Índice general

v

Índice general

Índice de figuras

VII

Índice de figuras

Índice de tablas

IX

Índice de tablas

1.1. Fecha de lanzamiento de distintos modelos YOLO. 3

2.1. Clases detectadas como intruso por el módulo. 8
2.2. Software requerido para el desarrollo y la ejecución del módulo. . . 11

3.1. Clases auxiliares para detección indirecta. 18

Capítulo 3

Diseño e implementación

En este capítulo se detallan los aspectos técnicos del desarrollo del trabajo. Incluye las consideraciones tomadas en cuenta durante el desarrollo, el detalle de las modificaciones realizadas al modelo de partida, así como el detalle del papel que juega el módulo en el sistema. De igual manera, se hace una descripción cronológica del desarrollo del módulo.

3.1. Consideraciones generales

El módulo de inteligencia artificial propuesto es una implementación del modelo YOLOv3, especialmente modificada para cumplir con las necesidades específicas del cliente, así como una serie de requisitos funcionales. Es por esto que fue necesario hacer modificaciones a los archivos originales de la biblioteca.

Como se puede observar en el capítulo 1, no todos los requerimientos funcionales propuestos para el inicio del trabajo. Recordando que esta versión del trabajo representa un mínimo producto viable, a continuación se presenta una lista de todos los requerimientos funcionales que fueron cubiertos:

- el módulo se ejecuta en un computador de uso de hogar.
- el módulo es capaz de reconocer n clases como intruso.
- se reporta al usuario de manera visual.
- se reporta al usuario a través de un archivo de fácil análisis.
- resulta muy sencillo iniciar la ejecución del módulo a través de su interfaz grafica.

De igual manera, el módulo está construido de manera que realizar modificaciones, o bien, expandir su funcionalidad en un futuro resulte muy sencillo. Es de vital importancia recalcar que esto se hizo teniendo en cuenta que se trata de un módulo que, a fin de salir de este estatus de mínimo producto viable, deberá sufrir un proceso de optimizacion muy fuerte. Algunos ejemplos de puntos en los que se puede aplicar estos trabajos son:

- el módulo está diseñado para filtrar las etiquetas que no hacen parte de las clases dadas en la definición de intruso. Esto implica que, a pesar de que no se reporte en ninguna de las dos metodologías, el módulo aún reconoce otras clases. En otras palabras, si se puede observar un modelo en el *frame*, este será satisfactoriamente detectado y son los métodos encargados del reporte los que no lo entregarán como un resultado positivo.

13

Capítulo 3

Diseño e implementación

En este capítulo se detallan los aspectos técnicos del desarrollo del trabajo. Incluye las consideraciones tomadas en cuenta durante el desarrollo, el detalle de las modificaciones realizadas al modelo de partida, así como el detalle del papel que juega el módulo en el sistema. De igual manera, se hace una descripción cronológica del desarrollo del módulo.

3.1. Consideraciones generales

El módulo de inteligencia artificial propuesto es una implementación del modelo YOLOv3, especialmente modificada para cumplir con las necesidades específicas del cliente, así como una serie de requisitos funcionales. Es por esto que fue necesario hacer modificaciones a los archivos originales de la biblioteca.

Como se puede observar en el capítulo 1, no todos los requerimientos funcionales propuestos para el inicio del trabajo pudieron ser cubiertos. Recordando que esta versión del trabajo representa un mínimo producto viable, a continuación se presenta una lista de todos los requerimientos funcionales que fueron cubiertos:

- El módulo se ejecuta en un computador de uso de hogar.
- El módulo es capaz de reconocer n clases como intruso.
- Se reporta al usuario de manera visual.
- Se reporta al usuario a través de un archivo de fácil análisis.
- Resulta muy sencillo iniciar la ejecución del módulo a través de su interfaz grafica.

De igual manera, el módulo está construido de manera que realizar modificaciones, o bien, expandir su funcionalidad en un futuro resulte muy sencillo. Es de vital importancia recalcar que esto se hizo teniendo en cuenta que se trata de un módulo que, a fin de salir de este estatus de mínimo producto viable, deberá sufrir un proceso de optimizacion muy fuerte. Algunos ejemplos de puntos en los que se pueden aplicar estos trabajos son:

- El módulo está diseñado para filtrar las etiquetas que no hacen parte de las clases dadas en la definición de intruso. Esto implica que, a pesar de que no se reporte en ninguna de las dos metodologías, el módulo aún reconoce otras clases. En otras palabras, si se puede observar un modelo en el *frame*, este será satisfactoriamente detectado y son los métodos encargados del reporte los que no lo entregarán como un resultado positivo.

- la precisión del modelo debe ser mejorada: dado que ocasionalmente se hacen detecciones de objetos que no corresponden con lo visto en el *frame*. Se detectó que, en particular, el módulo es especialmente susceptible a generar falsos positivos.
- los cuadros se devuelven en espacio de color bgr. Esto, en combinación con que el color de los rectángulos no se puede cambiar desde la interfaz grafica.
- el color de los rectángulos es igual para todas las clases de objeto.
- se debe mejorar significativamente el manejo que da el módulo de las excepciones que se generen, tanto a nivel de manejo de problemas originados en el código mismo, como recuperación cuando se generen cortes en ma recepcion del *streaming* de video.
- a pesar de que la interferir grafica se realizó pensando en la facilidad de uso para el usuario, su comportamiento podría resultar extraño dado que el diseño de experiencia de usuario (UX) quedó fuera del alcance de los trabajos durante su etapa de concepción.

Finalmente, resulta de especial interés tener en mente también las limitaciones del modelo:

- los modelos YOLO se ven limitados cuando se les pide detectar objetos que se encuentran muy cerca de una fuente de luz muy brillante, especialmente el sol, se puede generar dificultades para detectar los objetos. Esto es parcialmente contrarrestando añadiendo las clases adicionales detalladas en la tabla n.
- los objetos deben encontrarse a una distancia prudencial del dron. A pesar de que mas pruebas de vuelo no se han enfocado en el cálculo de este valor, sí se hace evidente que al estar el dron demasiado lejos de ellas, no se hace una detección oportuna.
- el modelo no cuenta con funciones de *auto-healing*, que deben ser tenidas en cuenta a fin de lograr mantener la integridad del modelo si sus pesos son actualizados constantemente por el cliente a fin de mejorar la predicción.

3.2. Arquitectura del módulo

Se propone como mínimo producto viable de este módulo, un código altamente modular. Esto significa que la manera en la que está desarrollado (a excepción del método principal, en el archivo Utils), está planteada en pequeños métodos que podrán ser modificados y mantenidos con facilidad. Esto supone, sin embargo, que sea de vital importancia entender la arquitectura del módulo a fin de poder realizar las modificaciones pertinentes en el punto adecuado del código. Así como se hizo en el literal anterior con el flujo de información entre el Sistema y el Módulo, a continuación, se describe el flujo de la información en el interior del módulo.

El módulo está compuesto por cuatro capas funcionales:

1. funciones originales de YOLO: encargadas de la funcionalidad base de detección de los objetos en la totalidad de las clases que componen el *dataset*

- La precisión del modelo debe ser mejorada: dado que ocasionalmente se hacen detecciones de objetos que no corresponden con lo visto en el *frame*. Se detectó que, en particular, el módulo es especialmente susceptible a generar falsos positivos.
- Los cuadros se devuelven en espacio de color bgr. Esto, en combinación con que el color de los rectángulos no se puede cambiar desde la interfaz grafica.
- El color de los rectángulos es igual para todas las clases de objeto.
- Se debe mejorar significativamente el manejo que da el módulo de las excepciones que se generen, tanto a nivel de manejo de problemas originados en el código mismo, como recuperación cuando se generen cortes en la recepcion del *streaming* de video.
- A pesar de que la interfaz grafica se realizó pensando en la facilidad de uso para el usuario, su comportamiento podría resultar extraño dado que el diseño de experiencia de usuario (UX) quedó fuera del alcance de los trabajos durante su etapa de concepción.

Finalmente, resulta de especial interés tener en mente también las limitaciones del modelo:

- Los modelos YOLO se ven limitados cuando se les pide detectar objetos que se encuentran muy cerca de una fuente de luz muy brillante, especialmente el sol, se pueden generar dificultades para detectar los objetos. Esto es parcialmente contrarrestado añadiendo las clases adicionales detalladas en la tabla 3.1.
- Los objetos deben encontrarse a una distancia prudencial del dron. A pesar de que las pruebas de vuelo no se han enfocado en el cálculo de este valor, sí se hace evidente que al estar el dron demasiado lejos de ellas, no se hace una detección oportuna.
- El modelo no cuenta con funciones de *auto-healing*, que deberían ser tenidas en cuenta a fin de lograr mantener la integridad del modelo si sus pesos son actualizados constantemente por el cliente a fin de mejorar la predicción.

3.2. Arquitectura del módulo

Se propone como mínimo producto viable de este módulo, un código altamente modular. Esto significa que la manera en la que está desarrollado (a excepción del método principal, en el archivo Utils), está planteada en pequeños métodos que podrán ser modificados y mantenidos con facilidad. Esto supone, sin embargo, que sea de vital importancia entender la arquitectura del módulo a fin de poder realizar las modificaciones pertinentes en el punto adecuado del código. Así como se hizo en el numeral anterior (consideraciones generales) con el flujo de información entre el sistema y el módulo, a continuación, se describe el flujo de la información en el interior del módulo.

El módulo está compuesto por cuatro capas funcionales:

1. Funciones originales de YOLO: encargadas de la funcionalidad base de detección de los objetos en la totalidad de las clases que componen el *dataset*

COCO. Incluye funciones como nms (encargada del *non-max supression*) y la función IoU.

2. funciones híbridas: normalmente funciones qué coordinan el llamado de las demás funciones. A pesar de que ofrecen una funcionalidad muy similar a las originales de YOLO, funcionan en un orden diferente, de forma que se puedan obtener los datos requeridos en el momento preciso para las necesidades del cliente.
3. funciones propias: Que permiten agregar al módulo la funcionalidad requerida por el cliente. Ejemplos de estas funciones incluyen la generación de archivos en formato CSV y la carga de etiquetas válidas.
4. interfaz gráfica: Que permite al usuario interactuar con el módulo para iniciar su ejecución fácilmente.

El diagrama de boques correspondiente a estas cuatro capas es el siguiente:

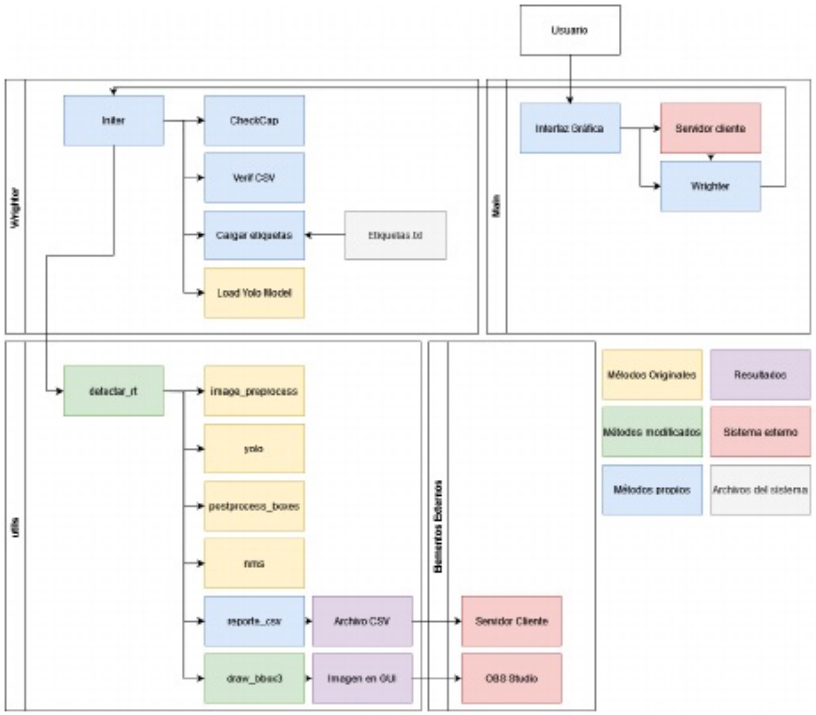

FIGURA 3.1. Diagrama de bloques del módulo.

3.2. Arquitectura del módulo 15

COCO. Incluye funciones como nms (encargada del *non-max supression*) y la función IoU.

2. Funciones híbridas: normalmente funciones que coordinan el llamado de las demás funciones. A pesar de que ofrecen una funcionalidad muy similar a las originales de YOLO, funcionan en un orden diferente, de forma que se puedan obtener los datos requeridos en el momento preciso para las necesidades del cliente.
3. Funciones propias: que permiten agregar al módulo la funcionalidad requerida por el cliente. Ejemplos de estas funciones incluyen la generación de archivos en formato CSV y la carga de etiquetas válidas.
4. Interfaz gráfica: que permite al usuario interactuar con el módulo para iniciar su ejecución fácilmente.

El diagrama de bloques correspondiente a estas cuatro capas es el siguiente:

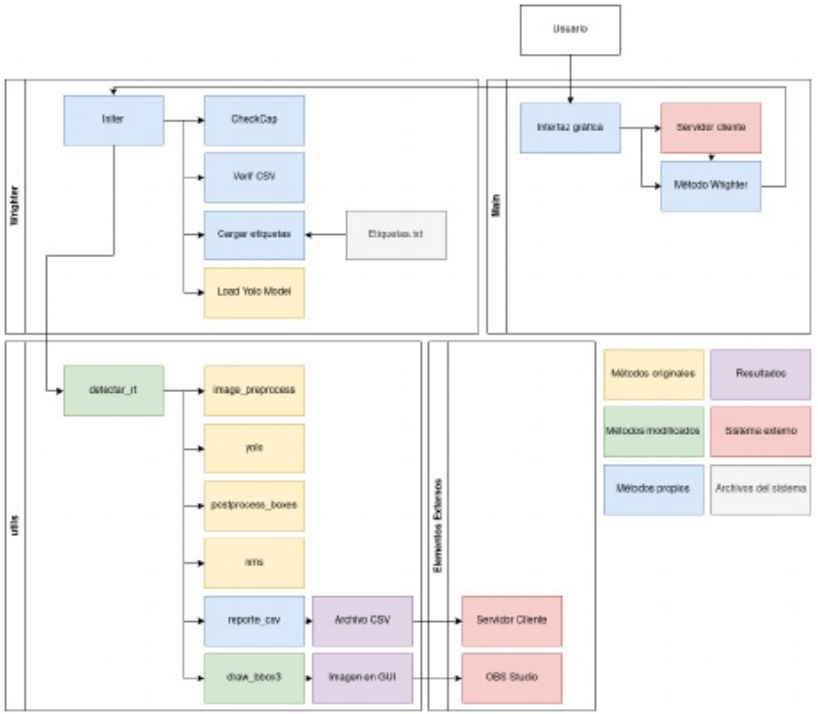

FIGURA 3.1. Diagrama de bloques del módulo.

3.3. Esquema detallado del módulo en el interior del sistema

Este módulo es una parte de un trabajo más grande desarrollado directamente por el cliente. Es decir, deberá estar en capacidad de comunicarse con otras partes del sistema, de forma que estos otros módulos puedan tomar decisiones y recibir órdenes de controladores humanos. A pesar de que se desconoce el funcionamiento de la totalidad del sistema.

Para lograr esto, durante la fase de concepción del trabajo se estipuló que el módulo deberá ser capaz de conectarse a los servidores del cliente, en los que se aloja la transmisión de vídeo. Esta conexión entre ambos equipos se logra cuando el usuario indica al módulo, a través de la interfaz gráfica (visible en la imagen). Una vez se presiona el botón "Iniciar ejecución". Esta interfaz está pensada para ser tan sencilla como sea posible, permitiendo que el usuario inicie la ejecución del módulo de la manera más sencilla posible.

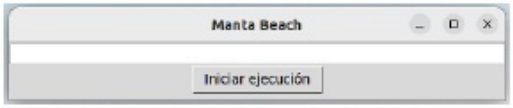

FIGURA 3.2. Interfaz gráfica del módulo.

Por otro lado, el módulo deberá reportar al resto del sistema si se detectó a algún intruso. Para esto, se definieron dos metodologías, que serán tocadas más en detalle más adelante en este capítulo:

1. reporte visual
2. reporte en formato CSV

Dada esta descripción, desde afuera, el módulo tendrá el siguiente aspecto:

Cabe mencionar que, dada la naturaleza confidencial de este módulo, así como de la totalidad del sistema, se buscó que el módulo requiriera la menor cantidad de información para funcionar como fuese posible. Esto también permite que los parámetros dados del vídeo y la cantidad de transmisiones que se procesan a la vez se encuentren bajo total control del cliente.

De igual manera es de vital importancia tener en cuenta que la totalidad del módulo se desarrolló en total desconocimiento del funcionamiento del resto del sistema. Es por esto que para iniciar su ejecución se debe ejecutar el código de manera manual. El sistema no puede detectar cuándo se ha iniciado una transmisión, y por lo tanto, se requiere una persona encargada de dar inicio cuando sea necesario.

Finalmente, este módulo está diseñado como un mínimo producto viable que permita a las personas tomar decisiones informadas, con soporte en vídeo. Esta versión no es de ninguna manera un sistema autónomo, sino que constituye un sistema de apoyo a la decisión.

16 Capítulo 3. Diseño e implementación

3.3. Esquema detallado del módulo en el interior del sistema

Este módulo es una parte de un trabajo más grande desarrollado directamente por el cliente. Es decir, deberá estar en capacidad de comunicarse con otras partes del sistema, de forma que estos otros módulos puedan tomar decisiones y recibir órdenes de controladores humanos. A pesar de que se desconoce el funcionamiento de la totalidad del sistema.

Para lograr esto, durante la fase de concepción del trabajo se estipuló que el módulo deberá ser capaz de conectarse a los servidores del cliente, en los que se aloja la transmisión de vídeo. Esta conexión entre ambos equipos se logra cuando el usuario indica al módulo, a través de la interfaz gráfica (visible en la imagen). Una vez se presiona el botón "Iniciar ejecución". Esta interfaz está pensada para ser tan sencilla como sea posible, permitiendo que el usuario inicie la ejecución del módulo de la manera más sencilla posible.

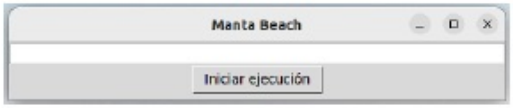

FIGURA 3.2. Interfaz gráfica del módulo.

Por otro lado, el módulo deberá reportar al resto del sistema si se detectó a algún intruso. Para esto, se definieron dos metodologías, que se detallarán más adelante en este capítulo:

1. Reporte visual
2. Reporte en formato CSV

Dada esta descripción, desde afuera, el módulo tendrá el siguiente aspecto:

Cabe mencionar que, dada la naturaleza confidencial de este módulo, así como de la totalidad del sistema, se buscó que el módulo requiriera la menor cantidad de información para funcionar como fuese posible. Esto también permite que los parámetros dados del vídeo y la cantidad de transmisiones que se procesan a la vez se encuentren bajo total control del cliente.

De igual manera es de vital importancia tener en cuenta que la totalidad del módulo se desarrolló sin conocer el funcionamiento del resto del sistema. Es por esto que para iniciar su ejecución se debe ejecutar el código de manera manual. El sistema no puede detectar cuándo se ha iniciado una transmisión, y por lo tanto, se requiere de una persona encargada de dar inicio cuando sea necesario.

Finalmente, este módulo está diseñado como un mínimo producto viable que permita a las personas tomar decisiones informadas, con soporte en vídeo. Esta versión no se considera un sistema autónomo, sino que constituye un sistema de apoyo a la decisión.

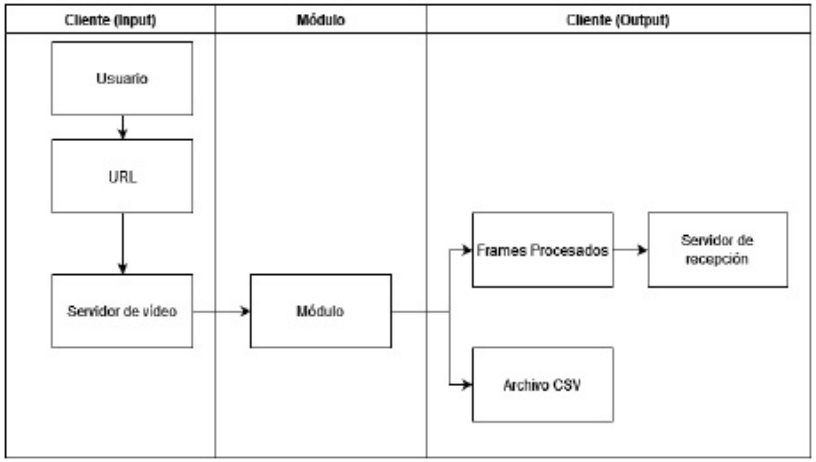

FIGURA 3.3. Diagrama de bloques del módulo en el interior del sistema.

3.4. Esquema detallado de las modificaciones hechas al modelo

Dado que se trata de una implementación personalizada, existen varias diferencias entre el módulo desarrollado y la implementación original del modelo YOLOv3. Dicho esto, las modificaciones realizadas generales al modelo, en orden cronológico son las siguientes:

- se preparó el código para que se ejecute desde un *streaming* de vídeo, en vez de un archivo local en el equipo. Esto implica que se debe detener la ejecución antes de pasar imágenes al método de detección si no es posible establecer conexión con el servidor.
- se filtraron las etiquetas a fin de lograr que sólo sean detectadas las clases definidas en la definición de intruso.
- se crearon los métodos encargados de generar una respuesta que se pasará al resto del módulo. Particularmente, se le permitió al programa generar dos tipos de respuesta.
- se estableció un método que permite descartar un número determinado de *frames*, de forma que se permita aliviar la carga computacional sobre el equipo.
- durante el desarrollo del módulo se estableció que el hacer que el módulo sea capaz de procesar varios *streamings* de vídeo de manera paralela está fuera del alcance de las temáticas cubiertas en el postgrado, a fin de cubrir satisfactoriamente este requisito, se informó al cliente que sería posible este análisis en caso de que se reciba el *streaming* con varias cámaras combinadas

3.4. Esquema detallado de las modificaciones hechas al modelo 17

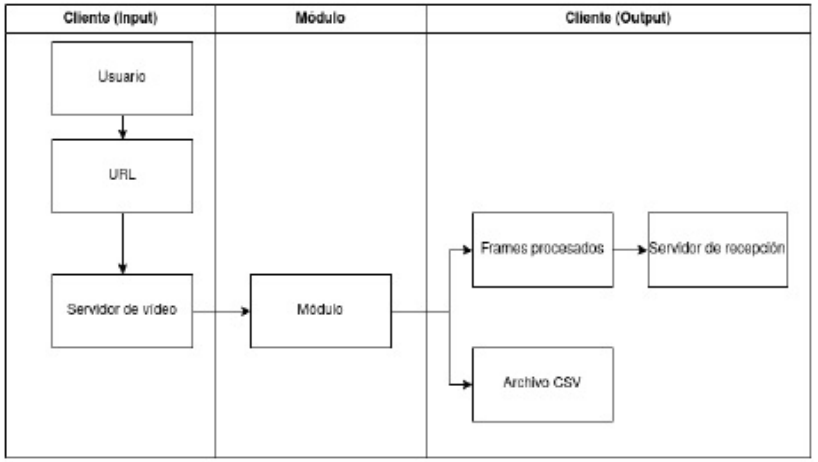

FIGURA 3.3. Diagrama de bloques del módulo en el interior del sistema.

3.4. Esquema detallado de las modificaciones hechas al modelo

Dado que se trata de una implementación personalizada, existen varias diferencias entre el módulo desarrollado y la implementación original del modelo YOLOv3. Dicho esto, las modificaciones realizadas generales al modelo, en orden cronológico son las siguientes:

- Se preparó el código para que se ejecute desde un *streaming* de vídeo, en vez de un archivo local en el equipo. Esto implica que se debe detener la ejecución antes de pasar imágenes al método de detección si no es posible establecer conexión con el servidor.
- Se filtraron las etiquetas a fin de lograr que solo sean detectadas las clases definidas en la definición de intruso.
- Se crearon los métodos encargados de generar una respuesta que se pasará al resto del módulo. Particularmente, se le permitió al programa generar dos tipos de respuesta.
- Se estableció un método que permite descartar un número determinado de *frames*, de forma que se permita aliviar la carga computacional sobre el equipo.
- Durante el desarrollo del módulo se estableció que el hacer que el módulo sea capaz de procesar varios *streamings* de vídeo de manera paralela está fuera del alcance de las temáticas cubiertas en el postgrado, a fin de cubrir satisfactoriamente este requisito, se informó al cliente que sería posible este análisis en caso de que se reciba el *streaming* con varias cámaras combinadas

en un único cuadro. Esto, por ser una decisión relacionada con la forma en la que el cliente maneja su *streaming*, no afectó el desarrollo del trabajo.

Cabe resaltar que cada una de estas modificaciones implicó una serie de tareas más pequeñas, a las que Manta Beach, como cliente, hizo un seguimiento cercano.

3.5. Ajustes para cumplir con el rendimiento requerido

Dadas las limitantes del equipo en el que se realizó el desarrollo del módulo (especialmente teniendo en cuenta que este desarrollo se hizo utilizando máquinas virtuales), fue necesario llevar a cabo un análisis detallado que, en teniendo en mente las necesidades del cliente, permitiera reducir la carga computacional sobre la máquina. De esta manera se intentó una serie de ajustes al modelo de forma que se lograra mantener los requisitos dados en el proceso de establecimiento de los trabajos:

- eliminación de un número determinado de *frames*. Si bien, este es un número fácil de establecer, y que el sistema recibirá como parámetro, se recomienda utilizar un valor de N=10, de forma que, para un fps de 30, se analicen 3 cuadros por segundo. Cabe resaltar que 30 será el número máximo para cumplir con el requisito de hacer un análisis en un tiempo no mayor a 1 segundo, mientras que con N=120, se cumplirá el requisito de hacer un análisis en un tiempo no mayor a 4 segundos.
- reducción de la calidad del *streaming* de vídeo a 200. A pesar de que esta reducción afecta muy notoriamente la calidad del vídeo, el módulo es capaz de detectar objetos. Se pierden, sin embargo, las propiedades probatorias del trabajo. En la imagen se puede ver un ejemplo del vídeo con calidad reducida. Como se mencionó en secciones anteriores este es un parámetro qué el módulo ya recibe en el *streaming* de vídeo, y cuyo control recae únicamente en el cliente.

Finalmente, la solución escogida fue descartar uno de cada N *frames*, de forma que no se procese la totalidad de ellos, y se requiera una mejor cantidad de operaciones. Cabe mencionar que, en este tipo de modelos, es común que conforme se aumente el rendimiento, también se reducirá la precisión del modelo. Es por esto que, para contrarrestar estos efectos, se decidió incluir algunas clases que permitieran hacer una detección indirecta. Estas clases corresponden a objetos que las personas cargan con frecuencia. Estas clases adicionales son:

TABLA 3.1. Clases auxiliares para detección indirecta.

Clases auxiliares	
Camión	Bicicleta
Automóvil	Bus
Motocicleta	Sombrilla
Mochila	Maletín

Dada la naturaleza del trabajo, cabe aclarar que, en este punto, el módulo no podrá identificar discernir que al detectar simultáneamente la clase mochila y la clase persona en un espacio muy cercano, podrá tratarse de una única detección. Estas serán registradas y reportadas por el módulo como dos detecciones separadas.

18 Capítulo 3. Diseño e implementación

en un único cuadro. Esto, por ser una decisión relacionada con la forma en la que el cliente maneja su *streaming*, no afectó el desarrollo del trabajo.

Cabe resaltar que cada una de estas modificaciones implicó una serie de tareas más pequeñas, a las que Manta Beach, como cliente, hizo un seguimiento cercano.

3.5. Ajustes para cumplir con el rendimiento requerido

Dadas las limitantes del equipo en el que se realizó el desarrollo del módulo (especialmente teniendo en cuenta que este desarrollo se hizo utilizando máquinas virtuales), fue necesario llevar a cabo un análisis detallado que, en teniendo en mente las necesidades del cliente, permitiera reducir la carga computacional sobre la máquina. De esta manera se intentó una serie de ajustes al modelo de forma que se lograra mantener los requisitos dados en el proceso de establecimiento de los trabajos:

- Eliminación de un número determinado de *frames*. Si bien, este es un número fácil de establecer, y que el sistema recibirá como parámetro, se recomienda utilizar un valor de N=10, de forma que, para un fps de 30, se analicen 3 cuadros por segundo. Cabe resaltar que 30 será el número máximo para cumplir con el requisito de hacer un análisis en un tiempo no mayor a 1 segundo, mientras que con N=120, se cumplirá el requisito de hacer un análisis en un tiempo no mayor a 4 segundos.
- Reducción de la calidad del *streaming* de vídeo a 200. A pesar de que esta reducción afecta muy notoriamente la calidad del vídeo, el módulo es capaz de detectar objetos. Se pierden, sin embargo, las propiedades probatorias del trabajo. En la imagen se puede ver un ejemplo del vídeo con calidad reducida. Como se mencionó en secciones anteriores este es un parámetro qué el módulo ya recibe en el *streaming* de vídeo, y cuyo control recae únicamente en el cliente.

Finalmente, la solución escogida fue descartar uno de cada N *frames*, de forma que no se procese la totalidad de ellos, y se requiera una mejor cantidad de operaciones. Cabe mencionar que, en este tipo de modelos, es común que conforme se aumente el rendimiento, también se reducirá la precisión del modelo. Es por esto que, para contrarrestar estos efectos, se decidió incluir algunas clases que permitieran hacer una detección indirecta. Estas clases corresponden a objetos que las personas cargan con frecuencia. Estas clases adicionales son:

TABLA 3.1. Clases auxiliares para detección indirecta.

Clases auxiliares	
Camión	Bicicleta
Automóvil	Bus
Motocicleta	Sombrilla
Mochila	Maletín

Por la naturaleza del trabajo, cabe aclarar, en este punto, que el módulo no podrá discernir que al detectar simultáneamente la clase mochila y la clase persona en un espacio muy cercano, podrá tratarse de una única detección. Estas serán registradas y reportadas por el módulo como dos detecciones separadas.

Esto, sin embargo, es preferible a eliminar detecciones, teniendo en mente que se prefiere que el módulo pueda detectar a tantas personas como sea posible, dada la sensibilidad de las consecuencias en caso de una falla del modelo.

3.6. Entrega de resultados obtenidos al resto del software y retransmisión del vídeo procesado

Una vez se ha analizado el *frame*, el módulo deberá estar en capacidad de devolver resultados al resto del sistema, de forma que tanto las personas encargadas, como el sistema mismo estén en capacidad de tomar decisiones rápidas, e informadas de cómo proceder. Dados estos dos actores a los que se les debe hacer llegar la información, en conjunto con el cliente se decidió utilizar dos métodos diferentes a través de los que el módulo dará su respuesta para cada uno de los *frames*:

- entrega visual: Imagen procesada con sus cajas identificando los objetos detectados. Estos *frames* ya procesados son retransmitidos al servidor original para su visualización en la plataforma designada por el cliente. Es importante tener en cuenta que, en esta versión del módulo, esta retransmisión no se hace directamente a través del módulo propuesto, sino que se requerirá el uso de software externo, que en esta ocasión es OBS Studio.
- entrega para análisis de datos: Archivo en formato CSV con la información de los objetos detectados. De igual manera, el sistema estará en capacidad de reportar el número del dron desde el que se hizo la detección, a pesar de que es una característica que no se ha probado, teniendo en cuenta que aún no se ha hecho vuelos con varios drones de manera simultánea. A diferencia de la imagen preprocesada, este archivo no es retransmitido. Se guarda por defecto en la carpeta de documentos de la maquina en la que se está ejecutando. Dado que esta versión corresponde a un mínimo producto viable, este archivo no se retransmite. Se guarda, en cambio, en la carpeta de documentos dr la máquina en la que se ejecuta el módulo.

Es de vital importancia tener en cuenta, sin embargo, que como se mencionó en el capítulo 1, no se espera que haya una persona encargada constantemente de la revisión de las cámaras, y en consecuencia, es posible deshabilitar la retransmisión del cuadro preprocesado. Por otro lado, y buscando que el sistema sea tan autónomo como resulte posible, no se podrá apagar el reporte a través de archivos JSON.

Por otro lado, los archivos JSON deberán contener los siguientes datos:

- fecha y hora de detección.
- coordenadas de detección dentro del *frame*.
- número de dron en el que ocurre la detección.

Los datos serán entregados en el orden en el que han sido mencionados. Esto es de vital importancia de forma que el resto del sistema sea capaz de interpretar correctamente los datos entregados. Por otro lado, se debe recordar que el sistema desconoce la ubicación del dron en su recorrido de vuelo. Es por esto que el módulo no podrá reportar la ubicación en la que detectó el objeto. Esta deberá ser

3.6. Entrega de resultados obtenidos al resto del software y retransmisión del vídeo procesado 19

Esto, sin embargo, es preferible a eliminar detecciones, teniendo en mente que se prefiere que el módulo pueda detectar a tantas personas como sea posible, dada la sensibilidad de las consecuencias en caso de una falla del modelo.

3.6. Entrega de resultados obtenidos al resto del software y retransmisión del vídeo procesado

Una vez que se ha analizado el *frame*, el módulo deberá ser capaz de devolver resultados al resto del sistema, de forma que tanto las personas encargadas, como el sistema mismo sean capaces de tomar decisiones rápidas, e informar cómo proceder. Dados estos dos actores a los que se les debe hacer llegar la información, en conjunto con el cliente se decidieron utilizar dos métodos diferentes a través de los que el módulo dará su respuesta para cada uno de los *frames*:

- Entrega visual: imagen procesada con sus cajas identificando los objetos detectados. Estos *frames* ya procesados son retransmitidos al servidor original para su visualización en la plataforma designada por el cliente. Es importante tener en cuenta que, en esta versión del módulo, esta retransmisión no se hace directamente a través del módulo propuesto, sino que se requerirá el uso de software externo, como el OBS Studio.
- Entrega para análisis de datos: archivo en formato CSV con la información de los objetos detectados. De igual manera, el sistema será capaz de reportar el número del dron desde el que se hizo la detección, a pesar de que es una característica que no se ha probado, teniendo en cuenta que aún no se han hecho vuelos con varios drones de manera simultánea. A diferencia de la imagen preprocesada, este archivo no es retransmitido. Se guarda por defecto en la carpeta de documentos de la máquina en la que se está ejecutando. Visto que esta versión corresponde a un mínimo producto viable, este archivo no se retransmite. Se guarda, en cambio, en la carpeta de documentos de la máquina en la que se ejecuta el módulo.

Es muy importante tener en mente que se espera que no haya personas supervisando constantemente las cámaras. Por este motivo, es posible deshabilitar la retransmisión del *frame* procesado. Por otro lado, y buscando que el sistema sea tan autónomo como resulte posible, no se podrá desactivar el reporte a través de archivos CSV.

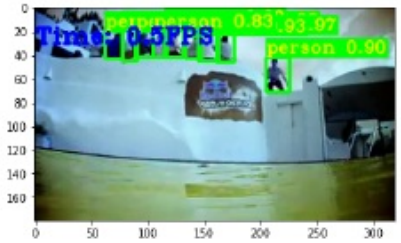

FIGURA 3.4. Ejemplo de resultados generados por el módulo.

Por otro lado, los archivos CSV deberán contener los siguientes datos:

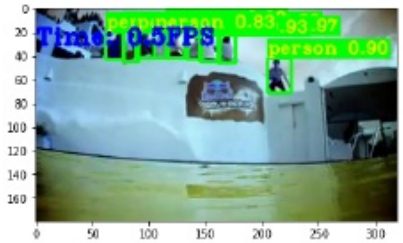

FIGURA 3.4. Ejemplo de resultados generados por el módulo.

calculada por el sistema a partir de los datos de posición, altura y posición en el *frame* (información proporcionada por el módulo).

20 Capítulo 3. Diseño e implementación

- Fecha y hora de detección.
- Coordenadas de detección dentro del *frame*.
- Número de dron en el que ocurre la detección.

Los datos serán entregados en el orden en el que han sido mencionados. Esto es de vital importancia de forma que el resto del sistema sea capaz de interpretar correctamente los datos entregados. Por otro lado, se debe recordar que el sistema desconoce la ubicación del dron en su recorrido de vuelo. Es por esto que el módulo no podrá reportar la ubicación en la que detectó el objeto. Esta deberá ser calculada por el sistema a partir de los datos de posición, altura y posición en el *frame* (información proporcionada por el módulo).

Capítulo 4

Ensayos y resultados

Este capítulo detalla las distintas pruebas que se hicieron sobre el módulo, así como el proceso de descarte de módulos a bordo de los drones. De igual manera, detalla los resultados obtenidos durante estos ensayos y los criterios utilizados para dar por cerrados los distintos requerimientos presentados en el capítulo 1. Finalmente, se explican los resultados mostrados durante una ejecución exitosa del módulo.

4.1. Descripción del proceso de pruebas

A pesar de que esto se estipuló en la concepción inicial del trabajo, conforme se dio su avance no se hizo posible llevar a cabo un proceso formal de pruebas al módulo. Sin embargo, el cliente estuvo involucrado directamente en el proceso de verificación de la funcionalidad del módulo a través de reuniones quincenales.

En estas reuniones fue posible hacer un seguimiento muy cercano y dar aprobación a las siguientes características para el funcionamiento del módulo. Durante estas reuniones se hizo posible dar por aprobado el código presentado para el trabajo, teniendo en cuenta que se cumplió con los siguientes criterios:

- el módulo obtiene correctamente los *frames* enviados por el cliente en el servidor que este dispone para tal fin.
- el módulo detecta intrusos en un tiempo no mayor a 4 segundos, aún cuando se ejecute en sistemas considerablemente menores a los que se espera lo ejecuten. En particular, utilizando CPU en vez de GPU durante la detección.
- la precisión del modelo es razonable.
- fue posible hacer la detección en tiempo real. Es decir, no solo se ejecutó con vídeos de prueba grabados por el cliente, sino que el módulo también se ejecutó correctamente durante vuelos transmitidos en tiempo real.
- el módulo es capaz de detectar intrusos en varios *streamings* de vídeo. Se deja la anotación que para que esto se logre, se deberá preprocesar el vídeo de forma que todos los *streamings* vengan combinados en un único *frame*. El módulo no tiene la capacidad de ejecutar varias instancias de manera paralela.
- se realizó una serie de pruebas durante estas reuniones, en las que se evidenció el funcionamiento del módulo.

21

Capítulo 4

Ensayos y resultados

Este capítulo detalla las distintas pruebas que se hicieron sobre el módulo, así como el proceso de descarte de módulos a bordo de los drones. De igual manera, detalla los resultados obtenidos durante estos ensayos y los criterios utilizados para dar por cerrados los distintos requerimientos presentados en el capítulo 1. Finalmente, se explican los resultados mostrados durante una ejecución exitosa del módulo.

4.1. Descripción del proceso de pruebas

A pesar de que esto se estipuló en la concepción inicial del trabajo, conforme se dio su avance no se hizo posible llevar a cabo un proceso formal de pruebas al módulo. Sin embargo, el cliente estuvo involucrado directamente en el proceso de verificación de la funcionalidad del módulo a través de reuniones quincenales.

En estas reuniones fue posible hacer un seguimiento muy cercano y dar aprobación a las siguientes características para el funcionamiento del módulo. Durante estas reuniones se hizo posible dar por aprobado el código presentado para el trabajo, teniendo en cuenta que se cumplió con los siguientes criterios:

- El módulo obtiene correctamente los *frames* enviados por el cliente en el servidor que este dispone para tal fin.
- El módulo detecta intrusos en un tiempo no mayor a 4 segundos, aún cuando se ejecute en sistemas considerablemente menores a los que se espera lo ejecuten. En particular, utilizando CPU en vez de GPU durante la detección.
- La precisión del modelo es razonable.
- Fue posible hacer la detección en tiempo real. Es decir, no solo se ejecutó con vídeos de prueba grabados por el cliente, sino que el módulo también se ejecutó correctamente durante vuelos transmitidos en tiempo real.
- El módulo es capaz de detectar intrusos en varios *streamings* de vídeo. Se deja la anotación que para que esto se logre, se deberá preprocesar el vídeo de forma que todos los *streamings* vengan combinados en un único *frame*. El módulo no tiene la capacidad de ejecutar varias instancias de manera paralela.
- Se realizó una serie de pruebas durante estas reuniones, en las que se evidenció el funcionamiento del módulo.

Para este visto bueno, no fueron considerados los siguientes requisitos, a pesar de que se trata de tareas que fueron ejecutadas:

- documentación del módulo.
- facilidad de uso del módulo.
- sistema operativo en el que se ejecutará el módulo. A pesar de que, durante la etapa de planeación, no se especificó el sistema operativo en el que se ejecutará el modelo, este fue diseñado para ejecutarse en Ubuntu 22.04.

4.2. Pruebas en Raspberry Pi

Desde la concepción inicial del trabajo, se estableció que se tendrían dos caminos para poder desarrollarlo: Por un lado, se planteó que el módulo debía ser capaz de reconocer intrusos en varios *streamings* de vídeo simultáneos (camino que finalmente se vio favorecido), o bien, hacer que el módulo fuese capaz de correr en equipos a bordo de los drones. Para esto, se llevaron a cabo una serie de pruebas y ensayos en equipos Raspberry Pi. Más concretamente, las especificaciones de los equipos sobre los que se hicieron las pruebas son:

- modelo: Raspberry Pi 2 modelo B
- sistema Operativo: Raspberry Pi OS (Debian 11 – Bullseye)
- CPU quad-core ARM Cortex-A7 900MHz
- memoria ram: 1 Gb.
- overclocking: Desactivado

Específicamente, el modelo con el que se tuvo los problemas más importantes, especialmente a la hora de la instalación fue la versión completa de Tensorflow, requerida para la ejecución del módulo. Cabe mencionar que las placas Raspberry Pi son compatibles con la versión Lite de Tensorflow. Dicho esto, se intentaron cinco métodos para la instalación de esta biblioteca. Ninguno de ellos fue exitoso. El proceso llevado a cabo en cada una de las cinco instalaciones fue el siguiente:

- instalación 1 [14]:
  - instalación de Tensorflow: utilizando PIP.
  - se clona el repositorio de GIT de Tensorflow.
- instalación 2 [15]:
  - instalación de Tensorflow: utilizando PIP y el comando –no-cache-dir.
- instalación 3 [16]:
  - instalación de Tensorflow Lite: utilizando PIP.
- instalación 4 [17]:
  - instalación de Libatlas: utilizando PIP.
  - instalación de Tensorflow: utilizando PIP.
- instalación 5 [18]:

Para este visto bueno, no fueron considerados los siguientes requisitos, a pesar de que se trata de tareas que fueron ejecutadas:

- Documentación del módulo.
- Facilidad de uso del módulo.
- Sistema operativo en el que se ejecutará el módulo. A pesar de que, durante la etapa de planificación, no se especificó el sistema operativo en el que se ejecutará el modelo, este fue diseñado para ejecutarse en Ubuntu 22.04.

4.2. Pruebas en Raspberry Pi

Desde la concepción inicial del trabajo, se estableció que se tendrían dos caminos para poder desarrollarlo: por un lado, se planteó que el módulo debía ser capaz de reconocer intrusos en varios *streamings* de vídeo simultáneos (camino que finalmente se vio favorecido), o bien, hacer que el módulo fuese capaz de correr en equipos a bordo de los drones. Para esto, se llevaron a cabo una serie de pruebas y ensayos en equipos Raspberry Pi. Más concretamente, las especificaciones de los equipos sobre los que se hicieron las pruebas son:

- Modelo: Raspberry Pi 2 modelo B
- Sistema operativo: Raspberry Pi OS (Debian 11 – Bullseye)
- CPU quad-core ARM Cortex-A7 900MHz
- Memoria ram: 1 Gb.
- Overclocking: Desactivado

Específicamente, el modelo con el que se tuvo los problemas más importantes a la hora de la instalación fue la versión completa de Tensorflow, requerida para la ejecución del módulo. Cabe mencionar que las placas Raspberry Pi son compatibles con la versión Lite de Tensorflow. Dicho esto, se intentaron cinco métodos para la instalación de esta biblioteca. Ninguno de ellos fue exitoso. El proceso llevado a cabo en cada una de las cinco instalaciones fue el siguiente:

- Instalación 1 [14]:
  - instalación de Tensorflow: utilizando PIP.
  - se clona el repositorio de GIT de Tensorflow.
- Instalación 2 [15]:
  - instalación de Tensorflow: utilizando PIP y el comando –no-cache-dir.
- Instalación 3 [16]:
  - instalación de Tensorflow Lite: utilizando PIP.
- Instalación 4 [17]:
  - instalación de Libatlas: utilizando PIP.
  - instalación de Tensorflow: utilizando PIP.
- Instalación 5 [18]:
  - Descarga de los paquetes y llaves disponibles en:

- descarga de los paquetes y llaves disponibles en https://packages.cloud.google.com/apt y https://packages.cloud.google.com/apt/doc/apt-key.gpg
- se agregaron estos paquetes y llaves a Debian.
- instalación de Libatlas: usando PIP.
- instalación de Tensorflow: usando PIP.

Al tratar de importar esta librería en el código de Python, se obtuvo un fallo en todas las instancias, por lo que se decidió no continuar con esta aproximación para dar solución a la problemática. Es importante destacar que, dado que el módulo se desarrolló tomando como base el modelo YOLOv3 original (no el modelo Lite), no se cumplía con las especificaciones requeridas por este tipo de equipos, y una segunda implementación del modelo utilizando la versión YOLOv3 Lite hubiese implicado en sí, el trabajo equivalente a un módulo completo adicional.

De igual manera atrae particular atención mencionar que la placa con la que se contaba era un modelo antiguo de Raspberry Pi (Raspberry Pi 2 Modelo B de febrero de 2015), que no cuenta con una compatibilidad tan completa como la que se da con una Raspberry Pi 4. Es, entonces, de vital importancia que, para iteraciones posteriores del desarrollo del trabajo, y en caso de que se decida continuar con el desarrollo de módulos a bordo de los drones, considerar placas más avanzadas, como Raspberry Pi 4.

4.3. Caso de Uso

Una vez establecido el funcionamiento interno del módulo, se hace importante también tener en cuenta el flujo de trabajo para iniciarlo y ponerlo en funcionamiento. Se deben tener en cuenta las siguientes consideraciones para que se pueda iniciar el módulo de la manera correcta:

- se debe contar con el link de conexión al servidor de transmisión.
- contar con el link de conexión al servidor receptor.
- el servidor de conexión debe estar transmitiendo.
- llave de transmisión proporcionada por el cliente.

En total, se debe seguir n pasos (contando el paso 0 de inicio de sesión en el sistema operativo y 1 de instalación) para lograr el funcionamiento esperado del módulo:

1. el usuario deberá iniciar sesión en Ubuntu 22.04 LTS. Deberá cerciorarse que lo haga utilizando xorg. A pesar de que esta versión Ubuntu es compatible con el gestor de ventanas Wayland, su uso podrá causar una ejecución inestable del módulo.
2. instalación del módulo: Este paso contempla la instalación del módulo en una instalación nueva de Ubuntu 22.04 LTS. Para esto, el usuario se deberá remitir al manual de instalación que se entregó al cliente. Dado que esta versión no cuenta con una versión empaquetada en formato .deb (compatible con los sistemas basados en Debian), este paso incluirá la descarga de un IDE y la correspondiente apertura del código fuente.

  - https://packages.cloud.google.com/apt
  - https://packages.cloud.google.com/apt/doc/apt-key.gpg
- Se agregaron estos paquetes y llaves a Debian.
- Instalación de Libatlas: usando PIP.
- Instalación de Tensorflow: usando PIP.

Al tratar de importar esta biblioteca en el código de Python, se obtuvo un fallo en todas las instancias, por lo que se decidió no continuar con esta aproximación para dar solución a la problemática. Es importante destacar que, dado que el módulo se desarrolló tomando como base el modelo YOLOv3 original (no el modelo Lite), no se cumplía con las especificaciones requeridas por este tipo de equipos, y una segunda implementación del modelo utilizando la versión YOLOv3 Lite hubiese implicado en sí, el trabajo equivalente a un módulo completo adicional.

De igual manera atrae particular atención mencionar que la placa con la que se contaba era un modelo antiguo de Raspberry Pi (Raspberry Pi 2 Modelo B de febrero de 2015), que no cuenta con una compatibilidad tan completa como la que se da con una Raspberry Pi 4. Es, entonces, muy importante considerar placas más avanzadas, como la Raspberry 4, en caso de que se decida continuar con el desarrollo de módulos a bordo de los drones.

4.3. Caso de Uso

Una vez establecido el funcionamiento interno del módulo, se hace importante también tener en cuenta el flujo de trabajo para iniciarlo y ponerlo en funcionamiento. Se contemplar las siguientes consideraciones para que se pueda iniciar el módulo de la manera correcta:

- Se debe contar con los links a los servidores de transmisión y recepción.
- El servidor de conexión debe estar transmitiendo.
- Se debe contar con la llave de transmisión proporcionada por el cliente.

En total, se debe seguir n pasos (contando el paso 0 de inicio de sesión en el sistema operativo y 1 de instalación) para lograr el funcionamiento esperado del módulo:

1. El usuario deberá iniciar sesión en Ubuntu 22.04 LTS. Deberá cerciorarse que lo haga utilizando xorg. A pesar de que esta versión Ubuntu es compatible con el gestor de ventanas Wayland, su uso podrá causar una ejecución inestable del módulo.
2. Instalación del módulo: este paso contempla la instalación del módulo en una instalación nueva de Ubuntu 22.04 LTS. Para esto, el usuario se deberá remitir al manual de instalación que se entregó al cliente. Como esta versión no cuenta con una versión empaquetada en formato .deb (compatible con los sistemas basados en Debian), este paso incluirá la descarga de un IDE y la correspondiente apertura del código fuente.
3. Ejecutar el código fuente en el archivo Main. Esto permitirá la apertura de la interfaz gráfica (visible en la figura 3.2). Aquí el usuario podrá introducir el link proporcionado por el cliente.

3. ejecutar el código fuente en el archivo Main. Esto permitirá la apertura de la interfaz gráfica (visible en la imagen n). Aquí el usuario podrá introducir el link proporcionado por el cliente.
4. en caso de una conexión exitosa al servidor de transmisión, el usuario verá que aparecerá una ventana en su pantalla mostrando los *frames* procesados por el módulo. Es por esto que dentro de los requisitos del sistema para esta versión del módulo.
5. el usuario encontrará en su carpeta de documentos el archivo CSV con todas las detecciones de la fecha.
6. a fin de retransmitir los *frames* procesados de regreso a los servidores del cliente, el usuario deberá abrir OBS Studio de manera independiente. Dentro de este programa, el usuario deberá seguir los siguientes pasos:
   6.1 crear un nuevo origen de datos, que corresponderá a la ventana generada por el módulo (mencionada el paso 3).
   6.2 dirigirse a las configuraciones de OBS Studio, particularmente en la pestaña *Stream*.
   6.3 proporcionar el link de *streaming* y su respectiva llave de transmisión.
7. observar el *streaming* en el visor que el cliente disponga para este fin.

De igual manera, una vez logrado esto, es importante observar los resultados obtenidos durante la ejecución del módulo:

- el módulo se ejecutó con un promedio de 0,54 fps. Este valor es mayor al requisito del cliente de procesar un número mayor a 0,25 fps. Estos valores se lograron con CPU, ejecutados en una máquina virtual.
- e promedio se pierden en la transmisión, un 15 % de los *frames*. Si bien, el módulo los reporta (según la imagen n), es capaz de recuperarse satisfactoriamente de esta situación.
- el modelo requiere un breve período de calentamiento para lograr las métricas mencionadas arriba. En este momento los fps son menores a los que logrará el modelo en su ejecución normal.

4. En el caso de una conexión exitosa al servidor de transmisión, el usuario verá que aparecerá una ventana en su pantalla mostrando los *frames* procesados por el módulo. Este no fue uno de los requisitos expuestos en la etapa de planificación del proyecto: fue agregado posteriormente durante el desarrollo del trabajo.
5. El usuario encontrará en su carpeta de documentos el archivo CSV con todas las detecciones de la fecha.
6. A fin de retransmitir los *frames* procesados de regreso a los servidores del cliente, el usuario deberá abrir OBS Studio de manera independiente. Dentro de este programa, el usuario deberá seguir los siguientes pasos:
   6.1 Crear un nuevo origen de datos, que corresponderá a la ventana generada por el módulo (mencionada el paso 3).
   6.2 Dirigirse a las configuraciones de OBS Studio, particularmente en la pestaña *Stream*.
   6.3 Proporcionar el link de *streaming* y su respectiva llave de transmisión.
7. Observar el *streaming* en el visor que el cliente disponga para este fin.

De igual manera, una vez logrado esto, es importante observar los resultados obtenidos durante la ejecución del módulo:

- El módulo se ejecutó con un promedio de 0,54 fps. Este valor es mayor al requisito del cliente de procesar un número mayor a 0,25 fps. Estos valores se lograron con CPU, ejecutados en una máquina virtual.
- En promedio se pierden en la transmisión, un 15 % de los *frames*. Si bien, el módulo los reporta (según la imagen 3.4), es capaz de recuperarse satisfactoriamente de esta situación.
- El modelo requiere un breve período de calentamiento para lograr las métricas mencionadas arriba. En este momento los fps son menores a los que logrará el modelo en su ejecución normal.

Capítulo 5

Conclusiones

En este capítulo se detallan los resultados obtenidos del proceso del desarrollo del módulo. Es decir, se hace un recuento de las características finales del mínimo producto viable que resultó de este proceso sin detallar ejecuciones específicas.

5.1. Resultados Obtenidos

Del desarrollo del modelo surgió una serie de conclusiones que representan, o bien, trabajos futuros (detallados en secciones posteriores), o recomendaciones para que se pueda sacar el provecho adecuado al módulo.

Por un lado, sale a relucir la importancia del uso de las versiones adecuadas de software. El no hacerlo puede implicar que el usuario vea inestabilidades fuertes en el funcionamiento del módulo. Por ejemplo, utilizar versiones no LTS *(Long Term Support)* de Ubuntu, implicará que, en un tiempo corto, el sistema pueda acceder a los paquetes requeridos para la correcta ejecución.

Igualmente, es muy importante tener en cuenta que a pesar de que es posible tomar algunos modelos generales de visión por computadora, estos son de fácil modificación, de forma que se hace muy evidente a través del desarrollo de este trabajo, que crear módulos personalizados de visión por computadora es muy sencillo. Esto puede ser de gran utilidad para diferentes industrias que pueden aprovechar el poder de este tipo de herramientas, manteniendo a su personal ocupado con tareas que generen un mayor valor.

Ahora bien, dados algunos cambios que se presentaron según avanzó el trabajo, no fue posible cumplir estrictamente con lo planeado: dados cambios en las prioridades en el proyecto, el proceso de pruebas no pudo ser llevado a cabo, dado que se le dio prioridad al sistema de reporte visual. Esto implica que el usuario tiene más maneras de monitorear los resultados del módulo y tomar decisiones informadas. De igual manera, se hace más sencillo identificar falsos positivos y falsos negativos que puedan aparecer.

5.2. Tiempos de ejecución

A pesar de que los tiempos de ejecución medidos a través del uso de procesamiento con CPU y máquinas virtuales cumplen con lo estipulado en la fase de definición de los requisitos del módulo con el cliente (según se detallan en el capítulo 4), dadas las restricciones de los *frameworks* utilizados comúnmente en la implementación de sistemas de inteligencia artificial, no fue posible probar con

25

Capítulo 5

Conclusiones

En este capítulo se detallan los resultados obtenidos del proceso del desarrollo del módulo. Es decir, se hace un recuento de las características finales del mínimo producto viable que resultó de este proceso sin detallar ejecuciones específicas.

5.1. Resultados Obtenidos

Del desarrollo del modelo surgieron una serie de conclusiones que representan, o bien, trabajos futuros (detallados en secciones posteriores), o recomendaciones para que se pueda sacar el provecho adecuado al módulo.

Por un lado, sale a relucir la importancia del uso de las versiones adecuadas de software. El no hacerlo puede implicar que el usuario vea fuertes inestabilidades en el funcionamiento del módulo. Por ejemplo, utilizar versiones no LTS *(Long Term Support)* de Ubuntu, implicará que, en un tiempo corto, el sistema no podrá acceder a los paquetes requeridos para la correcta ejecución.

Igualmente, es muy importante tener en cuenta que a pesar de que es posible tomar algunos modelos generales de visión por computadora, estos son de fácil modificación. Lo anterior expuesto se hace muy evidente a través del desarrollo de este trabajo, es decir, crear módulos personalizados de visión por computadora es muy sencillo. Esto puede ser de gran utilidad para diferentes industrias que pueden aprovechar el poder de este tipo de herramientas, manteniendo a su personal ocupado con tareas que generen un mayor valor.

Ahora bien, dados algunos cambios que se presentaron según avanzó el trabajo, no fue posible cumplir estrictamente con lo planeado: por cambios en las prioridades en el proyecto, el proceso de pruebas no pudo ser llevado a cabo, porque se le dio prioridad al sistema de reporte visual. Esto implica que el usuario tiene más maneras de monitorear los resultados del módulo y tomar decisiones informadas. De igual manera, se hace más sencillo identificar falsos positivos y falsos negativos que puedan aparecer.

5.2. Tiempos de ejecución

A pesar de que los tiempos de ejecución medidos a través del uso de procesamiento con CPU y máquinas virtuales cumplen con lo estipulado en la fase de definición de los requisitos del módulo con el cliente (según se detallan en el capítulo 4), dadas las restricciones de los *frameworks* utilizados comúnmente en la implementación de sistemas de inteligencia artificial, no fue posible probar con

las tarjetas gráficas apropiadas, dado que no se contó con el equipo. Sin embargo, se espera que este rendimiento mejore sustancialmente.

A fin de poder realizar estas pruebas, sin embargo, se requerirá contar con una de las siguientes opciones:

1. Computadora que cuente, idealmente, con una tarjeta gráfica NVidia Tesla T4 o NVidia Titan X (según referenciado en el capítulo 2).
2. 2. Código que no requiera de la generación de una ventana para la retransmisión de los *frames*. Esto permitirá montar el módulo en un servicio de *Cloud Computing* (como Google Cloud), en donde es posible alquilar tarjetas gráficas con este tipo de especificaciones, o bien, ejecutar pruebas sobre el código en Google Colab, en donde se podrá medir más fielmente los tiempos de ejecución del módulo.

5.3. Trabajos futuros

Teniendo en cuenta que se trata de un mínimo producto viable, cabe aclarar que los siguientes trabajos son aún necesarios para que el cliente pueda contar con un módulo de inteligencia artificial que se ajuste de manera adecuada a sus necesidades. Dentro de estos trabajos futuros están:

- añadir al módulo la capacidad de retransmitir los *frames* procesados. Esto eliminará la necesidad de utilizar software externo como OBS Studio.
- el módulo deberá ser optimizado muy fuertemente. Esto implica que se sigan los siguientes pasos:
  - actualmente el módulo detecta la totalidad de las clases del *dataset* COCO y descarta las clases que no se requieren durante el paso de reporte. Esto implica que el sistema hace una gran cantidad de cálculos innecesarios, correspondientes a clases que no se requieren por el cliente.
  - se deberá reentrenar los pesos a fin de evitar la ocurrencia de falsos positivos, así como reducir la ocurrencia de falsos negativos.
  - para versiones más avanzadas del código, se hará necesaria la generación, en conjunto con el cliente, de definiciones más precisas de qué se considera aceptable en una matriz de confusión.
- algunos de los requisitos del cliente (particularmente la detección en vuelos nocturnos) no pudieron ser cubiertos. Es imperativo entonces hacer las pruebas y ajustes correspondiente una vez exista el material de vuelos en horas de la noche.
- empaquetar el módulo, idealmente en formato .deb, de forma que se pueda hacer una fácil distribución del módulo.

26 Capítulo 5. Conclusiones

las tarjetas gráficas apropiadas, dado que no se contó con el equipo. Sin embargo, se espera que este rendimiento mejore sustancialmente.

A fin de poder realizar estas pruebas, sin embargo, se requerirá contar con una de las siguientes opciones:

1. Computadora que cuente, idealmente, con una tarjeta gráfica NVidia Tesla T4 o NVidia Titan X (según referenciado en el capítulo 2).
2. Código que no requiera de la generación de una ventana para la retransmisión de los *frames*. Esto permitirá montar el módulo en un servicio de *Cloud Computing* (como Google Cloud), en donde es posible alquilar tarjetas gráficas con este tipo de especificaciones, o bien, ejecutar pruebas sobre el código en Google Colab, en donde se podrán medir más fielmente los tiempos de ejecución del módulo.

5.3. Trabajos futuros

Teniendo en cuenta que se trata de un mínimo producto viable, cabe aclarar que los siguientes trabajos son aún necesarios para que el cliente pueda contar con un módulo de inteligencia artificial que se ajuste de manera adecuada a sus necesidades. Dentro de estos trabajos futuros están:

- Añadir al módulo la capacidad de retransmitir los *frames* procesados. Esto eliminará la necesidad de utilizar software externo como OBS Studio.
- El módulo deberá ser optimizado muy fuertemente. Esto implica que se sigan los siguientes pasos:
  - Actualmente el módulo detecta la totalidad de las clases del *dataset* COCO y descarta las clases que no se requieren durante el paso de reporte. Esto implica que el sistema hace una gran cantidad de cálculos innecesarios, correspondientes a clases que no se requieren por el cliente.
  - Se deberá reentrenar los pesos a fin de evitar la ocurrencia de falsos positivos, así como reducir la ocurrencia de falsos negativos.
  - Para versiones más avanzadas del código, se hará necesaria la generación, en conjunto con el cliente, de definiciones más precisas de qué se considera aceptable en una matriz de confusión.
- Algunos de los requisitos del cliente (particularmente la detección en vuelos nocturnos) no pudieron ser cubiertos. Es imperativo entonces hacer las pruebas y ajustes correspondiente una vez exista el material de vuelos en horas de la noche.
- Empaquetar el módulo, idealmente en formato .deb, de forma que distribuir con facilidad.
- Durante el desarrollo de este trabajo se descartó el uso de Raspberry Pi para el desarrollo de módulos a bordo del dron dadas las limitaciones del modelo utilizado (Raspberry Pi 2 Modelo B). Se deberá utilizar placas Rasbperry Pi más avanzadas, de preferencia Raspberry Pi 4.

Apéndice A

Tablas extendidas

TABLA A.1. Software requerido para el desarrollo y la ejecución del módulo

Paquete	Tipo	Licencia	Versión
PIP	Gestor de bibliotecas	MIT	22.0.2
Numpy	Biblioteca de Python	BSD	1.21.5
OpenCV	Biblioteca de Python	Apache 2	4.5.4
Matplotlib	Biblioteca de Python	BSD	3.6.2
Tensorflow	Biblioteca de Python	Apache 2	2.10.1
Pycharm	Software completo	Apache 2	2022.3.3 Community edition
OBS Studio	Software completo	GPLv2	28.1.2 (64 Bit)
Python	Lenguaje de programación	Python Software Foundation License Version 2	3.10.6
Ubuntu	Sistema operativo	Creative Commons CC-BY-SA version 3.0 UK	22.04.1 LTS (64 Bit)

27

Apéndice A

Tablas extendidas

TABLA A.1. Software requerido para el desarrollo y la ejecución del módulo.

Paquete	Tipo	Licencia	Versión
PIP	Gestor de bibliotecas	MIT	22.0.2
Numpy	Biblioteca de Python	BSD	1.21.5
OpenCV	Biblioteca de Python	Apache 2	4.5.4
Matplotlib	Biblioteca de Python	BSD	3.6.2
Tensorflow	Biblioteca de Python	Apache 2	2.10.1
Pycharm	Software completo	Apache 2	2022.3.3 Community edition
OBS Studio	Software completo	GPLv2	28.1.2 (64 Bit)
Python	Lenguaje de programación	Python Software Foundation License Version 2	3.10.6
Ubuntu	Sistema operativo	Creative Commons CC-BY-SA version 3.0 UK	22.04.1 LTS (64 Bit)