobre Unión promedio.
- Precision: Es el resultado de $TP/(TP + FP)$
- recall: Es el resultado de $TP/(TP + FN)$

Las primeras 500 iteraciones el modelo tiene una precisión promedio media (mAP) de 5.12%, una precisión del 20% y una Intersección sobre Unión promedio de

11.12%; estos valores son muy bajos e indican que hasta este punto el modelo se encuentra en un estado de “underfitting” o desajustado.

El mismo análisis se lo realiza para las siguientes 500 iteraciones y sucesivamente hasta llegar hasta las 4000 iteraciones, buscando la mejor precisión promedio media (mAP) para nuestro modelo.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 5830, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 72.06%          (TP = 452, FP = 296)
class_id = 1, name = Semaforo_verde, ap = 61.23%         (TP = 384, FP = 204)

 for conf_thresh = 0.25, precision = 0.63, recall = 0.69, F1-score = 0.66
 for conf_thresh = 0.25, TP = 836, FP = 500, FN = 378, average IoU = 41.63 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.666467, or 66.65 %
Total Detection Time: 92 Seconds
```

Figure 88 Resultados obtenidos para las 1000 iteraciones [Fuente: Los Autores]

En este caso, con el archivo “yolov4-tiny-custom_1000.weights” se obtiene mejores resultados. Una precisión promedio media (mAP) de 66.65%, una precisión de 63% y una intersección sobre unión promedio (Average IoU) de 41.63%.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 3694, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 90.19%          (TP = 481, FP = 208)
class_id = 1, name = Semaforo_verde, ap = 95.54%         (TP = 517, FP = 55)

 for conf_thresh = 0.25, precision = 0.79, recall = 0.82, F1-score = 0.81
 for conf_thresh = 0.25, TP = 998, FP = 263, FN = 216, average IoU = 56.07 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.928647, or 92.86 %
Total Detection Time: 94 Seconds
```

Figure 89 Resultados obtenidos para las 1500 iteraciones [Fuente: Los Autores]

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 4263, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 96.39%          (TP = 608, FP = 128)
class_id = 1, name = Semaforo_verde, ap = 95.15%         (TP = 536, FP = 52)

 for conf_thresh = 0.25, precision = 0.86, recall = 0.94, F1-score = 0.90
 for conf_thresh = 0.25, TP = 1144, FP = 180, FN = 70, average IoU = 63.99 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.957691, or 95.77 %
Total Detection Time: 85 Seconds
```

Figure 90 Resultados obtenidos para las 2000 iteraciones [Fuente: Los Autores]

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 2863, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 97.14%          (TP = 590, FP = 86)
class_id = 1, name = Semaforo_verde, ap = 96.64%         (TP = 542, FP = 26)

 for conf_thresh = 0.25, precision = 0.91, recall = 0.93, F1-score = 0.92
 for conf_thresh = 0.25, TP = 1132, FP = 112, FN = 82, average IoU = 67.45 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.968873, or 96.89 %
Total Detection Time: 88 Seconds
```

Figure 91 Resultados obtenidos para las 2500 iteraciones [Fuente: Los Autores]

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 3620, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 95.04%          (TP = 618, FP = 107)
class_id = 1, name = Semaforo_verde, ap = 95.17%         (TP = 538, FP = 42)

 for conf_thresh = 0.25, precision = 0.89, recall = 0.95, F1-score = 0.92
 for conf_thresh = 0.25, TP = 1156, FP = 149, FN = 58, average IoU = 64.68 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.951021, or 95.10 %
Total Detection Time: 84 Seconds
```

Figure 92 Resultados obtenidos para las 3000 iteraciones [Fuente: Los Autores]

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 2259, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 98.51%          (TP = 630, FP = 37)
class_id = 1, name = Semaforo_verde, ap = 98.34%         (TP = 551, FP = 15)

 for conf_thresh = 0.25, precision = 0.96, recall = 0.97, F1-score = 0.97
 for conf_thresh = 0.25, TP = 1181, FP = 52, FN = 33, average IoU = 74.70 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.984283, or 98.43 %
Total Detection Time: 89 Seconds
```

Figure 93 Resultados obtenidos para las 3500 iteraciones [Fuente: Los Autores]

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1212
 detections_count = 2092, unique_truth_count = 1214
class_id = 0, name = Semaforo_rojo, ap = 98.94%          (TP = 637, FP = 26)
class_id = 1, name = Semaforo_verde, ap = 98.54%         (TP = 552, FP = 14)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.98, F1-score = 0.97
 for conf_thresh = 0.25, TP = 1189, FP = 40, FN = 25, average IoU = 75.76 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.987385, or 98.74 %
Total Detection Time: 95 Seconds
```

Figure 94 Resultados obtenidos para las 4000 iteraciones [Fuente: Los Autores]

4.2 Detecciones realizadas con Darknet

El resultado de ejecutar el código anteriormente mencionado será la separación de la data en dos conjuntos de entrenamiento y de prueba, con su respectiva etiqueta, esto se hace para que sea más fácil localizar las imágenes de prueba y entrenamiento. A continuación, se mostrará el resultado de las detecciones en algunas imágenes de prueba escogidas al azar.

Imagen # 1

```
Loading weights from backup\yolov4-tiny-custom_4000.weights...
 seen 64, trained: 256 K-images (4 Kilo-batches_64)
Done! Loaded 38 layers from weights-file
Enter Image Path: C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_2997_flip.jpg
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_2997_flip.jpg: Predicted in 420.63600
0 milli-seconds.
Semaforo_rojo: 91%
```

Figure 95 Imagen de detección 1 [Fuente: Los Autores]

Imagen #2

```
Enter Image Path: C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_3745.jpg
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_3745.jpg: Predicted in 397.344000 mil
li-seconds.
Semaforo_verde: 81%
```

Figure 96 Imagen de detección 2 [Fuente: Los Autores]

Imagen #3

```
Enter Image Path: C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_2324_flip.jpg
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_2324_flip.jpg: Predicted in 387.45500
0 milli-seconds.
Semaforo_rojo: 92%
```

Figure 97 Imagen de detección 3 [Fuente: Los Autores]

Imagen #4

```
Enter Image Path: C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_2385_rot_-15.jpg
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_2385_rot_-15.jpg: Predicted in 394.73
3000 milli-seconds.
Semaforo_rojo: 94%
```

Figure 98 Imagen de detección 5 [Fuente: Los Autores]

Imagen #5

```
Enter Image Path: C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_4212_rot_15.jpg
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
C:\Users\fbrit\Desktop\DOCUMENTOS\modelo1YOLOv4-Tiny\darknet\build\darknet\x64\data\custom\test\imagen_4212_rot_15.jpg: Predicted in 375.456
000 milli-seconds.
Semaforo_verde: 94%
```

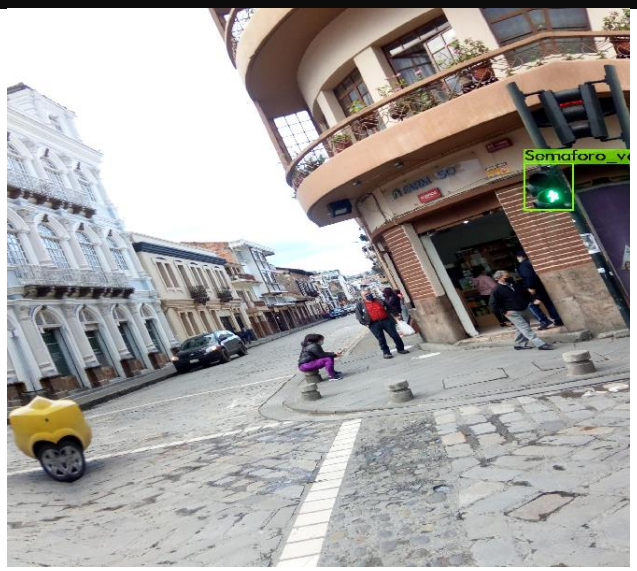


Figure 99 Imagen de detección 6 [Fuente: Los Autores]

4.3 Adaptación del modelo en la Raspberry Pi 4

Realizado los cambios mencionados anteriormente con las detecciones en tiempo real para analizar el resultado, el código que se ejecutará es el mismo que se usa para realizar detecciones en Windows, solo que al no usar "darknet.exe" al inicio del comando, se usará ". /Darknet".

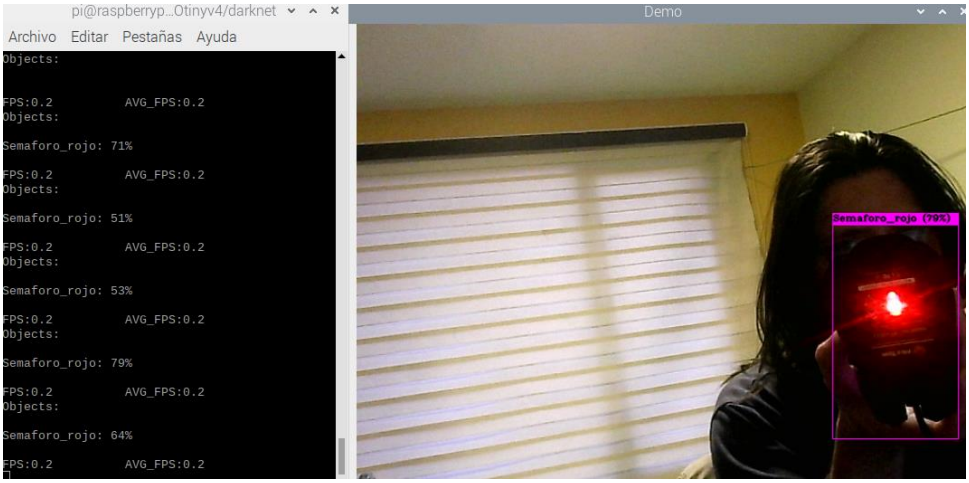


Figure 100 Detecciones realizadas con darknet en la Raspberry Pi 4 [Fuente: Los Autores]

Se puede apreciar que la diferencia de las detecciones realizadas con la Raspberry Pi 4 en comparación con el ordenador con el que se entrena la red, es la velocidad de los procesos de inferencia, en la Raspberry se ejecutan las detecciones a un promedio de 0.2 FPS.

4.4 Conversión al modelo de Tensorflow

Realizado el cambio se ejecutan las detecciones con el siguiente comando:

Python detectvideo.py --weights ./checkpoints/yolov4-tiny --size 320 --model yolov4 --video 0 –tiny

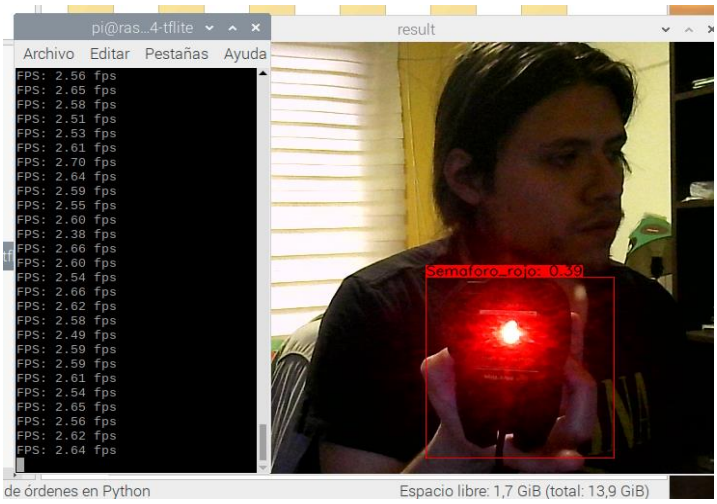


Figure 101 Detecciones usando Tensorflow con la arquitectura YOLOv4-tiny en Raspberry Pi 4 [Fuente: Los Autores]

Como se puede observar la velocidad de inferencia ha mejorado en una media de 2.5 FPS, pero ha demorado como mínimo 40 segundos en arrancar las detecciones. En la figura anterior se puede observar y comprobar que sigue confundiendo la parte inferior del mouse con un semáforo peatonal en estado de señalización rojo, pero como mencionamos antes, ese problema se solventa elevando el umbral de detección. En la figura anterior el umbral de detección era 0, es decir, detecta todo lo que tenga un valor de confianza entre 0 y 1.

4.5 Conversión al modelo de TensorFlow Lite

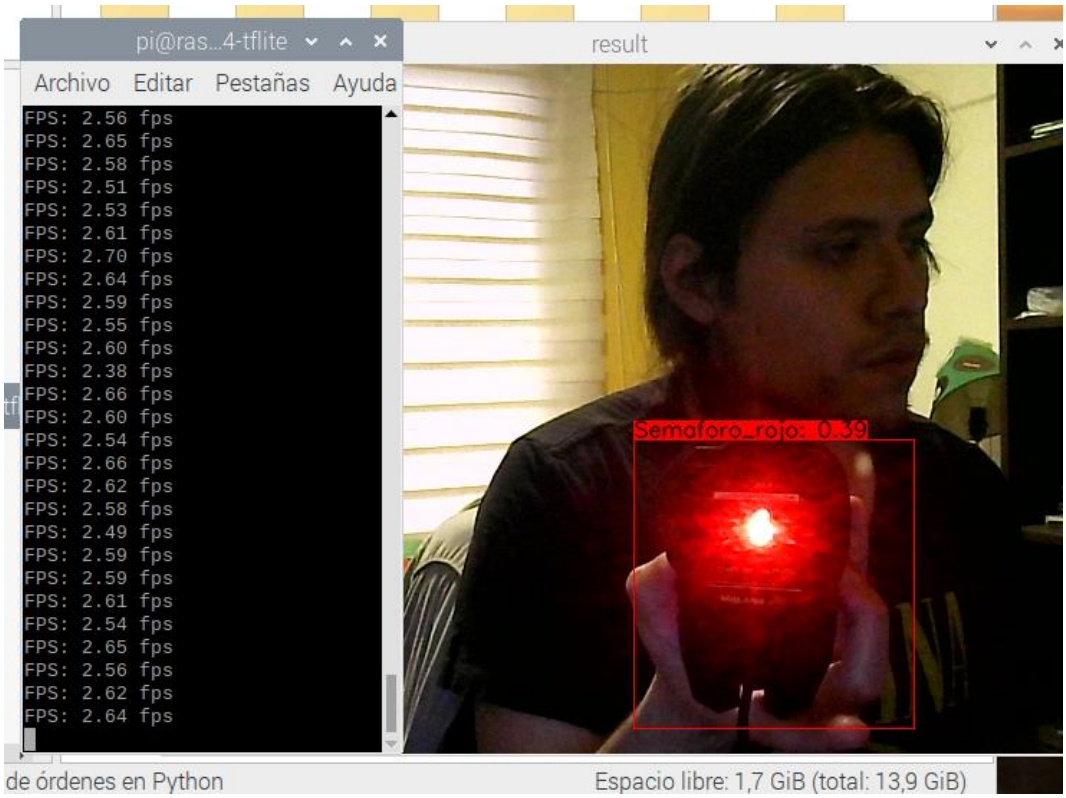


Figure 102 Detecciones realizadas con el nuevo intérprete [Fuente: Los Autores]

Como se puede observar, las detecciones volvieron a la media de 2.5 FPS pero con la diferencia de que ahora, como el modelo es más pequeño, arranca de una manera más rápida.

4.6 Detecciones realizadas con TensorFlow Lite

En esta sección se realizarán detecciones en imágenes con el script “detect.py”, con la intención de comparar las detecciones realizadas con Darknet, entonces se usarán las mismas imágenes.

Es importante mencionar que se realizó una “customización” al script para simplemente ingresar el nombre de la imagen que se desea pasar por la red, y de la cual se quiere obtener las predicciones.

```
.py ✖ utils.py ✖
import cv2
import numpy as np
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

from gpiozero import Button #Libreria para el control de los pines GPIO

#flags.DEFINE_string('framework', 'tf', '(tf, tflite, trt')   #original
flags.DEFINE_string('framework', 'tflite','framework')                    #customizada
#flags.DEFINE_string('weights', './checkpoints/yolov4-416',   #original
                    #'path to weights file')                  #original
flags.DEFINE_string('weights', './checkpoints/yolov4-tiny.tflite','path to weights file')    #customizada
#flags.DEFINE_integer('size', 416, 'resize images to')                  #original
flags.DEFINE_integer('size', 320,'resize images to')                                  #customizada
#flags.DEFINE_boolean('tiny', False, 'yolo or yolo-tiny')               #original
flags.DEFINE_boolean('tiny', True, 'version Tiny?')                                  #customizada
#flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')             #original
flags.DEFINE_string('model', 'yolov4','yolov3 or yolov4')                             #customizada
flags.DEFINE_string('image', './data/kite.jpg', 'path to input image')  #original
#flags.DEFINE_string('output', 'result.png', 'path to output image')      #original
flags.DEFINE_string('output', 'resultado.jpg','path to output image')   #customizada
flags.DEFINE_float('iou', 0.45, 'iou threshold')                         #original
flags.DEFINE_float('score', 0.25, 'score threshold')                     #original
```

Figure 103 Customización de las banderas en el script "detect.py" [Fuente: Los Autores]

Imagen #1

```
(vemodeloYOLOtinyv4) pi@raspberrypi:~/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite $ python detect.py --image imag
en_2997_flip.jpg
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
[{'name': 'input_1', 'index': 289, 'shape': array([  1, 320, 320,   3]), 'shape_signature': array([  1, 320, 320,   3])
, 'dtype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=flo
at32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
[{'name': 'Identity', 'index': 0, 'shape': array([   1, 1500,    4]), 'shape_signature': array([   1, 1500,    4]), 'dt
ype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32)
, 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}, {'name': 'Identity_1',
'index': 1, 'shape': array([   1, 1500,    2]), 'shape_signature': array([   1, 1500,    2]), 'dtype': <class 'numpy.fl
oat32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array
([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
Clase: Semaforo_rojo: 0.93
```

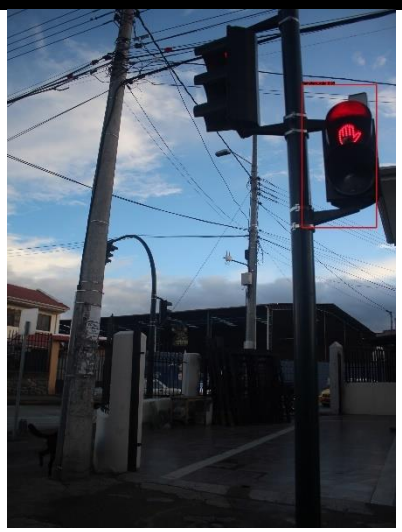


Figure 104 Detección en la imagen 1 utilizando TensorFlow Lite [Fuente: Los Autores]

Imagen # 2

```
(vemodeloYOLOtinyv4) pi@raspberrypi:~/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite $ python detect.py --image imag
en_3745.jpg
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
[{'name': 'input_1', 'index': 289, 'shape': array([  1, 320, 320,   3]), 'shape_signature': array([  1, 320, 320,   3])
, 'dtype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=flo
at32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
[{'name': 'Identity', 'index': 0, 'shape': array([   1, 1500,    4]), 'shape_signature': array([   1, 1500,    4]), 'dt
ype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32)
, 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}, {'name': 'Identity_1',
'index': 1, 'shape': array([   1, 1500,    2]), 'shape_signature': array([   1, 1500,    2]), 'dtype': <class 'numpy.fl
oat32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array
([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
Clase: Semaforo verde: 0.84
```

Figure 105 Detección en la imagen 2 utilizando TensorFlow Lite [Fuente: Los Autores]

Imagen #3

```
(vemodeloYOLOtinyv4) pi@raspberrypi:~/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite $ python detect.py --image imag
en_2324_flip.jpg
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
[{'name': 'input_1', 'index': 289, 'shape': array([  1, 320, 320,   3]), 'shape_signature': array([  1, 320, 320,   3])
, 'dtype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=flo
at32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
[{'name': 'Identity', 'index': 0, 'shape': array([   1, 1500,    4]), 'shape_signature': array([   1, 1500,    4]), 'dt
ype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32)
, 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}, {'name': 'Identity_1',
'index': 1, 'shape': array([   1, 1500,    2]), 'shape_signature': array([   1, 1500,    2]), 'dtype': <class 'numpy.fl
oat32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array
([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
Clase: Semaforo_rojo: 0.88
```

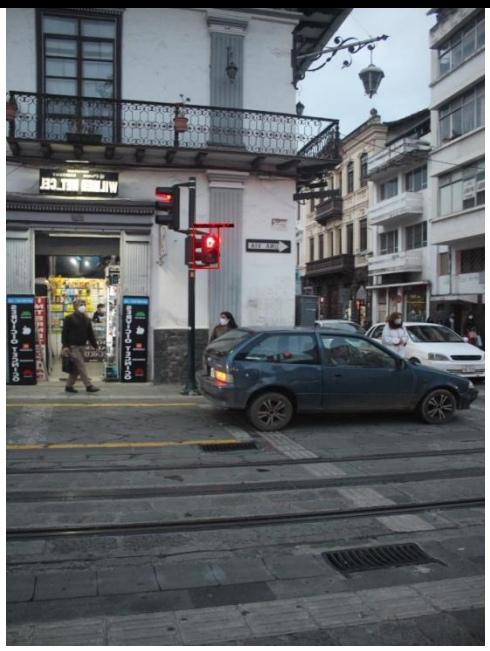


Figure 106 Detección en la imagen 3 utilizando TensorFlow Lite [Fuente: Los Autores]

Imagen #4

```
(vemodeloYOLOtinyv4) pi@raspberrypi:~/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite $ python detect.py --image imag
en_2385_rot_-15.jpg
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
[{'name': 'input_1', 'index': 289, 'shape': array([  1, 320, 320,   3]), 'shape_signature': array([  1, 320, 320,   3])
, 'dtype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=flo
at32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
[{'name': 'Identity', 'index': 0, 'shape': array([   1, 1500,    4]), 'shape_signature': array([   1, 1500,    4]), 'dt
ype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32)
, 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}, {'name': 'Identity_1',
'index': 1, 'shape': array([   1, 1500,    2]), 'shape_signature': array([   1, 1500,    2]), 'dtype': <class 'numpy.fl
oat32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array
([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
Clase: Semaforo_rojo: 0.94
```

Figure 107 Detección en la imagen 4 utilizando TensorFlow Lite [Fuente: Los Autores]

Imagen #5

```
(vemodeloYOLOtinyv4) pi@raspberrypi:~/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite $ python detect.py --image imag
en_4212_rot_15.jpg
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
[{'name': 'input_1', 'index': 289, 'shape': array([  1, 320, 320,   3]), 'shape_signature': array([  1, 320, 320,   3])
, 'dtype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=flo
at32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
[{'name': 'Identity', 'index': 0, 'shape': array([   1, 1500,    4]), 'shape_signature': array([   1, 1500,    4]), 'dt
ype': <class 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32)
, 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}, {'name': 'Identity_1',
'index': 1, 'shape': array([   1, 1500,    2]), 'shape_signature': array([   1, 1500,    2]), 'dtype': <class 'numpy.fl
oat32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array
([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
Clase: Semaforo_verde: 0.95
```

Figure 108 Detección en la imagen 5 utilizando TensorFlow Lite [Fuente: Los Autores]

4.7 Detecciones en imágenes de entrada con "detect.py"

Al analizar la velocidad de inferencia obtenidas en cada framework utilizado, se determina que el más óptimo, para la Raspberry Pi 4, es "Tensorflow Lite", pero debido al tiempo de respuesta por cada imagen que entra a la red en tiempo real, se toma la decisión de plantear el sistema como un dispositivo que capture la imagen al accionar una pulsante y posteriormente dicha imagen pase por la red para su correspondiente predicción. Esto ahorrará recursos tanto energéticos como computacionales a la Raspberry debido a que no deberá estar procesando imágenes todo el tiempo.

Dentro del repositorio que se obtiene descargando [53], se encuentra un script llamado "detect.py" que básicamente permite ingresar una imagen y predecir la clase que esa imagen contiene, además de escribir una nueva imagen con la caja delimitadora alrededor del objeto. Este algoritmo llama a una función, dentro del script "utils.py", que se llama

“draw_bbox” a la cual se le pasan varios parámetros, entre ellos las clases que estamos detectando, es aquí en esta función donde se edita algunas líneas de código para ajustar el algoritmo a nuestras precisas necesidades.

El primer cambio dentro de “utils.py” será la importación de la librería “pyttsx3”.

```
import cv2
import random
import colorsys
import numpy as np
import tensorflow as tf
from core.config import cfg

import pyttsx3 #Importamos la nueva libreria
```

Figure 109 Importación de la librería dentro de “utils.py” [Fuente: Los Autores]

Se crea una lista vacía llamada “clases_puntajes” en donde se almacenarán los pares de datos que se necesitan, en este caso, el score y el nombre de esa clase. Esto servirá para obtener la clase con mayor valor de confianza y enviar esa etiqueta a las funciones de la librería “pyttsx3” para su respectiva salida de audio.

```
def draw_bbox(image, bboxes, classes=read_class_names(cfg.YOLO.CLASSES), show_label=True):
    num_classes = len(classes)
    image_h, image_w, _ = image.shape
    hsv_tuples = [(1.0 * x / num_classes, 1., 1.) for x in range(num_classes)]
    colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
    colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)), colors))
    random.seed(0)
    random.shuffle(colors)
    random.seed(None)

    out_boxes, out_scores, out_classes, num_boxes = bboxes

    clases_puntajes = []
    engine = pyttsx3.init()
    engine.setProperty('voice',"spanish")
    engine.setProperty('rate',150)

    if num_boxes[0] == 0:
        #********Codigo para usar detectvideo.py************
        #with open("detecciones.txt","w") as txt:
            #txt.write("")
            #txt.close()
        #**************************************************
        #********Codigo para usar detect.py*****************
        engine.say("No se encontraron semáforos")
        engine.runAndWait()
        #**************************************************
    for i in range(num_boxes[0]):
```

Figure 110 Adición de funciones extras en la función “draw_bbox” [Fuente: Los Autores]

Dentro de bucle es en donde se irán adicionando estos pares de datos que mencionamos antes en la lista que hemos creado.

```
for i in range(num_boxes[0]):
    if int(out_classes[0][i]) < 0 or int(out_classes[0][i]) > num_classes: continue
    coor = out_boxes[0][i]
    coor[0] = int(coor[0] * image_h)
    coor[2] = int(coor[2] * image_h)
    coor[1] = int(coor[1] * image_w)
    coor[3] = int(coor[3] * image_w)

    coor_central_x = coor[3]-coor[1]
    coor_central_y = coor[2]-coor[0]

    #print("x1: %.1f , y1: %.1f" %(coor[1],coor[0]))
    #print("x2: %.1f , y2: %.1f" %(coor[3],coor[2]))

    fontScale = 0.5
    score = out_scores[0][i]
    class_ind = int(out_classes[0][i])
    #with open("detecciones.txt","w") as txt:
        #txt.write(classes[class_ind]+" "+str(score))
        #txt.close()

    #**************Codigo para usar detect.py****************

    clases_puntajes.append([classes[class_ind],score])

    #*******************************************************
```

Figure 111 Adición de los pares de datos dentro de la lista "clases_puntajes" [Fuente: Los Autores]

Finalmente, fuera del iterador "for" se crea otro condicional con la intención de elegir a la etiqueta que pasara a ser usada por las funciones del "pyttsx3", que obviamente deberá ser la clase que tenga mayor valor de confianza.

```
if len(clases_puntajes)>0:
    maximo_score = max(clases_puntajes)
    print(maximo_score)
    if maximo_score[0]=="Semaforo_rojo":
        engine.say("Semáforo rojo")
        engine.runAndWait()
    if maximo_score[0]=="Semaforo_verde":
        engine.say("Semáforo verde")
        engine.runAndWait()


return image
```

Figure 112 Condicional para obtener la etiqueta con el mayor valor de confianza [Fuente: Los Autores]

```
(vemodeloYOLOtinyv4) pi@raspberrypi:~/Desktop/modeloYOLOtinyv4/tensorflow-yolov4
-tflite $ python detect.py --image imagen_3621_rot_10.jpg
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
[{'name': 'input_1', 'index': 289, 'shape': array([  1, 320, 320,   3]), 'shape_
signature': array([  1, 320, 320,   3]), 'dtype': <class 'numpy.float32'>, 'quan
tization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float
32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity
_parameters': {}}]
[{'name': 'Identity', 'index': 0, 'shape': array([   1, 1500,    4]), 'shape_sig
nature': array([   1, 1500,    4]), 'dtype': <class 'numpy.float32'>, 'quantizat
ion': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32),
'zero_points': array([], dtype=int32), 'quantized_dimension': 0}, 'sparsity_para
meters': {}}, {'name': 'Identity_1', 'index': 1, 'shape': array([   1, 1500,
2]), 'shape_signature': array([   1, 1500,    2]), 'dtype': <class 'numpy.float3
2'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], d
type=float32), 'zero_points': array([], dtype=int32), 'quantized_dimension': 0},
 'sparsity_parameters': {}}]
Clase: Semaforo_verde: 0.86
Clase: Semaforo_verde: 0.85
El bbox con mayor score es:  ['Semaforo_verde', 0.863639]
```

Figure 113Detección de dos objetos y selección de la clase con mayor score [Fuente: Los Autores]

Figure 114 Detección de dos objetos en una imagen de la misma clase, pero diferente score [Fuente: Los Autores]

Finalmente se integra un pequeño fragmento de código para obtener la imagen de la webcam con OpenCV cada vez que se presione el pulsante conectado a los pines GPIO2 y GND de la Raspberry Pi 4.

```
def main(_argv):
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)
    STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)
    input_size = FLAGS.size
#     image_path = FLAGS.image    #original
    cap = cv2.VideoCapture(0)
    boton = Button(2) #boton para capturar imagen
    engine = pyttsx3.init()
    engine.setProperty('voice',"spanish")
    engine.setProperty('rate',150)
    engine.say("Todo listo")
    engine.runAndWait()

    while True:
        while True:
            ret, frame = cap.read()
            if ret == True:
                #cv2.imshow("Camara", frame)
                #cv2.waitKey(1)
                if boton.is_pressed:
                    cv2.imwrite("imagen_semaforo.jpg",frame)
                    break

        #original_image = cv2.imread(image_path)   #original
        original_image = cv2.imread("imagen_semaforo.jpg")  #customizada
        original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
```

Figure 115 Fragmento de código para obtener la imagen de la webcam con OpenCV [Fuente: Los Autores]

4.8 Arranque automático del Script "detect.py" en la Raspberry Pi 4

Es importante ejecutar el script de las detecciones de forma automática cuando la Raspberry haya acabado de cargar todos sus servicios, incluyendo los de la interfaz gráfica. Para realizar esta tarea, la Raspberry inicia automáticamente los archivos ".desktop" dentro de la dirección "/etc/xdg/autostart/" y será en esa dirección en donde crearemos un archivo propio para ejecutar automáticamente el script de las detecciones "detect.py".

```
pi@raspberrypi: ~
Archivo  Editar  Pestañas  Ayuda
  GNU nano 3.2                    /etc/xdg/autostart/semaforos.desktop

[Desktop Entry]
Name=Detecciones
Exec=/home/pi/.virtualenvs/vemodeloYOLOtinyv4/bin/python3 /home/pi/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite/detect.py
```

Figure 116 Contenido de "semaforos.desktop" [Fuente: Los Autores]

Para evitar conflictos y problemas con las direcciones en algunos scripts se cambian sus rutas relativas a sus rutas absolutas.

```
#flags.DEFINE_string('framework', 'tf', '(tf, tflite, trt')   #original
flags.DEFINE_string('framework', 'tflite','framework')                    #customizada
#flags.DEFINE_string('weights', './checkpoints/yolov4-416',   #original
                    #'path to weights file')                  #original
flags.DEFINE_string('weights', '/home/pi/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite/checkpoints/yolov4-tiny.tflite','path to weights file')
#flags.DEFINE_integer('size', 416, 'resize images to')                #original
flags.DEFINE_integer('size', 320,'resize images to')                                  #customizada
#flags.DEFINE_boolean('tiny', False, 'yolo or yolo-tiny')             #original
flags.DEFINE_boolean('tiny', True, 'version Tiny?')                                  #customizada
#flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')           #original
flags.DEFINE_string('model', 'yolov4','yolov3 or yolov4')                             #customizada
# flags.DEFINE_string('image', './data/kite.jpg', 'path to input image')  #original
#flags.DEFINE_string('output', 'result.png', 'path to output image')      #original
#flags.DEFINE_string('output', '/home/pi/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite/resultado.jpg','path to output image')   #customizada
flags.DEFINE_float('iou', 0.45, 'iou threshold')                        #original
flags.DEFINE_float('score', 0.25, 'score threshold')                    #original
```

Figure 117 Resumen de banderas en "detect.py" y cambio a ruta absoluta de los pesos de la red [Fuente: Los Autores]

```
__C.YOLO.CLASSES              = "/home/pi/Desktop/modeloYOLOtinyv4/tensorflow-yolov4-tflite/data/classes/semaforos_peatonales.names" #coco.names
__C.YOLO.ANCHORS              = [12,16, 19,36, 40,28, 36,75, 76,55, 72,146, 142,110, 192,243, 459,401]
```

Figure 118 Cambio a ruta absoluta en "config.py" [Fuente: Los Autores]

4.9 Encendido y Apagado de la Raspberry Pi 4

Generalmente apagar la Raspberry desconectando directamente la alimentación puede resultar en daños a la tarjeta SD y puede ocasionar perdida de datos. La solución puede variar desde solo emplear cambios en el Hardware a solo realizar cambios en el Software, en este caso, se tomó la decisión de realizar solo cambios en el Software.

Se crea un Script para el apagado de la Raspberry, con un aviso en la salida de audio para el usuario, y mientras siga conectada a la alimentación, es decir, en modo "stand-by", se podrá volver a encender con el mismo actuador, en este caso un pulsante.

```python
import RPi.GPIO as GPIO
import time
import os
import pyttsx3

GPIO.setmode(GPIO.BCM)
GPIO.setup(3, GPIO.IN, pull_up_down = GPIO.PUD_UP)

while True:
    if(GPIO.input(3) == False):
        engine = pyttsx3.init()
        engine.setProperty('voice',"spanish")
        engine.setProperty('rate',150)
        GPIO.wait_for_edge(3,GPIO.FALLING)
        engine.say("Apagando dispositivo")
        engine.runAndWait()
        #time.sleep(3)
        os.system("sudo shutdown -h now")
        break
    time.sleep(1)
```

Figure 119 Script para el apagado y encendido de una Raspberry Pi 4 [Fuente: Los Autores]

De igual forma se crea un archivo ".desktop" para ejecutar este script cuando se vuelva a encender la Raspberry Pi 4

```
pi@raspberrypi: ~
Archivo  Editar  Pestañas  Ayuda
  GNU nano 3.2                    /etc/xdg/autostart/apagado_semaforos.desktop

[Desktop Entry]
Name=Apagado
Exec=/usr/bin/python3 /home/pi/Desktop/apagado.py
```

Figure 120 Contenido de "apagado_semaforos.desktop" [Fuente_ Los Autores]

De esta forma el sistema queda totalmente listo para su control con los dos pulsantes definidos en la programación, un pulsante para la toma de fotografías y detección posterior de la misma, y otro pulsante para el control del apagado y encendido de la Raspberry Pi 4, siempre y cuando esté conectada a una fuente de alimentación o batería.

CONCLUSIONES

El prototipo desarrollado funciona correctamente, el sistema de detección señalético de semáforos peatonales otorga una respuesta casi inmediata a pesar de ser procesada en un micro computador de recursos limitados, esto fue gracias a las pruebas realizadas en varios frameworks como Darknet, TensorFlow o TensorFlow Lite, concluyendo que el más óptimo para este tipo de plataformas es TensorFlow Lite debido a su comprimido modelo que aumento la velocidad de inferencia de nuestro modelo. Aun así, ejecutar detecciones en tiempo real en la Raspberry Pi 4 sigue siendo un reto si se plantea utilizar una arquitectura de base como la de YOLOv4-tiny, que a pesar de ser mucho más comprimida que la versión 4 normal de YOLO, requiere de mucho calculo computacional por parte del procesador, dejando así ver que la mejor forma de realizar las detecciones era que el usuario, mediante un pulsante, fotografiara lo que pudiera tener delante y que esta imagen pasara por la red para su consecuente predicción, que demora en responder aproximadamente un segundo o menos. La precisión media promedio (mAP) del modelo fue bastante aceptable y durante el entrenamiento no se observó un "overfitting", aunque la intersección sobre unión promedio (IoU) dejo un valor de aproximadamente 0.75 que se podría mejorar si se etiquetan de mejor forma las imágenes, siempre es recomendable encuadrar el objeto en el espacio justo. Los hiper parámetros de la red que se configuraron para este proyecto se los puede ir variando con la finalidad de observar alguna mejora, como, por ejemplo, se puede empezar a incrementar el redimensionamiento de la imagen de entrada a la red de 320x320 a 416x416 y observar si el tiempo de respuesta aumenta conjuntamente con una mejor precisión a la hora de calcular la predicción, en lo que respecta para el diseño y la construcción de este dispositivo se usó el software de inventor, debido a su facilidad y a su acople que tiene hoy en día este programa con las impresiones en 3D, además de ser un software bastante cómodo y profesional para realizar lo que son este tipo de proyectos, el material usado para este dispositivo es el ABS(Acrílico butadieno estireno) ya que nos otorga una mayor resistencia, gran durabilidad, un aguante de temperatura mayor a 100°C y porque este nos garantiza una mejor impresión en lo que vendría a ser resistencia, flexibilidad y durabilidad, en base a todas estas características se llegó a la conclusión de que el material de ABS es mucho mejor que el material de PLA (Ácido Poliláctico) ya que este viene a ser un material biodegradable y que no soporta temperaturas mayores de 60°C y tiene una menor durabilidad.

RECOMENDACIONES Y TRABAJOS FUTUROS

Para un correcto funcionamiento de los instaladores ya sean OpenCV, CUDA, YOLO es necesario tener un sistema operativo fuerte que soporte el entrenamiento de la red, la computadora que se usó consta con una GPU NVIDIA de 920M, sin embargo para un mejor entrenamiento y eficiencia es posible usar una de mayor potencia, además de tener instalados todos los controladores gráficos dentro de Ubuntu, ya que es el sistema operativo con el que se trabaja normalmente estos programas por la eficiencia que nos brinda.

Para una óptima función del detector de imágenes en tiempo real se tiene que saber manejar a que resolución se están trabajando las imágenes y de ser posible obtener una cámara que trabaje a esa misma resolución o pueda ser configurada de la misma, es decir si se entrena la red con imágenes de 5megapixeles, es preferible que la cámara que se use sea de la misma capacidad para que el sistema operativo aguante la ejecución de todo este proceso y la entrega de datos sea más eficiente.

Se recomienda que los valores de los "Anchors" a calcular sean mayores a 416 para una buena resolución además como dato interesante que se encuentra en el repositorio de AlexeyAB es que si se compila Darknet sin OpenCV y con el parámetro "mosaic=0", entonces la precisión puede ser de un 2% menos y en este caso se lo ha compilado con OpenCV y con el parámetro "mosaic=1" para evitar esa pérdida de precisión.

Una cosa importante que se debe aclarar es que el nombre de la carpeta "images_si" debe tener el mismo nombre que la carpeta "labels_si", de lo contrario, al ejecutar el entrenamiento puede ocurrir un error y dentro de la carpeta "images" se procede a colocar todas las fotografías obtenidas por la cámara, es decir, la base de datos.

Debido a que las personas no videntes se les dificulta al momento de cruzar las calles otro posible trabajo a futuro seria implementar a este sistema una detección de distancia, pero en este caso lo dificultaría debido a que se tendría que implementar un sistema grande y sensores de larga distancia lo cual lo vuelve a este proyecto más costoso; o incluso se puede agregar detección de números que permitirá saber cuándo se realiza el cambio de señal en los semáforos y estos facilitaría un poco la autonomía de las personas no videntes pero se tendrá que realizar un entrenamiento más robusto en la red.

REFERENCIAS

[1] C. N. p. i. d. capacidades, «Estadística de DIscapacidades,» [En línea]. Available: https://www.consejodiscapacidades.gob.ec/estadisticas-de-discapacidad. [Último acceso: 05 Nov 2020].

[2] OrCam, «OrCam MyEye 2,» [En línea]. Available: https://www.orcam.com/es/myeye2. [Último acceso: 05 Nov 2020].

[3] C. Vicario, Neurobiología de la VIsón, Cataluña-España: Universidad Politécnica de Cataluya, 1996.

[4] J. P. y. C. Aguilar, «Gafas especiales para detección de obstáculos con sistema de ubicación en caso de emergencia y ayuda de reconocimiento de billetes para personas con discapacidad visual,» Universidad Politécnica Salesiana, Ing. Electrónica CUE-Tesis Pregra, 2016. [En línea]. Available: http://dspace.ups.edu.ec/handle/123456789/12295..

[5] R.Hidalgo, Relaciones Interpersonales entre niños con discapacidad visual y sus compañeros videnteees en el contexto educativo regular, Cuenca-Azuay: Universidad de cuenca 2010.

[6] M.A.Núñez, La deficiencia Visual, Salamanca-España: Universidad de Salamanca. Instituto Universitario de integración de la counidad(INICIO).

[7] C. N. p. l. I. d. D. (CONADIS), «TOTAL DE PERSONAS CON DISCAPACIDAD REGISTRADAS EN EL REGISTRO NACIONAL DE DISCAPACIDAD,» Ministerio de Salud Pública, 06 Abril 2021. [En línea]. Available: https://www.consejodiscapacidades.gob.ec/estadisticas-de-discapacidad/. [Último acceso: 15 Enero 2021].

[8] J. e. al., Técnicas y algoritmos básicos de vision artificial, Universidad de la Rioja,2006, December,2016.

[9] C. G. G. y. H. H. Orbea, «Desarrollo de un Prototipo electrónico que guíe a personas invidentes en la recolección de objetos mediante audio y visión artificial,» Universidad Politécnica Salesiana, Tesis, 2019. [En línea]. Available: http://dspace.ups.edu.ec/bitstream/123456789/5081/1/UPS-CYT00109.pdf.

[10] T. e. al, Microsoft COCO: Common objects in context, 10.1007/978-3-319-10602-1_48: Vol3, 2014.

[11] «YOLO:Real-Time Object Detection,» [En línea]. Available: https://pjreddie.com/darknet/yolo/. [Último acceso: 05 Nov 2020].

[12] S. C. C. D. P. B. G. T. y. R. P. Stavroulakis, Rapid tracking of extrinsic projector parameters in fringe projection, Optics and Lasers in Engineering, vol. 114, 2018.

[13] Z. Z. X. M. L. L. G. W. X. H. L. L. J. P. G. L. y. Y. L. Deng, Fast object tracking on a many-core neural network chip, Frontiers in Neuroscience,n°841, 2018.

[14] W. Y. Z. H. L. L. C. S. y. T. K. S. Jiang, A classification-lock tracking strategy allwing a person-allowing robot to operate in a complicated indoor enviroments,, Sensor,vol 18,n°3903, 2018.

[15] NVIDIA, «NVIDIA,» NVIDIA, 2018. [En línea]. Available: https://www.nvidia.com/es-la/. [Último acceso: 15 Enero 2021].

[16] I. R. S. J. M. G. S. S. y. S. M. Hernandez-Fernandez, Using GPUs to accelerate computational diffusion MRI: From microstructure estimation to tractography and connectomes, NeuroImage, vol. 188, pp. 598-615, 2019.

[17] «Darknet:Open Source Neural Netwoks in,» [En línea]. Available: https://pjreddie.com/darknet/. [Último acceso: 05 Nov 2020].

[18] OpenCV, «Open Source Computer Vision,» [En línea]. Available: https://docs.opencv.org/master/d1/dfb/intro.html.

[19] NVIDIA, «NVIDIA DEVELOPER,» [En línea]. Available: https://developer.nvidia.com/cuda-zone.

[20] S. M. H. y. C. M. Benavides, Herramientas de simulación libres y abiertas para el diseño de unidades de procesamiento de potencia para sistemas fotovoltaicos, Tecnología en Marcha, vol. 28, nº 2, pp. 44-60, 2015.

[21] J. M. y. R. C. J. Merwin Moteiro, Sympl (v. 0.4.0) and climt (v. 0.15.3) - Towards a flexible framework for building model hierarchies in Python, Geoscientific Model Development, vol. 11, pp. 3781-3794, 2018.

[22] TensorFlow, «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/ .

[23] tzutalin, «GitHub,» [En línea]. Available: https://github.com/tzutalin/labelImg.

[24] TensorFlow, «Guia de TensorFlow Lite,» [En línea]. Available: https://www.tensorflow.org/lite/guide?hl=es-419.

[25] Darknet, «github,» YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet), [En línea]. Available: https://github.com/AlexeyAB/darknet.

[26] AlexeyAB, «YOLOv4.weights,» [En línea]. Available: https://drive.google.com/file/d/1cewMfusmPjYWbrnuJRuKhPMwRe_b9PaT/view.

[27] AlexeyAB, «yolov4-tiny.weights,» [En línea]. Available: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.weights.

[28] P. Clic, «Programador Clic,» 2020-2021. [En línea]. Available: https://programmerclick.com/article/11701269529/. [Último acceso: 10 Enero 2021].

[29] ProgrammerSought, «Install and use LabelImg on Ubuntu 18.04 (with link to install labelImg on windows and mac),» 2018-2021. [En línea]. Available: https://www.programmersought.com/article/86744971305/. [Último acceso: 10 Enero 2021].

[30] A. Gandhi, «Nanonets,» [En línea]. Available: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ .

[31] AlexeyAB, «GitHub, Darknet,» [En línea]. Available: https://github.com/AlexeyAB/darknet/archive/master.zip.

[32] NVIDIA, «CUDA TOOLKIT DOCUMENTATION,» [En línea]. Available: https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html.

[33] NVIDIA, «CUDA Toolkit 10.1 update 2 Archive,» [En línea]. Available: https://developer.nvidia.com/cuda-10.1-download-archive-update2?target_os=Windows&target_arch=x86_64&target_version=10&target_type=exelocal.

[34] NVIDIA Developer Zone, «CUDA Toolkit documentation,» [En línea]. Available: https://docs.nvidia.com/cuda/archive/10.1/cuda-installation-guide-microsoft-windows/index.html.

[35] TensorFlow, «Compila con el código fuente,» [En línea]. Available: https://www.tensorflow.org/install/source#gpu.

[36] NVIDIA Developer, «cuDNN Archive,» [En línea]. Available: https://developer.nvidia.com/rdp/cudnn-archive.

[37] CMake, «Binary distributions,» [En línea]. Available: https://github.com/Kitware/CMake/releases/download/v3.21.1/cmake-3.21.1-windows-x86_64.msi.

[38] OpenCV, «Releases,» [En línea]. Available: https://opencv.org/releases/.

[39] OpenCV, «Building OpenCV from source,» [En línea]. Available: https://docs.opencv.org/4.5.3/d5/de5/tutorial_py_setup_in_windows.html.

[40] AlexeyAB, «CUDA status Error: file,» [En línea]. Available: https://github.com/AlexeyAB/darknet/issues/2691.

[41] AlexeyAB, «Github, stereolabs, zed-yolo,» [En línea]. Available: https://github.com/stereolabs/zed-yolo/issues/10.

[42] Darknet, «Installing Darknet,» [En línea]. Available: https://pjreddie.com/darknet/install/.

[43] G. G. J. A. FARIAS RIVERA FATIMA KATIUSKA, Implementación de un sistema para el conteo volumétrico de objetos mediante Redes, Machala: Universida Técnica de Machala, 11 de Febrero de 2019.

[44] AlexeyAB, «CFG Parameters in the [net] section,» [En línea]. Available: https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-%5Bnet%5D-section.

[45] ProgrammerClick, «Comprensión de los parámetros de salida de entrenamiento de yolo,» [En línea]. Available: https://programmerclick.com/article/47772118879/.

[46] K. Fromm, «Epochs number #559,» [En línea]. Available: https://github.com/pjreddie/darknet/issues/1900.

[47] AlexeyAB, «yolov4-tiny.conv.29,» [En línea]. Available: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29.

[48] nateshmbhat, «pyttsx,» [En línea]. Available: https://github.com/nateshmbhat/pyttsx3.

[49] PyPI, «pynput 1.7.3,» [En línea]. Available: https://pypi.org/project/pynput/.

[50] A. Rosebrock, «Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster,» [En línea]. Available: https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/?fbclid=IwAR3NrXuPNVW_homgkIYvi9X4TedX0kWK5HRQ8u3t8jdz5fRfHs3UgzB_eAM.

[51] lhelontra, «tensorflow-on-arm,» [En línea]. Available: https://github.com/lhelontra/tensorflow-on-arm/releases.

[52] PINTO0309, «Tensorflow-bin,» [En línea]. Available: https://github.com/PINTO0309/Tensorflow-bin/?fbclid=IwAR3NrXuPNVW_homgkIYvi9X4TedX0kWK5HRQ8u3t8jdz5fRfHs3UgzB_eAM#usage.

[53] hunglc007, «tensorflow-yolov4-tflite,» [En línea]. Available: https://github.com/hunglc007/tensorflow-yolov4-tflite.

[54] TensorFlow, «TensorFlow Lite,» [En línea]. Available: https://www.tensorflow.org/lite/guide?hl=es-419.

[55] Tensorflow, «Python quickstart, About the Tensorflow Lite runtime package,» [En línea]. Available: https://www.tensorflow.org/lite/guide/python?fbclid=IwAR0-5ugIEWmdymsfhwXxPZSYclB7zX_ykctPsSenRCfcUsV0pR4aRFPFzdA.

[56] ARDUINO, «Arduino Nano,» Arduino, 2021. [En línea]. Available: https://store-usa.arduino.cc/products/arduino-nano?selectedStore=us.

[57] P. Components, «Mundo de Componentes,» Polígono industrial Las Salinas, [En línea]. Available: https://www.pccomponentes.com/caracteristicas-raspberry-pi-4?__cf_chl_captcha_tk__=pmd_YHBJ.gus8mRdY0Ffdtj1xHQ21iojeOE1CqaUQf7NoOM-1635189305-0-gqNtZGzNAxCjcnBszQgR.

[58] SuperRobotica, «S320120 MAXSONAR EZ1 SENSOR DE DISTANCIAS POR ULTRASONIDOS,» [En línea]. Available: http://www.superrobotica.com/download/S320120/Manula%20S320120%20MaxSonar%20EZ1.pdf. [Último acceso: 25 Enero 2021].

[59] MaxBotix, «LV-MaxSonar-EZ, Performance Sonar Range Finder MB1000,MB1020,MB1030,MB1040,» Datasheet, USA, 2005.

[60] Ruiling, «Amazon,» [En línea]. Available: https://www.amazon.com/RuiLing-1034-vibraci%C3%B3n-electrodom%C3%A9sticos-0-134/dp/B07TBCCZS9?pd_rd_w=E7GGI&pf_rd_p=c4085173-f058-4953-ae5c-65d9590cc2d0&pf_rd_r=EBDB07S13YZ63P3XNB2P&pd_rd_r=1cd9b622-187a-4916-9f59-17c32489b446&pd_rd_wg=GUPuf&pd_rd_i=B07TB. [Último acceso: 16 Enero 2021].

[61] Y. D. y. H. S. J. A. Khan, New dark area sensitive tone mapping, Sensors, vol. 11, nº 3776 pp. 426-791, 2018.