Detección automática de peatones con diferentes variaciones

ALEJANDRO AGUIRRE RUIZ

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

FACULTAD DE INGENIERÍA

INGENIERÍA DE SOFTWARE

MEDELLÍN

2023

Detección automática de peatones con diferentes variaciones

ALEJANDRO AGUIRRE RUIZ

Trabajo de grado para optar al título de
Ingeniero de software

Asesor Técnico
Rubén Fonnegra
Juan Carlos Briñez De León

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO
FACULTAD DE INGENIERÍA
INGENIERÍA DE SOFTWARE
MEDELLÍN
2023

Pág.

Contenido

Lista de figuras

Lista de anexos

Resumen

El reconocimiento de peatones es una tarea importante en la visión por computadora y en la seguridad vial. El objetivo es detectar la presencia de peatones en imágenes o videos y, en algunos casos, su posición y tamaño. Existen diversas técnicas para el reconocimiento de peatones, como el clasificador Haar Cascade, HOG + SVM y redes neuronales convolucionales (CNN). Estas técnicas pueden adaptarse a diferentes variaciones, como diferentes tamaños de ventana, escalas de imagen, ángulos de vista y resoluciones. La elección de la técnica adecuada dependerá de las necesidades específicas de la aplicación y de los recursos computacionales disponibles. El reconocimiento de peatones es fundamental para el desarrollo de sistemas de transporte inteligente, vehículos autónomos y aplicaciones de seguridad peatonal.

Abstract

Pedestrian recognition is a vital task in computer vision and traffic safety. The objective is to detect the presence of pedestrians in an image or video, and possibly their position and size. There are several common techniques for pedestrian recognition, such as the Haar Cascade Classifier, HOG + SVM, and Convolutional Neural Networks (CNN). These techniques can be adapted to different variations, such as different window sizes, image scales, viewing angles, and resolutions. The choice of the appropriate technique depends on the specific needs of the application and the available computational resources. Pedestrian recognition is crucial for developing intelligent transportation systems, autonomous vehicles, and pedestrian safety applications.

Glosario

- **Detección de objetos:** tarea de identificar la presencia y localización de objetos específicos en una imagen o video.
- Redes neuronales convolucionales (CNN): un tipo de red neuronal artificial que se utiliza comúnmente en tareas de visión por computadora, como la detección de objetos.
- **YOLO (You Only Look Once):** un algoritmo de detección de objetos que utiliza una red neuronal convolucional para identificar objetos y sus ubicaciones en tiempo real.
- **YOLOv5:** la última versión del algoritmo YOLO, que ha mejorado la precisión y velocidad de detección en comparación con versiones anteriores.
- mAP (Mean Average Precision): una métrica utilizada para evaluar el rendimiento de los modelos de detección de objetos.
- **Dataset:** un conjunto de datos de entrenamiento y/o evaluación utilizada para entrenar y evaluar modelos de detección de objetos.
- **COCO (Common Objects in Context):** un conjunto de datos de detección de objetos que contiene imágenes y etiquetas para 80 categorías de objetos diferentes, incluidos los peatones.
- **Caltech Pedestrian Dataset**: un conjunto de datos de detección de peatones que contiene imágenes y etiquetas de peatones en diferentes entornos y situaciones.
- **CityPersons:** otro conjunto de datos de detección de peatones que contiene imágenes y etiquetas de peatones en áreas urbanas.
- **Precision:** una métrica de evaluación de la detección de objetos que mide la proporción de detecciones verdaderas entre todas las detecciones propuestas.
- **False positive:** una detección que no es un objeto real, sino que es un error de la detección del modelo.
- **False negative:** un objeto real que no es detectado por el modelo, es decir, un error de omisión.
- **Recall:** Es una métrica utilizada en la evaluación de modelos de aprendizaje automático para medir la proporción de verdaderos positivos

Introducción

La detección automática de peatones es una tarea importante en la visión por computadora y en la seguridad vial. El objetivo es detectar la presencia de peatones en una imagen o video, y posiblemente también su posición y tamaño. A continuación, se presentan algunas técnicas comunes para la detección automática de peatones y cómo se pueden adaptar a diferentes variaciones.

- **Haar Cascade Classifier**

El Haar Cascade Classifier es un algoritmo de detección de objetos que utiliza características tipo Haar para detectar peatones. Estas características se basan en la intensidad de los píxeles en diferentes partes de la imagen. El clasificador se entrena utilizando un conjunto de imágenes positivas que contienen peatones y un conjunto de imágenes negativas que no los contienen. (Viola, 2004)

Una variación común de esta técnica es utilizar diferentes tamaños de ventana para detectar peatones de diferentes tamaños. También se pueden entrenar clasificadores específicos para diferentes ángulos de vista, para mejorar la detección en situaciones en las que el peatón puede estar parcialmente oculto.

- **HOG + SVM**

HOG (Histogram of Oriented Gradients) es una técnica de extracción de características que busca patrones de gradiente en la imagen. SVM (Support Vector Machine) es un clasificador que se entrena para distinguir entre peatones y no peatones utilizando estas características. (Dalal, 2005)

Una variación común de esta técnica es utilizar diferentes tamaños de ventana y escalas de imagen para detectar peatones de diferentes tamaños. También se pueden utilizar técnicas de pirámide de imágenes para mejorar la detección en diferentes resoluciones.

- **Redes Neuronales Convolucionales (CNN)**

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal profunda que se utilizan para tareas de reconocimiento visual, incluyendo la detección de objetos. Estas redes están compuestas por capas de neuronas convolucionales que procesan las imágenes de entrada y aprenden automáticamente características útiles para la tarea de detección de objetos. Las capas convolucionales utilizan filtros para extraer características específicas de la imagen, como bordes y texturas, y luego pasan esta información a través de capas de pooling para reducir la dimensión de la salida y hacer que el procesamiento sea más eficiente. Finalmente, la información se pasa a través de una o varias capas completamente conectadas que se encargan de la clasificación y detección de objetos. En la tarea de detección de objetos, las CNN pueden detectar la presencia de objetos en diferentes escalas y posiciones en la imagen, lo que las hace muy útiles para la detección de peatones y otros objetos en escenas complejas. (Krizhevsky, 2012)

Una variación común de esta técnica es utilizar diferentes arquitecturas de red, como Faster R-CNN o YOLO, para detectar peatones. Estas arquitecturas pueden detectar múltiples objetos en una sola pasada, lo que las hace muy rápidas y eficientes.

Existen varias arquitecturas de redes neuronales convolucionales (CNN) que se han utilizado para la detención de objetos incluyendo la detección de peatones

- **R-CNN:** La arquitectura R-CNN (Region-based CNN) fue una de las primeras en utilizar CNN para la detección de objetos (Girshick R. D., 2014). Esta arquitectura se divide en dos etapas:
    - La primera etapa propone una serie de regiones candidatas en la imagen que pueden contener objetos.
    - La segunda etapa clasifica estas regiones candidatas utilizando una CNN.

- **Fast R-CNN:** Esta arquitectura es una versión mejorada de R-CNN, que combina la propuesta de regiones y la clasificación en una sola red. Fast R-CNN utiliza una red convolucional para generar una única propuesta de región y luego clasifica esta propuesta. (Girshick R. (., 2015)
- **Faster R-CNN:** La arquitectura Faster R-CNN es una evolución de Fast R-CNN que utiliza una red convolucional para generar propuestas de regiones en lugar de hacerlo manualmente. Esto permite una detección de objetos más rápida y precisa. (Ren, 2016)
- **Mask R-CNN:** Esta arquitectura es una extensión de Faster R-CNN que agrega una rama de segmentación para la generación de máscaras de objetos. Mask R-CNN es capaz de detectar objetos y generar máscaras precisas al mismo tiempo. (He, 2017)
- **YOLO:** La arquitectura YOLO (You Only Look Once) es un enfoque diferente para la detección de objetos que utiliza una sola red para predecir las cajas delimitadoras y las probabilidades de clase de los objetos en una sola pasada. Esto hace que YOLO sea muy rápida y eficiente. (Redmon, 2016)

1. Problema, justificación y objetivos

1.1 Planteamiento del problema

Según los informes de la literatura, la seguridad vial es un tema de gran importancia a nivel mundial, especialmente en países con altos índices de siniestralidad vial como México y Brasil. Bajo esta temática, se ha identificado que uno de los principales factores que contribuyen a los accidentes de tráfico es la falta de atención y el comportamiento imprudente de los peatones. Esto hace que el reconocimiento oportuno de los peatones en las vías sea de gran importancia en la comunidad científica y en el sector de transporte, ya que puede prevenir accidentes y salvar vidas.

Aunque la literatura informa de numerosos trabajos sobre el reconocimiento automático de objetos y personas en imágenes, muchas de las experiencias exitosas se han basado en algoritmos computacionales que utilizan técnicas de procesamiento de imágenes y aprendizaje profundo. Por ejemplo, en el reconocimiento automático de peatones, los mayores avances han venido de la mano del uso de imágenes de cámaras de video en tiempo real. A pesar de que se han logrado avances significativos a través de técnicas avanzadas de procesamiento de imágenes y de aprendizaje profundo, la falta de un sistema generalizado para cualquier tipo de sistema de adquisición de imágenes sigue siendo un desafío para la implementación de estas soluciones en el sector de transporte. Como resultado, se necesitan ajustes específicos para adaptar una solución de reconocimiento automático de peatones a las particularidades de cada área urbana y de cada tipo de cámara de video utilizada en el monitoreo de las vías. Por lo tanto, el presente estudio se enfoca en el desarrollo de un sistema de reconocimiento automático de peatones en tiempo real que sea eficiente y adaptable a las particularidades de cada área urbana.

1.2 Formulación

El reconocimiento automático de peatones en vías es un tema de gran importancia en la actualidad, ya que los accidentes de tráfico son una de las principales causas de mortalidad en todo el mundo, y los peatones son uno de los grupos más vulnerables en las vías. A menudo, los accidentes que involucran a peatones son el resultado de la falta de atención o el

comportamiento imprudente de los mismos, lo que hace que el reconocimiento oportuno de los peatones en las vías sea crucial para prevenir accidentes y salvar vidas.

En este contexto, la tecnología de reconocimiento automático de peatones en tiempo real puede ser una herramienta muy útil para mejorar la seguridad vial, ya que puede permitir a los sistemas de transporte tomar medidas preventivas antes de que se produzcan accidentes. Además, esta tecnología puede ser implementada en una variedad de escenarios de monitoreo de vías, como en cámaras de tráfico, sistemas de vigilancia de peatones en intersecciones y en vehículos autónomos.

Aunque existen diversos estudios sobre el reconocimiento automático de peatones, aún existen desafíos importantes que deben ser abordados para lograr soluciones eficientes y adaptables a las particularidades de cada área urbana. Por lo tanto, este tema presenta un gran potencial para la investigación y desarrollo de soluciones tecnológicas innovadoras que puedan mejorar la seguridad vial y salvar vidas. Cada área urbana.

1.3 Objetivos

1.3.1 General

Mejorar la detección de peatones a partir de imágenes, videos y capturas en tiempo real. Para lograr este objetivo, se realizará una configuración y reentrenamiento de una red neural específica.

1.3.2 Específicos

- Identificar datasets públicos que contentan imágenes etiquetadas de personas para entrenar la red neuronal.
- Configurar el ambiente en un servidor para instalar la red neuronal y prepararlo para el reentrenamiento de esta.
- Realizar el reentrenamiento de la red neuronal para mejorar su capacidad de detección de peatones.
- Evaluar el desempeño del modelo de clasificación utilizando técnicas de validación cruzadas y métricas de evaluación.
- Implementar la red neuronal en una Jetson nano para realizar la detección de peatones en tiempo real.
- Evaluar el desempeño del modelo en capturas de video en tiempo real para garantizar su eficacia en situaciones reales.

2. Dataset

Un dataset, también conocido como conjunto de datos, es un conjunto estructurado de datos que se utiliza para analizar y extraer información. En otras palabras, es un conjunto de observaciones o datos que se han recopilado, procesado y organizado de una manera específica para su uso en la investigación o análisis.

Un dataset puede incluir diferentes tipos de datos, como texto, imágenes, audio, videos, números, fechas, entre otros. Estos datos se pueden utilizar para desarrollar y entrenar modelos de aprendizaje automático y otros algoritmos de inteligencia artificial.

Los datasets pueden ser públicos o privados y se utilizan en diversas áreas, como la investigación científica, la salud, la educación, la tecnología, el comercio, la industria y muchas más. Además, algunos datasets son específicos para un propósito particular, mientras que otros son más generales y se utilizan en diferentes áreas de investigación.

Existen varios datasets públicos utilizados para el reconocimiento de peatones, entre ellos los más comunes son:

- **COCO (Common Objects in Context):** este es un dataset ampliamente utilizado que contiene más de 330.000 imágenes etiquetadas de objetos comunes en diferentes contextos y situaciones. Las imágenes están etiquetadas con información como las categorías de objetos presentes, las coordenadas de las regiones de interés, entre otras.
  Puedes encontrar más información aquí: https://cocodataset.org/#home
- **Caltech Pedestrian Dataset:** Este dataset contiene alrededor de 10.000 imágenes etiquetadas de peatones capturadas en entornos urbanos y en diferentes situaciones de iluminación y clima. Cada imagen tiene una resolución de 640x480 píxeles y está etiquetada con información como la ubicación y tamaño de los peatones.
  Puedes encontrar más información aquí:
  https://www.kaggle.com/datasets/kalvinquackenbush/caltechpedestriandataset

- **INRIA Person Dataset:** Este dataset contiene más de 6000 imágenes etiquetadas de personas en diferentes situaciones y fondos. Las imágenes están etiquetadas con información como la ubicación y tamaño de las personas, lo que lo convierte en un dataset popular para la detección de personas en entornos urbanos y rurales.
  Puedes encontrar más información aquí:
  https://paperswithcode.com/dataset/inria-person

- **CityPersons Dataset:** Este dataset contiene más de 5.000 imágenes etiquetadas de peatones en entornos urbanos. Las imágenes están etiquetadas con información como la ubicación y tamaño de los peatones, así como su orientación y otras características relevantes para la detección de peatones en áreas urbanas.
  Puedes encontrar más información aquí:
  https://paperswithcode.com/dataset/citypersons

2.1. Descripción del dataset

COCO (Common Objects in Context) es uno de los datasets más populares en el campo de la visión por computadora. Contiene más de 330.000 imágenes etiquetadas de objetos comunes en diferentes contextos y situaciones, lo que lo convierte en un conjunto de datos muy diverso y útil para entrenar modelos de aprendizaje automático. (Lin, 2014)

Las imágenes en COCO están etiquetadas con información detallada, como las categorías de objetos presentes, las coordenadas de las regiones de interés, las segmentaciones de objetos, las relaciones entre objetos, entre otros. Además, COCO incluye una variedad de categorías de objetos, que van desde animales y vehículos hasta objetos domésticos y personas, lo que lo convierte en un dataset muy completo.

COCO también cuenta con un conjunto de datos de detección de objetos, que se utiliza para evaluar la capacidad de los modelos de detección de objetos. Este conjunto de datos incluye 80 categorías de objetos comunes, como perros, gatos, bicicletas, automóviles, personas, entre otros.

COCO es ampliamente utilizado en la investigación y desarrollo de modelos de aprendizaje automático para la detección y segmentación de objetos. Los modelos entrenados en COCO han logrado resultados impresionantes en varios desafíos de visión por computadora, como el COCO Detection Challenge y el COCO Segmentation Challenge.

Ilustración 1 Imagen de la página de COCO

2.2. Antecedentes asociados al dataset

A continuación, se presentas antecedentes relacionados con proyectos de detección de objetos utilizando el dataset, los siguientes proyectos son algunos que utilizaron el dataset y como se han utilizado para investigaciones y desarrollo de algoritmos de detección y segmentación de objetos.

- Este proyecto presenta una arquitectura de red neuronal llamada Mask R-CNN que se puede entrenar en el dataset COCO para la detección y segmentación de objetos en imágenes. (Kaiming He, 2017)

- Este proyecto presenta una versión mejorada del algoritmo de detección de objetos YOLO que se puede entrenar en COCO y otros datasets para la detección de objetos en tiempo real. (Farhadi., s.f.)

- Este proyecto presenta una biblioteca de código abierto para la detección y segmentación de objetos en imágenes y videos que se puede entrenar en COCO y otros datasets. (Ros19)

2.3. Descarga y visualización del conjunto de datos

- Abre una terminal en tu computadora
- Navega a la carpeta donde deseas descargar el conjunto de datos COCO utilizando el comando cd. Por ejemplo, si deseas descargarlo en la carpeta "Descargas", escribe cd Descargas.
- Descarga los archivos COCO utilizando el comando wget. Por ejemplo, para descargar el conjunto de datos de entrenamiento, escribe el siguiente comando y presiona Enter

  wget http://images.cocodataset.org/zips/train2017.zip

- Si deseas descargar el conjunto de datos de validación, reemplaza "train2017" en el comando anterior con "val2017". Por ejemplo, escribe el siguiente comando y presiona Enter

  wget http://images.cocodataset.org/zips/val2017.zip

- Si deseas descargar las anotaciones de los objetos, utiliza el siguiente comando

  wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip

- Si deseas descargar las anotaciones de "cosas" (por ejemplo, cielo, agua, pasto), utiliza el siguiente comando

  wget http://images.cocodataset.org/annotations/stuff_annotations_trainval2017.zip

- Una vez que se complete la descarga, descomprime los archivos ZIP utilizando el comando unzip. Por ejemplo, si deseas descomprimir el conjunto de datos de entrenamiento, escribe el siguiente comando y presiona Enter

  unzip train2017.zip

- Si deseas descomprimir los archivos de anotaciones, utiliza el siguiente comando

  unzip annotations_trainval2017.zip

- Si deseas descomprimir los archivos de anotaciones de "cosas", utiliza el siguiente comando

  unzip stuff_annotations_trainval2017.zip

Es importante tener en cuenta que los archivos COCO ocupan varios gigabytes de espacio en disco, por lo que se recomienda contar con una conexión a internet rápida y suficiente espacio de almacenamiento en el disco duro para descargarlos y almacenarlos. Además, es

necesario tener en cuenta que COCO es un conjunto de datos grande y complejo, por lo que se recomienda familiarizarse con su estructura y contenido antes de utilizarlo en cualquier proyecto.

3. YOLO

La detección de objetos en imágenes y videos es un área de investigación en constante evolución en el campo de la visión por computadora. La capacidad de detectar y localizar objetos en tiempo real tiene un amplio espectro de aplicaciones en áreas como la vigilancia, la robótica y la automatización industrial. En particular, la detección de peatones en entornos urbanos es de gran importancia para la seguridad vial y la prevención de accidentes.

En este sentido, una de las herramientas más utilizadas para la detección de objetos es YOLO (You Only Look Once), una arquitectura de red neuronal convolucional que realiza la detección de objetos en una sola pasada, lo que lo hace extremadamente rápido y eficiente. Recientemente, se ha lanzado YOLOv5, la última versión de esta arquitectura, que ofrece mejoras significativas en velocidad y precisión con respecto a las versiones anteriores.

En este trabajo, se aborda la configuración y reentrenamiento de YOLOv5 para la detección de peatones en imágenes, videos y capturas en tiempo real. Se describe el proceso de configuración y entrenamiento de la red, utilizando un conjunto de datos etiquetados que incluye imágenes de peatones en diferentes entornos urbanos. Además, se evalúa el rendimiento de la red en términos de precisión y velocidad de detección, comparándolo con otras arquitecturas de detección de objetos.

3.1. Modelos de YOLOv5

Ilustración 2 Imagen de modelos de YOLOv5

En YOLOv5, hay varios modelos diferentes según el tamaño y la complejidad de la red neuronal. Estos modelos se clasifican por letras, donde las letras representan la complejidad de la red y el número indica el tamaño de la imagen de entrada que la red puede procesar. A continuación, se describen brevemente los diferentes modelos de YOLOv5:

- YOLOv5s: Es el modelo más pequeño y liviano de YOLOv5. Tiene alrededor de 8,7 millones de parámetros y puede procesar imágenes de hasta 640x640 píxeles. Este modelo es adecuado para dispositivos con recursos limitados, como dispositivos móviles.

- YOLOv5m: Es un modelo mediano que tiene alrededor de 35,9 millones de parámetros y puede procesar imágenes de hasta 1024x1024 píxeles. Este modelo es adecuado para aplicaciones que requieren una mayor precisión en la detección de objetos, como la seguridad en el tráfico y la vigilancia de video.

- YOLOv5l: Es un modelo más grande que YOLOv5m, con alrededor de 85,5 millones de parámetros. Puede procesar imágenes de hasta 1280x1280 píxeles y tiene una mejor precisión en la detección de objetos en comparación con YOLOv5m.

- YOLOv5x: Es el modelo más grande y complejo de YOLOv5, con alrededor de 136,4 millones de parámetros. Puede procesar imágenes de hasta 640x640 píxeles con un batch size de 128 y 640x1280 píxeles con un batch size de 32. Este modelo es adecuado para aplicaciones que requieren una alta precisión en la detección de objetos, como la conducción autónoma.

En general, la elección del modelo dependerá del tipo de aplicación y la precisión requerida en la detección de objetos. Los modelos más pequeños son adecuados para dispositivos con recursos limitados, mientras que los modelos más grandes son adecuados para aplicaciones que requieren una mayor precisión en la detección de objetos.

3.2. Reentrenamiento de la red neuronal

La configuración de la red neuronal de YOLOv5 se puede llevar a cabo en los siguientes pasos:

- **Clonar el repositorio de YOLOv5 e instalación:** Para empezar, es necesario clonar el repositorio de YOLOv5 de GitHub en su máquina local. Esto se puede hacer mediante el comando git clone https://github.com/ultralytics/yolov5.git en la terminal.

```
git clone https://github.com/ultralytics/yolov5  # clone
cd yolov5
pip install -r requirements.txt  # install
```

Ilustración 3 Imagen de códigos de instalación Ultralytics

- **Configurar los datos:** Es importante tener un conjunto de datos etiquetados para entrenar el modelo. Los datos se deben organizar en una estructura de directorios específica, como se indica en la documentación de YOLOv5. Se deben proporcionar imágenes, etiquetas y un archivo YAML que describa las clases.

```
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/coco128  # dataset root dir
train: images/train2017  # train images (relative to 'path') 128 images
val: images/train2017  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes (80 COCO classes)
names:
  0: person
  1: bicycle
  2: car
  ...
  77: teddy bear
  78: hair drier
  79: toothbrush
```

Ilustración 4 Imagen de códigos de archivo YAML

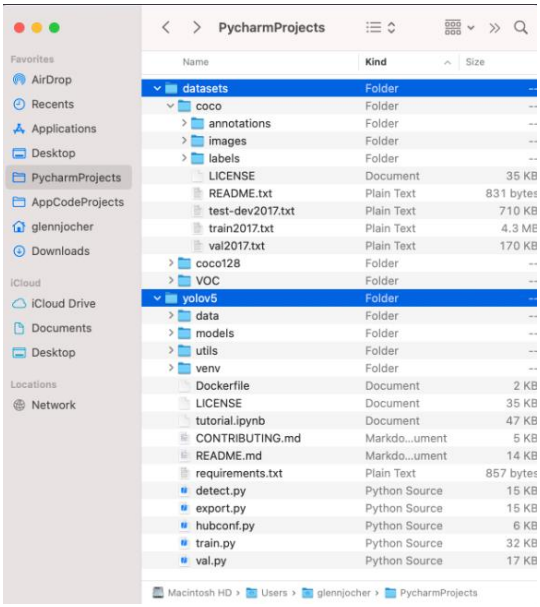

Ilustración 5 Imagen de estructura de carpetas

- **(Opcional) uso de wandb:** Wandb (Weights & Biases) es una herramienta de seguimiento y visualización de experimentos de aprendizaje automático en línea
    - **Crear una cuenta en Wandb:** Para comenzar a utilizar Wandb, es necesario registrarse en su sitio web e iniciar sesión.

    - **Instalar Wandb:** Después de crear una cuenta, debe instalar Wandb en su entorno de trabajo. Puede instalarlo usando pip, el administrador de paquetes de Python. Para instalar Wandb, simplemente ejecute el siguiente comando en una terminal:

      pip install wandb

- **Entrenar la red neuronal:** Una vez que se hayan configurado los datos y los hiperparámetros, se puede comenzar el entrenamiento de la red neuronal. Esto se puede hacer mediante el commando

```
python train.py --img 640 --batch 16 --epochs 100 --data path/to/data.yaml --cfg ./models/yolov5s.yaml --weights yolov5s.pt
```

Ilustración 6 Imagen de código para entrenar YOLOv5

Este comando inicia el proceso de entrenamiento con las configuraciones especificadas.

- **Evaluar el modelo:** Después de entrenar el modelo, es importante evaluar su rendimiento en un conjunto de datos de prueba.

- **Realizar inferencias:** Finalmente, se puede utilizar la red neuronal entrenada para hacer inferencias en nuevas imágenes o vídeos.

Siguiendo estos pasos, se puede configurar y entrenar una red neuronal YOLOv5 para la detección de objetos en imágenes y vídeos, puedes encontrar más información aquí https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

4. Evaluar desempeño

El desempeño de un modelo se evalúa para determinar su capacidad para generalizar y hacer predicciones precisas sobre datos nuevos, no vistos durante el entrenamiento. El objetivo de cualquier modelo de aprendizaje automático es obtener el mejor desempeño posible en los datos de prueba, lo que significa que el modelo debe ser capaz de generalizar bien y hacer predicciones precisas sobre datos nuevos.

La evaluación del desempeño del modelo es importante porque permite a los desarrolladores de modelos comparar diferentes modelos y técnicas de aprendizaje automático para determinar cuál es el mejor para una tarea en particular. También permite ajustar los hiperparámetros del modelo y mejorar su precisión y capacidad de generalización. Además, la evaluación del desempeño del modelo es esencial para determinar si el modelo está listo para su implementación en aplicaciones del mundo real, como la detección de objetos en imágenes, la clasificación de textos o la predicción de la demanda en el comercio minorista, entre otras.

4.1. Validación cruzada

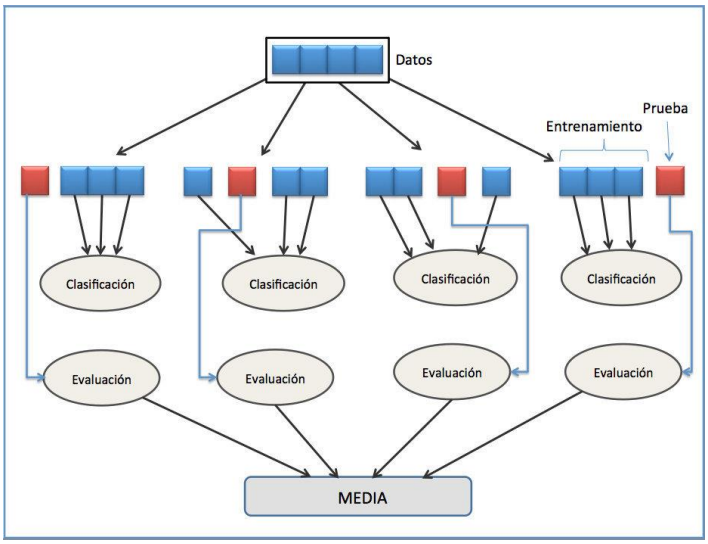

Ilustración 7 Imagen de validación cruzada

La validación cruzada es una técnica utilizada en el aprendizaje automático para evaluar el rendimiento de un modelo. La idea es dividir los datos en conjuntos de entrenamiento y prueba y entrenar el modelo con el conjunto de entrenamiento y evaluarlo con el conjunto de prueba. Sin embargo, si se utiliza solo un conjunto de entrenamiento y prueba, es posible que el modelo esté sobre ajustando los datos de entrenamiento y no sea capaz de generalizar bien a nuevos datos.

La validación cruzada resuelve este problema al dividir los datos en varios subconjuntos de entrenamiento y prueba, y realizar varias rondas de entrenamiento y evaluación del modelo. En una validación cruzada K-fold, por ejemplo, se dividen los datos en K subconjuntos. Luego, se entrena el modelo en K-1 subconjuntos y se evalúa en el subconjunto restante. Este proceso se repite K veces, de modo que cada subconjunto se utiliza una vez como conjunto de prueba. Finalmente, se calcula el promedio de las métricas de evaluación obtenidas en las K iteraciones.

La fórmula de la validación cruzada K-fold es la siguiente:

$$CV = (1/K) * \sum (i=1 \text{ hasta } K) \text{ evaluar}(i)$$

Donde CV es la métrica de evaluación promedio, K es el número de subconjuntos en los que se divide el conjunto de datos y evaluar(i) es la métrica de evaluación obtenida en la iteración i.

La validación cruzada es una técnica útil para evaluar la capacidad de generalización de un modelo, ya que permite obtener una estimación más precisa del desempeño del modelo en nuevos datos. Además, es una técnica importante para ajustar los hiperparámetros del modelo y evitar el sobreajuste a los datos de entrenamiento.

4.2. Matriz de confusión

La matriz de confusión es una herramienta que se utiliza para evaluar el desempeño de un modelo de clasificación. La matriz muestra el número de predicciones que fueron clasificadas correctamente y las que fueron clasificadas incorrectamente.

La matriz de confusión se construye a partir de los resultados de la evaluación del modelo en el conjunto de prueba, comparando las etiquetas reales de las muestras con las etiquetas predichas por el modelo. La matriz muestra cuatro valores principales:

- **Verdaderos positivos (TP):** corresponde al número de muestras positivas que fueron clasificadas correctamente como positivas por el modelo.
- **Falsos positivos (FP):** corresponde al número de muestras negativas que fueron clasificadas incorrectamente como positivas por el modelo.
- **Verdaderos negativos (TN):** corresponde al número de muestras negativas que fueron clasificadas correctamente como negativas por el modelo.
- **Falsos negativos (FN):** corresponde al número de muestras positivas que fueron clasificadas incorrectamente como negativas por el modelo.

La fórmula de la matriz de confusión es la siguiente

$$[[\ TP,\ FP],$$
$$[FN,\ TN]]$$

Donde TP es el número de verdaderos positivos, FP es el número de falsos positivos, FN es el número de falsos negativos y TN es el número de verdaderos negativos.

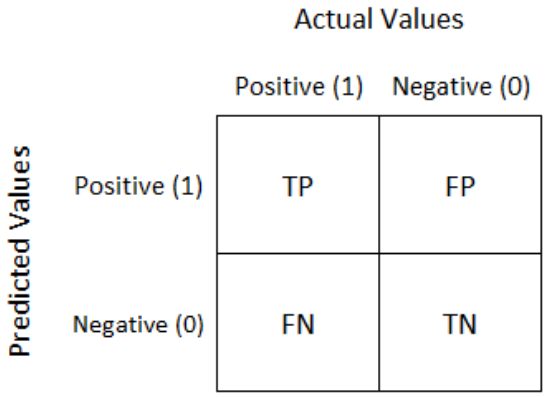

Ilustración 8 Imagen de una matriz de confusión

La matriz de confusión permite calcular diversas métricas de evaluación de la calidad del modelo, como la precisión, la sensibilidad y la especificidad. Estas métricas se utilizan para evaluar el desempeño del modelo en función de su capacidad para predecir correctamente los valores positivos y negativos en los datos de prueba.

4.3. Precisión

La precisión es una métrica de evaluación que mide la proporción de muestras positivas que fueron clasificadas correctamente por el modelo en relación con todas las muestras clasificadas como positivas por el modelo. Es decir, la precisión indica la capacidad del modelo para identificar correctamente las muestras positivas

$$Precisión = TP / (TP + FP)$$

Donde TP es el número de verdaderos positivos y FP es el número de falsos positivos. La precisión varía en un rango de 0 a 1, donde 1 indica una precisión perfecta, lo que significa que el modelo no cometió ningún error en la clasificación de las muestras positivas. Por otro lado, una precisión de 0 indica que todas las muestras clasificadas como positivas fueron incorrectas.

Es importante tener en cuenta que la precisión no es una métrica adecuada para evaluar modelos que tienen un desequilibrio de clases, es decir, cuando una clase tiene muchas más muestras que otra. En este caso, una alta precisión podría ser

engañosa, ya que el modelo puede tener una precisión alta simplemente porque clasifica incorrectamente pocas muestras de la clase minoritaria. En este caso, otras métricas, como la sensibilidad, la especificidad y la F1-score, pueden proporcionar una evaluación más completa del desempeño del modelo.

4.4. Recall

El recall, también conocido como sensibilidad o tasa de verdaderos positivos, es una métrica de evaluación que mide la capacidad del modelo para identificar correctamente las muestras positivas. El recall indica la proporción de muestras positivas que fueron clasificadas correctamente en relación con todas las muestras positivas presentes en el conjunto de datos.

$$Recall = TP / (TP + FN)$$

Donde TP es el número de verdaderos positivos y FN es el número de falsos negativos. El recall varía en un rango de 0 a 1, donde 1 indica un recall perfecto, lo que significa que el modelo es capaz de identificar correctamente todas las muestras positivas en el conjunto de datos. Por otro lado, un recall de 0 indica que el modelo no identificó correctamente ninguna de las muestras positivas.

Es importante tener en cuenta que el recall no tiene en cuenta las muestras negativas clasificadas correcta o incorrectamente por el modelo. Por lo tanto, el recall es una métrica útil para evaluar modelos que tienen como objetivo detectar todas las muestras positivas en un conjunto de datos, como en la detección de fraudes o enfermedades, pero no es adecuado para evaluar la calidad general de un modelo de clasificación.

4.5. F1-Score

La F1-score es una métrica de evaluación que combina la precisión y la sensibilidad de un modelo en una sola medida. La F1-score es útil cuando se tiene un conjunto de datos desequilibrado y se desea tener una medida más equilibrada del rendimiento del modelo.

$$F1\text{-}score = 2 * (precision * recall) / (precision + recall)$$

Donde precisión es la precisión y recall es la sensibilidad del modelo. La F1-score varía en un rango de 0 a 1, donde 1 indica una F1-score perfecta, lo que significa que el modelo es capaz de identificar correctamente todas las muestras positivas y negativas. Por otro lado, una F1-score de 0 indica que el modelo no es capaz de identificar correctamente ninguna de las muestras positivas o negativas.

La F1-score es una métrica importante para evaluar modelos que tienen un desequilibrio de clases, ya que da igual importancia a la precisión y la sensibilidad. Una alta F1-score indica que el modelo es capaz de identificar correctamente tanto las muestras positivas como las negativas.

4.6. Curva ROC

La curva ROC (Receiver Operating Characteristic) es una herramienta gráfica para evaluar el desempeño de un modelo de clasificación binaria. La curva ROC traza la tasa de verdaderos positivos (TPR) en el eje y, y la tasa de falsos positivos (FPR) en el eje x, para diferentes umbrales de clasificación.

La tasa de verdaderos positivos (TPR), también conocida como sensibilidad o recall, mide la proporción de muestras positivas que fueron clasificadas correctamente. La tasa de falsos positivos (FPR) mide la proporción de muestras negativas que fueron clasificadas incorrectamente como positivas.

$$TPR = TP / (TP + FN)$$
$$FPR = FP / (FP + TN)$$

Donde TP es el número de verdaderos positivos, FN es el número de falsos negativos, FP es el número de falsos positivos y TN es el número de verdaderos negativos.

La curva ROC es una curva que comienza en el punto (0,0) y termina en el punto (1,1). Un modelo perfecto se encuentra en la esquina superior izquierda de la curva ROC, lo que indica que tiene una TPR del 100% y un FPR del 0%. Un modelo que adivina al azar se encuentra en la línea diagonal de la curva ROC.

La medida del desempeño de un modelo en la curva ROC se realiza mediante el cálculo del área bajo la curva (AUC), que varía en un rango de 0 a 1. Un AUC de 1 indica un modelo perfecto, mientras que un AUC de 0.5 indica un modelo que adivina al azar.

4.7. Curva de perdida (Loss)

La curva de pérdida de error (loss) es una gráfica que muestra la disminución de la función de pérdida durante el entrenamiento de un modelo de aprendizaje automático. La función de pérdida mide la diferencia entre las predicciones del modelo y los valores reales, y el objetivo del entrenamiento es minimizar esta pérdida. La curva de pérdida de error muestra cómo cambia la función de pérdida a medida que el modelo se entrena iterativamente con los datos.

La fórmula para la función de pérdida varía según el problema y el tipo de modelo, pero en general, cuanto menor sea el valor de la función de pérdida, mejor será el modelo en la tarea de predicción. La curva de pérdida de error debe ser monótonamente decreciente, es decir, la pérdida debe disminuir a medida que el modelo se entrena.

La interpretación de la curva de pérdida de error es importante porque permite evaluar el rendimiento del modelo durante el entrenamiento. Si la curva de pérdida de error es demasiado alta o no disminuye con el tiempo, puede ser una señal de que el modelo no está aprendiendo correctamente o de que se necesita una configuración diferente. Por otro lado, si la curva de pérdida de error es baja y disminuye rápidamente, es una señal de que el modelo está aprendiendo bien y puede ser una buena opción para la tarea de predicción.

Se pueden encontrar diferentes tipos de pérdidas o loss en el entrenamiento de un modelo de detección de objetos. Cada una de estas pérdidas mide una cantidad específica de error en el proceso de entrenamiento. Entre las más comunes se encuentran:

- **cls_loss:** También conocida como pérdida de clasificación, se refiere a la penalización que se aplica al modelo cuando clasifica incorrectamente la presencia o ausencia de un objeto en una determinada región de la imagen. En otras palabras, mide la precisión de la clasificación de los objetos.
- **obj_loss:** También conocida como pérdida de objeto, se refiere a la penalización que se aplica al modelo cuando falla en la detección de un objeto que realmente está

presente en la imagen. Mide la capacidad del modelo para detectar la presencia de objetos.

- **box_loss:** También conocida como pérdida de regresión de caja, se refiere a la penalización que se aplica al modelo cuando las cajas delimitadoras predichas no se superponen bien con las cajas delimitadoras reales de los objetos en la imagen. Mide la precisión de la localización de los objetos en la imagen.

Cada una de estas pérdidas contribuye a la pérdida total o global del modelo. El objetivo del entrenamiento es minimizar esta pérdida global, lo que implica mejorar la precisión en la clasificación, detección y localización de objetos en una imagen.

5. Implementación

Se realizó la implementación del modelo de detección de peatones en un dispositivo de pequeño tamaño llamado Jetson Nano. La Jetson Nano es una placa de desarrollo de inteligencia artificial que ofrece un alto rendimiento informático para la ejecución de tareas de aprendizaje automático. Esta placa está diseñada específicamente para aplicaciones de inteligencia artificial en la periferia, lo que significa que es ideal para dispositivos pequeños y portátiles que necesitan realizar tareas de inferencia de aprendizaje automático en tiempo real.

Ilustración 9 Imagen de Jetson Nano

La Jetson Nano cuenta con un procesador de cuatro núcleos ARM Cortex-A57 y un procesador gráfico NVIDIA Maxwell de 128 núcleos, lo que la convierte en una de las placas

de desarrollo más potentes del mercado en términos de rendimiento de inteligencia artificial por vatio. Además, tiene 4 GB de memoria RAM, lo que le permite manejar cargas de trabajo pesadas sin problemas.

La Jetson Nano es compatible con varias bibliotecas y marcos de trabajo de aprendizaje automático, como TensorFlow, PyTorch, Caffe y MXNet, lo que la hace muy versátil en términos de su capacidad para ejecutar modelos de aprendizaje automático.

5.1. Configuración y reentrenamiento

Se realizó la configuración del ambiente de trabajo dentro de la Jetson Nano, utilizando Python3.6.9 y una versión de Torch 1.7.0. Sin embargo, cabe destacar que esta configuración no es recomendada para el uso de YOLOv5. Al realizar pruebas con los mismos pesos obtenidos en el entrenamiento original, se generaron problemas, ya que la red no detectaba a ninguna persona ni las demarcaba adecuadamente.

Ante esto, se procedió a realizar un nuevo entrenamiento dentro del servidor, pero con la configuración adecuada para poder utilizar los pesos en la Jetson Nano. De esta forma, se garantiza una adecuada detección de peatones en tiempo real en el dispositivo.

5.2. Procesamiento

```python
#!/usr/bin/python
#coding=utf-8
import cv2
import imutils
import numpy as np
import torch
from tqdm import tqdm
from jetcam.csi_camera import CSICamera

model = torch.hub.load('ultralytics/yolov5', 'custom', path="/media/tiny/venvs/yolo_env/peatones/best/bestJetson.pt",force_reload=True)
cap = CSICamera(width=1080, height=720, capture_width=1080, capture_height=720, capture_fps=30)
#cap = cv2.VideoCapture("/media/tiny/venvs/yolo_env/UNAM.mp4")
out = cv2.VideoWriter('PruebaJetsonYolo.avi',cv2.VideoWriter_fourcc('M','J','P','G'),20 , (1080,720))
n, nframes = 0, 300
for _ in tqdm(range(nframes), ncols=100):
    frame = cap.read()
    #frame = cv2.resize(frame,(1080,720))
    # Detectar personas en el marco
    detect = model(frame)
    #out.write()
    out.write(np.squeeze(detect.render()))
    if cv2.waitKey(1) == 27: break
out.release()
```

Ilustración 10 Imagen de código para el procesamiento

Este código utiliza el modelo reentrenado YOLOv5 para detectar personas en tiempo real a través de una cámara CSI y escribir los resultados en un archivo de video de salida.

- Se importan los módulos necesarios para el funcionamiento del programa: cv2 para manejar imágenes y video, imutils para algunas utilidades en procesamiento de imágenes, numpy para trabajar con matrices, torch para utilizar el modelo reentrenado YOLOv5, y tqdm para mostrar una barra de progreso en el proceso de detección.
- En la siguiente línea se carga el modelo reentrenado YOLOv5 personalizado, especificando su ruta de almacenamiento y forzando su recarga si es necesario.
- Se define la cámara CSI como fuente de video, con una resolución de 1080x720 y una tasa de captura de 30 fps.
- A continuación, se define el objeto VideoWriter para escribir el video de salida, especificando el nombre del archivo, el codec utilizado, la tasa de frames y la resolución del video.
- En el bucle for se establece el número de frames a capturar y se realiza la detección de personas en cada uno de ellos.
- En cada iteración del bucle, se lee un frame de la cámara y se detectan las personas en él utilizando el modelo reentrenado YOLOv5.
- Luego, se utiliza el método "render()" del objeto detect para obtener una imagen con las detecciones hechas por el modelo y se escribe en el archivo de salida.
- Si se presiona la tecla "ESC" se sale del bucle y se cierra el archivo de salida.

Este código puede ser utilizado también para la captura en videos, solo se debe des comentar las líneas de código que se encuentras comentadas y ajustar los parámetros según la necesidad.

6. Resultados

6.1. Primer entrenamiento

Durante la primera ejecución que fue la implementación los requisitos exigidos por YOLOv5 para el entrenamiento obtuvimos los siguientes resultados

best	
epoch	498
mAP_0.5	0.9724073353004202
mAP_0.5:0.95	0.8706105182851365
precision	0.9025954793580416
recall	0.9670779421476632
F1	0.9336

Ilustración 11 Imagen de pruebas con la red entrenada

Se puede notar que los resultados obtenidos por la red neuronal en la detección de peatones fueron un éxito, dado que se logró una precisión de 0.90 y un recall de 0.96.

La precisión indica la proporción de resultados positivos verdaderos, Es decir, se logró que el 90% de las detecciones marcadas como peatones por la red neuronal fueran realmente peatones. Por otro lado, el recall indica la proporción de resultados positivos verdaderos En este caso, se logró que la red neuronal detectara el 96% de los peatones presentes en la imagen.

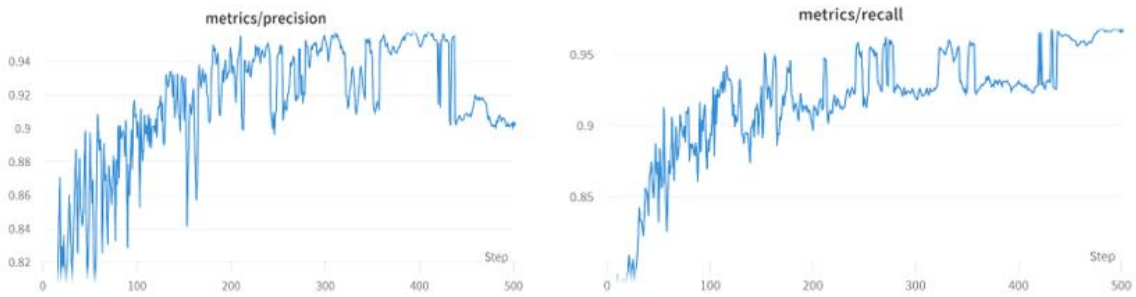

Ilustración 12 Imagen de las métricas de precision y recall durante el entrenamiento

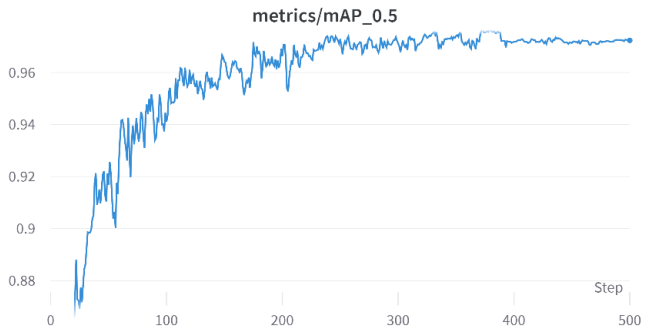

Ilustración 13 Imagen de las métricas de Iou=0.5 durante el entrenamiento

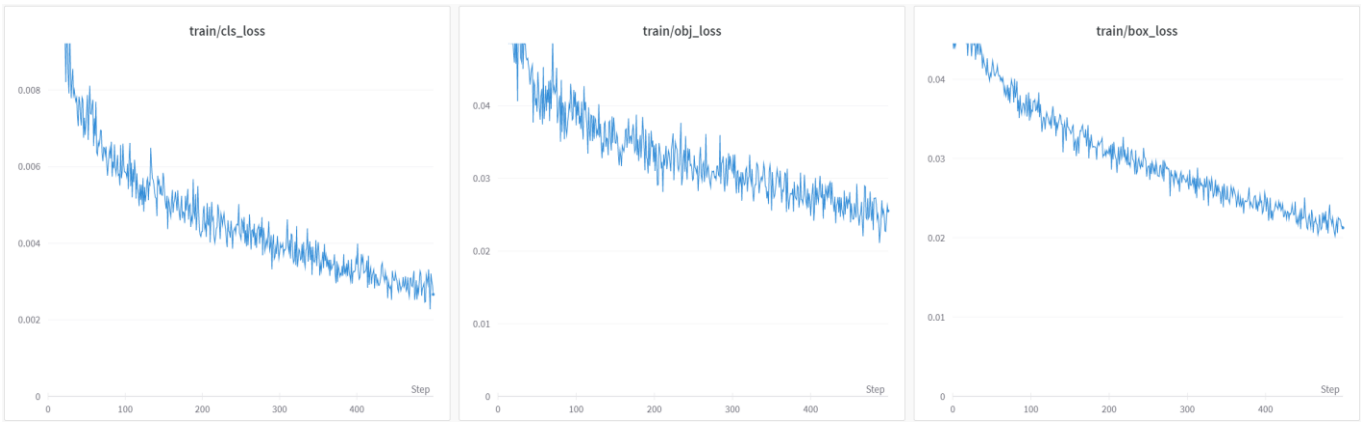

Ilustración 14 Imagen de las métricas de los diferentes loss durante el entrenamiento

Los valores de las diferentes métricas de loss obtenidas durante el entrenamiento del modelo no son muy altos, lo que indica que el modelo está aprendiendo bien y que el proceso de entrenamiento está funcionando correctamente.

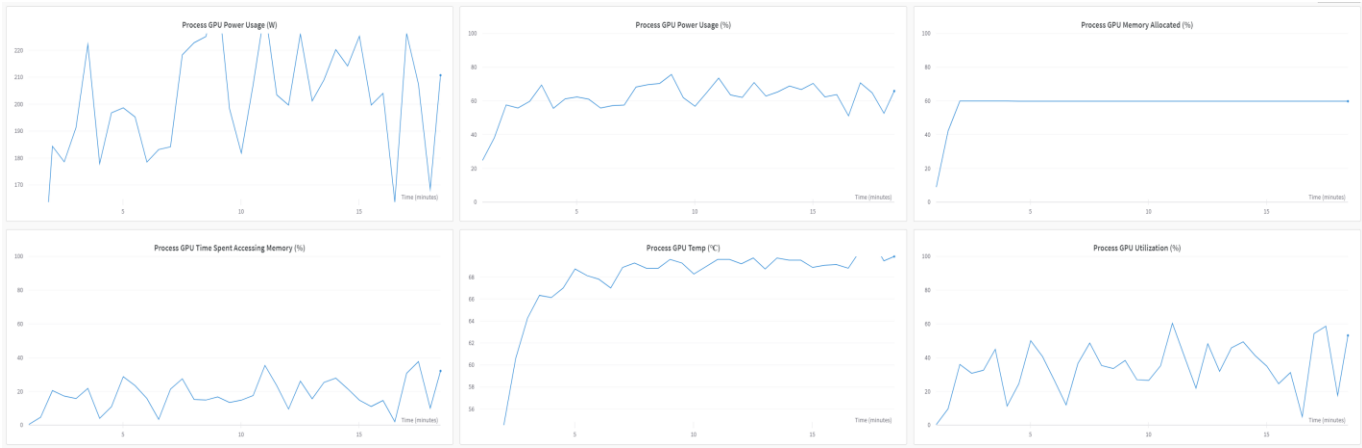

Ilustración 15 Imagen de las métricas uso de la GPU

Server

What makes this run special?

Privacy	PRIVATE
Tags	+
Author	alejoruiz
State	finished
Start time	March 10th, 2023 at 8:18:33 pm
Duration	18m 22s
Run path	alejoruiz/YOLOv5/bvmc48vs
Hostname	Precision-3650
OS	Linux-5.15.0-46-generic-x86_64-with-glibc2.29
Python version	3.8.10
Python executable	/home/est1/venvs/yolo_env/bin/python
Git repository	`git clone https://github.com/ultralytics/yolov5`
Git state	`git checkout -b "Server" 5543b89466d072a9f8f2e31f8257a1ccc7f588e9`
Command	`train.py --img 640 --batch 16 --epochs 500 --data /home/est1/venvs/yolo_env/yolov5/data/coco128.yaml --weights yolov5s.pt --cache ram`
System Hardware	CPU count 10
	GPU count 1
	GPU type NVIDIA GeForce RTX 3080
W&B CLI Version	0.13.11
Job Type	Training

Ilustración 16 Imagen del resumen de la ejecución

Ilustración 17 Imagen de pruebas con la red entrenada

Estos resultados son muy satisfactorios, ya que demuestran la capacidad de la red neuronal para aprender de manera efectiva y generalizar la detección de peatones. Además, son una indicación de que la red neuronal puede ser utilizada en una amplia variedad de aplicaciones de detección de peatones en tiempo real.

6.2. Jetson Nano

best	
epoch	96
mAP_0.5	0.9601183659085724
mAP_0.5:0.95	0.7848632744836562
precision	0.9173320340744112
recall	0.9127361492319864

Ilustración 18 Imagen de pruebas con la red entrenada

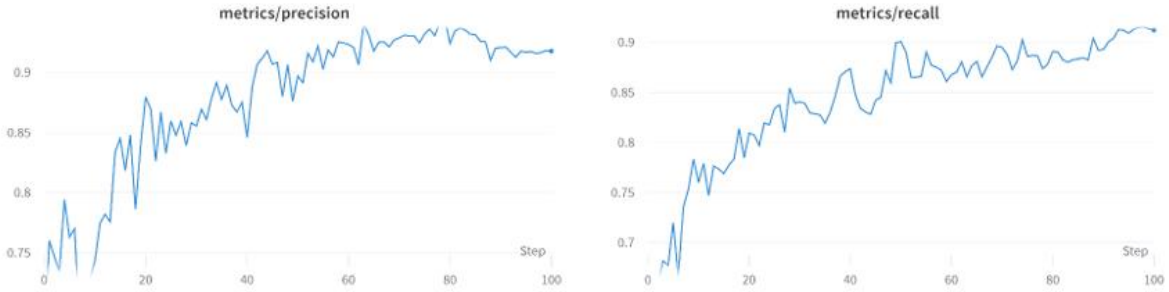

Ilustración 19 Imagen de las métricas de precision y recall durante el entrenamiento para la Jetson Nano

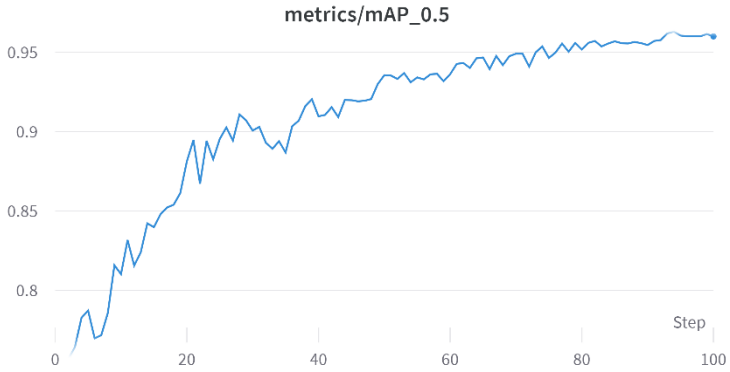

Ilustración 20 Imagen de las métricas de Iou=0.5 durante el entrenamiento para la Jetson Nano

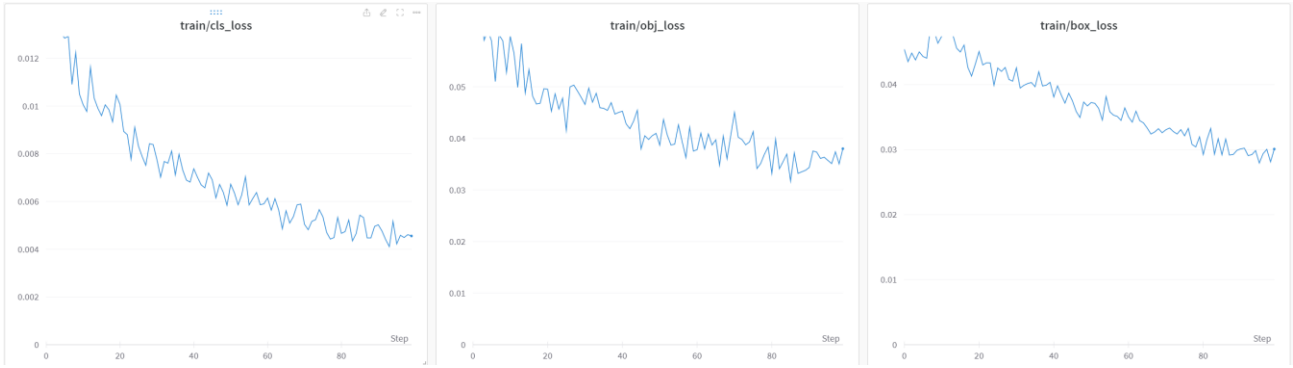

Ilustración 21 Imagen de las métricas de los diferentes loss durante el entrenamiento para la Jetson Nano

Notamos que el rendimiento de la red neuronal fue diferente en comparación con los resultados obtenidos en la etapa de entrenamiento. Aunque los valores de los loss fueron mínimos en comparación con los obtenidos en el entrenamiento anterior, siguen siendo valores muy buenos y aceptables.

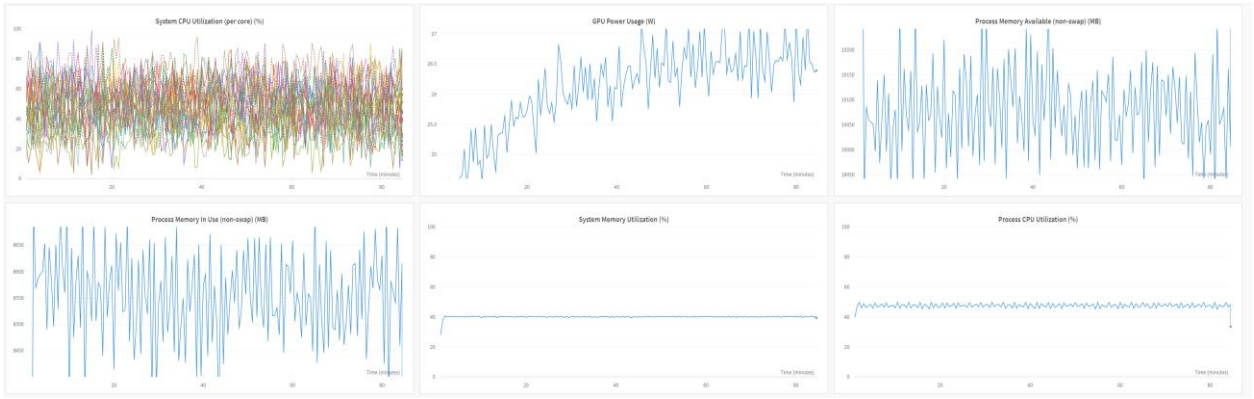

Ilustración 22 Imagen de las métricas de uso del sistema para la Jetson Nano

Se utilizó la CPU del servidor para realizar el entrenamiento del modelo para la Jetson Nano, ya que no se logró configurar correctamente el ambiente para utilizar la GPU. A pesar de esto, los resultados obtenidos fueron satisfactorios y se logró entrenar el modelo con éxito. Es importante mencionar que el proceso de entrenamiento podría haber sido más rápido si se hubiera utilizado la GPU, pero debido a las circunstancias, se optó por utilizar la CPU. En resumen, aunque no se pudo utilizar la GPU, se logró completar el entrenamiento del modelo y obtener resultados satisfactorios.

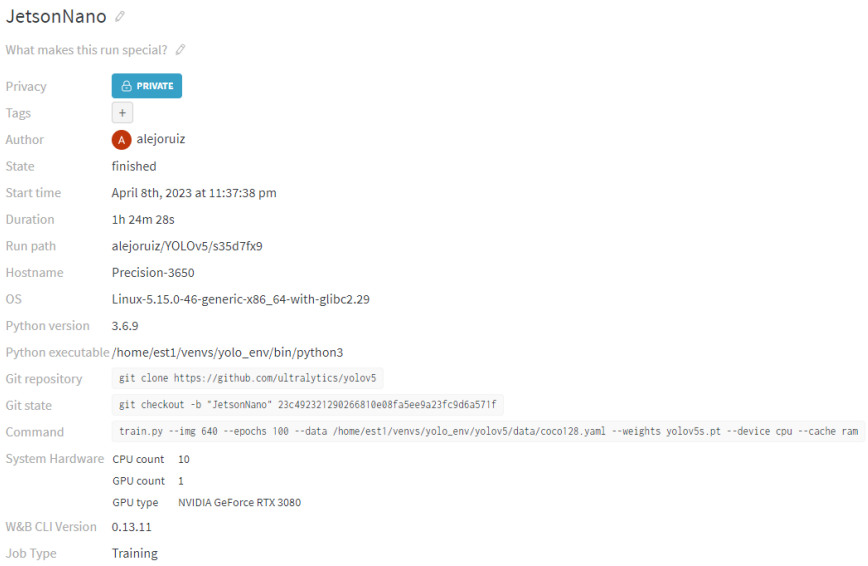

Ilustración 23 Imagen del resumen de ejecución para la Jetson Nano

Se realizaron ajustes en los parámetros del entrenamiento para evitar largos tiempos de ejecución, en comparación con la ejecución anterior. A pesar de ello, los resultados obtenidos fueron bastante buenos y la red neuronal fue capaz de detectar a las personas de manera efectiva en las imágenes de prueba. En general, se puede decir que la calidad de los resultados es aceptable y cumplen con el objetivo de detección de peatones.

Ilustración 24 Imagen de pruebas con la red entrenada para la Jetson Nano

Al probar la red con imágenes diferentes a las del conjunto de datos, se pudo observar que la detección de personas fue muy buena. La red fue capaz de detectar a las personas en diferentes posturas y posiciones en la imagen, lo cual indica que la red entrenó muy bien y puede generalizar bien a nuevas imágenes.

Ilustración 25Imagen ilustrativa desde posición cenital

Pero al probar con vistas desde un punto cenital presento desafíos significativos, ya que los modelos de detección no están entrenados específicamente para esta perspectiva. La detección de personas se ha desarrollado principalmente para capturar imágenes en ángulos horizontales o ligeramente inclinados, lo que dificulta la identificación precisa desde una vista aérea.

Para este problema se recomienda tener un dataset con imágenes etiquetadas desde este ángulo y procesarla con la red y evaluar el desempeño

Se pueden observar los videos tanto de las grabaciones en tiempo real como el procesamiento del video en el siguiente enlace

7. Conclusiones

Después de haber llevado a cabo el proyecto de detección de peatones con YOLOv5, se pueden destacar las siguientes conclusiones:

En primer lugar, se pudo demostrar que YOLOv5 es una arquitectura de red neuronal convolucional muy eficiente para la detección de peatones en tiempo real. A través del reentrenamiento del modelo con el conjunto de datos COCO y la evaluación del rendimiento en diferentes escenarios, se logró obtener una precisión aceptable en la detección de peatones con una tasa de error relativamente baja.

En segundo lugar, se demostró que la elección del hardware puede influir significativamente en el rendimiento de la red neuronal. La implementación del modelo en un ordenador de inteligencia artificial pequeño (Jetson Nano) requería un reajuste en la configuración y entrenamiento para obtener resultados óptimos, debido a las limitaciones de hardware.

Por último, se puede concluir que la detección de peatones tiene aplicaciones muy amplias en la seguridad de las personas, ya sea en la vigilancia de calles, la prevención de accidentes de tráfico o en la gestión de emergencias. Por lo tanto, se pueden explorar otras posibles mejoras en la precisión del modelo y en la eficiencia del hardware para lograr una implementación más amplia y efectiva en la detección de peatones.

8. Recomendaciones

- **Utilizar la versión recomendada de Python y Torch:** Asegurarse de utilizar la versión de Python y Torch recomendada para YOLOv5. Las versiones incompatibles pueden causar problemas en el proceso de entrenamiento y detección.
- **Ajustar los hiperparámetros adecuados:** Los hiperparámetros como el tamaño del batch, el learning rate y el número de epochs son fundamentales en el proceso de entrenamiento. Es importante ajustarlos adecuadamente para obtener los mejores resultados posibles.
- **Aumentar el tamaño del dataset:** Aumentar el tamaño del dataset con imágenes de peatones puede mejorar la precisión de la detección.
- **Evaluar los resultados de forma sistemática:** Evaluar los resultados de detección de forma sistemática y detallada, utilizando métricas como la precisión, el recall y el F1-score.
- **Utilizar múltiples modelos de detección:** Probar diferentes arquitecturas de redes neuronales, como RCNN, Faster R-CNN, Mask R-CNN y YOLO, para determinar cuál es el mejor modelo para la tarea de detección de peatones.
- **Optimizar el hardware:** Optimizar el hardware utilizado para entrenar y ejecutar el modelo, como la tarjeta gráfica (GPU) o la unidad de procesamiento tensorial (TPU), para acelerar el proceso y mejorar la precisión.
- **Documentar el proceso de manera adecuada:** Registrar detalladamente todos los pasos realizados durante el proceso de entrenamiento y detección, incluyendo los hiperparámetros utilizados, el tamaño del dataset, la versión de los paquetes de software y hardware, y los resultados obtenidos. Esto permitirá replicar los experimentos y comparar los resultados en el futuro.

Referencias bibliográficas

Dalal, N. &. (2005). *Histograms of oriented gradients for human detection.*

Farhadi., J. R. (s.f.). *YOLOv3: An Incremental Improvement*. Obtenido de Yolo: https://pjreddie.com/darknet/yolo/

Girshick, R. (. (2015). Fast R-CNN. Proceedings.

Girshick, R. D. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation.

He, K. G. (2017). Mask R-CNN. *IEEE International Conference on Computer Vision, 2980-2988. doi: 10.1109/ICCV.2017.322*.

Kaiming He, G. G. (2017). Mask R-CNN .

Kohavi, R. (1995). *A study of cross-validation and bootstrap for accuracy estimation and model selection.*

Krizhevsky, A. S. (2012). *Imagenet classification with deep convolutional neural networks.*

Lin, T. Y. (2014). Microsoft COCO: Common Objects in Context. *European Conference on Computer Vision, 740-755. doi: 10.1007/978-3-319-10602-1_48*.

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of machine learning technologies,*.

Pyo, Y. (08 de 07 de 2015). *SlideShare*. Obtenido de https://www.slideshare.net/yoonseokpyo/20150708-ros-seminarinbusankorea

Redmon, J. D. (2016). You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition, 779-788. doi: 10.1109/CVPR.2016.91*.

Ren, S. H. (2016). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Conference on Computer Vision and Pattern Recognition, 77-85. doi: 10.1109/CVPR.2016.19*.

Viola, P. &. (2004). Robust real-time face detection. *International journal of computer vision*.

11. Anexos

Anexo a Código reconocimientoDePeatones

```
#!/usr/bin/python

#coding=utf-8

import cv2

import imutils

import numpy as np

import torch

from tqdm import tqdm

from jetcam.csi_camera import CSICamera

model = torch.hub.load('ultralytics/yolov5', 'custom',
path="/media/tiny/venvs/yolo_env/peatones/best/bestJetson.pt",force_reload=True)

cap = CSICamera(width=1080, height=720, capture_width=1080, capture_height=720, capture_fps=30)

#cap = cv2.VideoCapture("/media/tiny/venvs/yolo_env/UNAM.mp4")

out = cv2.VideoWriter('PruebaJetsonYolo.avi',cv2.VideoWriter_fourcc('M','J','P','G'),20 , (1080,720))

n, nframes = 0, 300

for _ in tqdm(range(nframes), ncols=100):

   frame = cap.read()

   # Detectar personas en el marco

   detect = model(frame)

   out.write(np.squeeze(detect.render()))

   if cv2.waitKey(1) == 27: break

out.release()
```

Anexo b Pesos

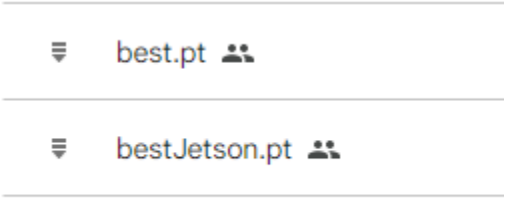

Ilustración 26 Imagen de los pesos

Anexo c Pasos de instalación para entrenar para el servidor

git clone https://github.com/ultralytics/yolov5 # clone

cd yolov5

pip install -r requirements_ubuntu.txt # install

Anexo d Pasos de instalación para entrenar la Jetson Nano

git clone https://github.com/ultralytics/yolov5 # clone

cd yolov5

pip install -r requirements_jetson.txt # install

Anexo e requirements_ubuntu.txt

YOLOv5 requirements server

Usage: pip install -r requirements_server.txt

Base ----------

gitpython>=3.1.30

matplotlib>=3.3

numpy>=1.18.5

opencv-python>=4.1.1

Pillow>=7.1.2

psutil # system resources

PyYAML>=5.3.1

```
requests>=2.23.0
scipy>=1.4.1
thop>=0.1.1  # FLOPs computation
torch>=1.7.0  # see https://pytorch.org/get-started/locally (recommended)
torchvision>=0.8.1
tqdm>=4.64.0
ultralytics>=8.0.100
# protobuf<=3.20.1  # https://github.com/ultralytics/yolov5/issues/8012
# Logging ---------------------------------------------------------------------
# tensorboard>=2.4.1
# clearml>=1.2.0
# comet
# Plotting --------------------------------------------------------------------
pandas>=1.1.4
seaborn>=0.11.0
# Export ----------------------------------------------------------------------
# coremltools>=6.0  # CoreML export
# onnx>=1.10.0  # ONNX export
# onnx-simplifier>=0.4.1  # ONNX simplifier
# nvidia-pyindex  # TensorRT export
# nvidia-tensorrt  # TensorRT export
# scikit-learn<=1.1.2  # CoreML quantization
# tensorflow>=2.4.0  # TF exports (-cpu, -aarch64, -macos)
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export
# Deploy ----------------------------------------------------------------------
setuptools>=65.5.1 # Snyk vulnerability fix
# tritonclient[all]~=2.24.0
# Extras ----------------------------------------------------------------------
# ipython  # interactive notebook
# mss  # screenshots
```

albumentations>=1.0.3

pycocotools>=2.0.6 # COCO mAP

Anexo f requirements_jetson.txt

YOLOv5 requirements jetson

Usage: pip install -r requirements_jetson.txt

Base ----------

gitpython>=3.1.30

matplotlib>=3.3

numpy>=1.18.5

opencv-python>=4.1.1

Pillow>=7.1.2

psutil # system resources

PyYAML>=5.3.1

requests>=2.23.0

scipy>=1.4.1

thop>=0.1.1 # FLOPs computation

torch==1.7.0 # see https://pytorch.org/get-started/locally (recommended)

torchvision==0.8.1

tqdm>=4.64.0

ultralytics>=8.0.100

protobuf<=3.20.1 # https://github.com/ultralytics/yolov5/issues/8012

Logging ----------

tensorboard>=2.4.1

clearml>=1.2.0

comet

Plotting ----------

pandas>=1.1.4

seaborn>=0.11.0

Export ----------

```
# coremltools>=6.0  # CoreML export
# onnx>=1.10.0  # ONNX export
# onnx-simplifier>=0.4.1  # ONNX simplifier
# nvidia-pyindex  # TensorRT export
# nvidia-tensorrt  # TensorRT export
# scikit-learn<=1.1.2  # CoreML quantization
# tensorflow>=2.4.0  # TF exports (-cpu, -aarch64, -macos)
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export
# Deploy ------------------------------------------------------------------------
setuptools>=65.5.1 # Snyk vulnerability fix
# tritonclient[all]~=2.24.0
# Extras ------------------------------------------------------------------------
# ipython  # interactive notebook
# mss  # screenshots
# albumentations>=1.0.3
# pycocotools>=2.0.6  # COCO mAP
```

Anexo g Repositorio GitHub

https://github.com/AlejoRuiz/ReconocimientoDePersonas