Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Inteligencia Artificial - IC6200

JetBot a self driven Car

Estudiantes:

Rebeca Bolaños Arce
Ignacio Cruz Montero
Jaffette Solano Arias
Sarah Uriarte Porras

Profesor:

Rogelio González Quirós

Campus Tecnológico Local San Carlos, 2020

Índice general

Índice de figuras

Introducción

Actualmente, hay una gran cantidad de soluciones de software basadas en la implementación de procesamiento de imágenes para solventar problemas cotidianos que quizás anteriormente no funcionaban. Esto ha abierto infinitas posibilidades para aplicar diferentes análisis de imágenes. Entre estas aplicaciones se encuentra la detección de objetos en una imagen, lo que aumenta la capacidad operativa de los robots autónomos y semi autónomos. Los robots reales requieren interacción con el entorno en el que se desarrollan para adaptar sus acciones en busca de un rendimiento óptimo. Tal es el caso del robot que desarrollamos, Jetbot, el cual deberá ser capaz de conducir sobre un circuito de carretera dado por el equipo de desarrollo. Para poder movilizarse sobre el circuito deberá ser capaz de reconocer no solo la carretera sino una serie de señales de transito como el ceda, alto, reductores de velocidad y señales de virar a la derecha e izquierda. Además, deberá ser capaz de reconocer un peatón y otros vehículos dentro del circuito.

Problema

El problema abarca la simulación de un carro autónomo a escala, que sea capaz de avanzar sobre una carretera, y durante el recorrido sobre esta logre detectar posibles peatones, autos y señalización que debe de acatar conforme avance sobre el camino. Además, el robot deberá completar el circuito de trayecto sin intervención humana, deberá ser capaz de iniciar y detenerse por sí mismo.

El robot deberá tomar en cuenta las siguientes consideraciones:

1. Se requiere que el robot siga la carretera y evite salirse de esta.
2. Se requiere que el robot detecte y acate todas las señales que aparezcan durante su trayecto de conducción.
3. El robot deberá ser capaz de reconocer los posibles peatones y detenerse para así evitar colisionar con estos.
4. El robot deberá ser capaz de reconocer otros autos en la vía para evitar colisionar con ellos, lo que simularía la prevención de un accidente de tránsito.
5. El robot no deberá iniciar o detenerse hasta que detecte y procese las señales con las que fue entrenado para este fin.

El circuito deberá ser diseñado por los estudiantes y deberá tener las dimensiones correctas para que el robot pueda circular sobre el camino sin ningún problema.

Solución planteada

Solución planteada para el movimiento del robot sobre la carretera

En cuanto a la solución planteada para el movimiento del Jetbot en la carretera, lo primero que se realizó fue en la implementación de una pista para ser impresa, con esto se logró tener el plano por el cual el Jetbot realizará su recorrido, seguidamente, se inicio con la investigación de cómo realizar el seguimiento de líneas del Jetbot dentro de la carretera, para lograr implementar este seguimiento se utilizó el repositorio Road_Following, este repositorio cuenta con notebooks, donde uno de estos se encarga de la captura de datos por medio de imágenes mediante la cámara del Jetbot en tiempo real(Agregaría la manera de tomar las fotos), además cuenta con otro notebook que se encarga del entrenamiento utilizando el set de datos capturado mediante el notebook anteriormente mencionado que se encargaba de la toma de datos, por último se encuentra el notebook llamado live_demo que se encarga del movimiento del Jetbot mediante el seguimiento de las líneas utilizando el modelo previamente entrenado con el set de datos capturado. Estos notebooks utilizan librerías como PyTorch y ResNet18 que es una red neuronal convolucional para el entrenamiento del modelo. Es importante mencionar que en el notebook de entrenamiento se utiliza Transfer Learning que es una técnica de machine learning para el entrenamiento automático, esto se utiliza para acelerar el proceso de entrenamiento y para mejorar el rendimiento del modelo de aprendizaje.

Durante el proceso de entrenamiento encontramos diferentes problemas con la precisión sin importar los cambios en la velocidad y el procesamiento de imágenes cuando el Jetbot estaba ejecutándose con el modelo entrenado. Por lo cual se realizó una búsqueda de estos conflicto que teníamos y encontramos que mediante la agregación de valores alpha al motor del robot nos podrían generar mejor precisión para la ejecución del Jetbot, por lo que en

la implementación tuvimos que editar las variables para nuestro robot, donde encontramos que los valores correctos para las variables a editar fueron las siguientes, para el speed_gain el valor de 0.14, el steering_gain de 0.3, alpha_left de 2.5 y el alpha_right de 1.5. Con estas se dio la mejor reacción para el movimiento y el uso del modelo.

Solución planteada para el análisis de imágenes

Se plantea la utilización de Tiny-yolov3 y darknet, además de la creación de un dataset personalizado, con imágenes similares al ambiente en el que se movilizará el robot.

El primer paso para la construcción del modelo consiste en la creación de un dataset personalizado, para el cual se tomaron y etiquetaron un promedio de doscientas treinta imágenes por clase. Las clases definidas fueron las siguientes, virar a la derecha, virar a la izquierda, alto, ceda, límite de velocidad treinta, autos, peatones, incio del recorrido y fin del recorrido. Teniendo un total de nueve clases.

Para el desarrollo de un modelo que le permita al robot la detección de señales, peatones y vehículos en la carretera, utilizamos una red convolucional, específicamente Tiny-yolov3 desarrollado por Joseph Redmon [5]. La razón por la que utilizamos esta red convolutiva se debe a su rapidez, precisión y fácil entrenamiento. Este algoritmo funciona de manera supervisada en su fase de entrenamiento.

Para poder realizar un entrenamiento personalizado de los datos, solo se deberán implementar un archivo de configuración que se ajuste a la cantidad de clases a entrenar, un archivo para las clases y eventualmente las imágenes con su propia etiqueta correspondiente a los "Bounding Boxes".

A continuación en la figura 1 se puede apreciar el ciclo que se debe seguir en cuanto a la solución planteada para la ejecución de ambos modelos, en esta figura se divide en sectores pasando desde la obtención de datos, entrenamiento y concluyendo con la ejecución del modelo, aclarando el orden de los pasos a seguir.

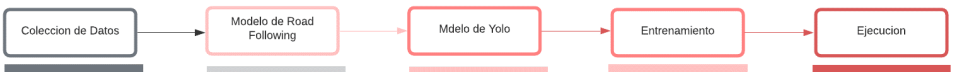

Figura 1: Ciclo de la Solución. Elaboración propia.

Para la ejecución de esta soluciones se deberá seguir esta linea de tiempo planteada en la figura anterior. Primeramente en lo que sera la recolección de datos, se deberá realizar capturas de imágenes utilizando como recurso la cámara del Jetbot, además, se tiene que colocar en diferentes posiciones y lugares con el fin de que se obtengan diferentes ángulos de imágenes para su respectivo entrenamiento.

Además, se deberá tener en cuenta que durante la obtención de fotos para el modelo de Road_following que es utilizado para el reconocimiento de la carretera, se tendrá que realizar un entrenamiento de su movimiento por cada imagen que es tomada, en este caso, cuando se toma la imagen, el robot en cada imagen mostrar dos puntos de referencia con valores 'x' y 'y' cada punto, donde el punto rojo hará referencia a donde se encuentra actualmente el robot y un punto verde que es el que se tiene que asignar hacia donde se quiere mover el robot, es aquí donde se usan los valores 'x' y 'y' para posicionar este punto verde en la imagen que se identifique, después de haber realizado estas especificaciones por imagen, se podrá continuar con el entrenamiento de la red neuronal preentrenada donde se debera enviar el set de datos capturado y con las indicaciones de lo que se deberá de hacer por cada frame para ser entrenado mediante la CNN Resnet18.

Por último, ya creado el modelo se proseguirá con implementar la ejecución en tiempo real del Jetbot, donde se debera cargar el modelo, ya cargado en el notebook, se tendrá que asignar los valores a las distintas variables que se mencionaron en la descripción del modelo anteriormente. Con todos estos paso ya realizados el Jetbot se debería estar moviendo por la carretera, identificando los frames y realizando los movimientos correspondientes a cada frame visualizado.

Modelo de inteligencia artificial creado

Modelo creado para el movimiento del robot sobre la carretera y el modo libre

Una de las partes más importantes antes de entrar en la explicación del modelo creado para el movimiento del robot sobre la carretera, es el set de datos. Se realizó una toma de 350 fotografías, esta captura de imágenes fue hecha desde su propio hardware, es decir, con su propia cámara (método explicado anteriormente), el 90 % de este set de datos se utilizó para entrenamiento y el 10 % para validación.

Por otra, parte para la sección de movimiento libre se entrenó un modelo con dos clases, una llamada 'free' que indica que el robot puede caminar y una llamada 'blocked' que le permite al robot detenerse y cambiar de ruta cuando tiene un obstaculo al frente. El tamaño del set de datos fue de 226 fotos, es decir 113 para cada clase. Este se divide arbitrariamente en una cantidad de datos de prueba de 50 imágenes y las restantes para el entrenamiento.

Además, para llevar a cabo el modelo en ambos casos se está utilizando deep learning con el framework pytorch. También se utiliza una arquitectura resnet-18, la cual es una red convolucional con una profundidad de 16 capas convolucionales y 2 capas de submuestreo [4]. Seguido a esto, bajo el concepto de aprendizaje transferido, utilizamos esta red pre-entrenada de ImageNet, la cual fue entrenada para 1000 clases, con 1.28 millones de imágenes de entrenamiento y 50 mil imágenes de validación[2], la arquitectura de esta red se puede apreciar mejor en la figura 2.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 2: Arquitectura de ImageNet con diferentes convoluciones, tomada de [2]

Para el entrenamiento del modelo de seguimiento de la carretera utilizamos un batch de 16, y permitimos que las imágenes se tomen aleatoriamente del set de datos, para darle variedad, además utilizamos el optimizador Adam, y entrenamos con 150 epochs. En cada epoch del entrenamiento se evalúa su resultado de pérdida de entrenamiento y de validación, y se almacena en el modelo únicamente aquellos datos en el que la pérdida de validación es menor a la mejor pérdida de validación ya almacenada.

Algo similar sucede con el algoritmo para el modo libre, se usa igual un 'batch size' de 16, pero el optimizador utilizado es el de SGD y se entrena por 50 epochs.

Ambos entrenamientos se llevaron a cabo en la computadora de el robot, utilizando para una mejor visualización de la información jupyter lab, el tiempo promedio de entrenamiento para el seguimiento de de la carretera es aproximadamente de 3h 30, mientras que el del modo libre es de 2h.

Modelo creado para el análisis de imágenes

Debido a la necesidad de un hardware sumamente potente para lograr entrenar los algoritmos planteados en un tiempo prudente, y a la falta de este tipo de hardware por parte de los miembros del grupo de trabajo, se decidió utilizar computación el la nube, para así hacer uso de hardware mucho mas potente, en este caso brindado por Google. Para el almacenamiento del dataset, los archivos necesarios para el entrenamiento y el respaldo del modelo

entrenado, se utilizó Google Drive, mientras que para ejecutar el código con el que se realizó el entrenamiento del modelo se usó Google Colaboratory.

En la creación del dataset personalizado, se tomó un 80 % de las imágenes desde teléfonos celulares, exponiendo las señales y objetos a diferentes ambientes, iluminación y ángulos. Mientras que el restante 20 % se tomó desde la cámara del Jetbot. Para el proceso de etiquetar las imágenes se utilizó la herramienta Make Sense [6], la cual genera una serie de archivos con extensión txt, en un formato que es comprendido por YOLO. Antes de iniciar con el proceso de entrenamiento debemos tomar en cuenta una serie de consideraciones. Al entrenar una versión personalizada, los archivos de configuración varían según el numero de clases que vayamos a utilizar.

Los principales aspectos que se deben tomar en cuenta son los siguientes:

1. Se debe descargar y respaldar en Google Drive el archivo de configuración para Tiny-yolov3, así como modificarlo según el numero de clases. Lo primero de se debe cambiar, serán los valores de batch y subdivisions, dándoles un valor de 64 y 16 respectivamente. Seguidamente, se cambiará el width y height correspondiente a las imágenes, configurando este valor a 416x416, pero si se desea tener mejores resultados se puede cambiar estos valores a 608x608. Luego, según la documentación de Darknet [1], calcularemos los valores de max-batches, steps y filters de la siguiente manera.
    - Max-batches: Fórmula = (numero clases x 2000). En nuestro caso sería 9x2000, lo que equivale a 18.000 max-batches. Es importante recalcar en este punto, que el valor de max-batches no debe ser menor a 6000, por lo que si se tiene 1 o 2 clases, el max-batches debe ser 6000.
    - Steps: Fórmula = (max-batches x 0.8 , max-batches x 0.9). Resultando así nuestros steps de la siguiente forma, 14.400, 16.200.
    - Filters: Fórmula = ((numero clases + 5) x 3). Obteniendo como resultado 42 filters.
2. Se debe crear un archivo bajo la extensión obj.names, el cual deberá tener el nombre de las clases que se vayan a utilizar durante el entrenamiento y además deberán coincidir con el orden de las etiquetas para las imágenes. Es decir, si suponemos que solo tenemos 2 clases, mis archivos txt que indican la clase por imagen tendrán un 0 o 1 delante de los puntos correspondientes a los bounding boxes, por esta razón, si la clase 0 es un perro y la clase 1 un gato, en el obj.names primero

se registrara el perro y luego el gato asegurándonos así que las clases hagan match. Para tener un concepto mas gráfico puede consultar la figura 3

3. Otro de los archivos que debemos crear/configurar es el obj.data, este archivo nos permitirá indicarle a YOLO cuales serán los archivos que recibirá como insumo, es decir, tendremos un primer apartado que corresponde al numero de clases, el segundo hace referencia a los archivos de entrenamiento, el tercero es para indicar el obj.names y por ultimo el directorio de la carpeta "backup" en donde se almacenaran los datos correspondientes a los pesos luego de haber iniciado con el entrenamiento. La figura 4 ilustra mejor como debería lucir el archivo.

4. Se debe descargar y almacenar en Drive los pesos pre-entrenados para Tiny-yolov3.

5. Se debe almacenar todas las imágenes y sus respectivas archivos de etiqueta en una capeta con extensión zip. Dicha carpeta debe almacenarse también en Drive.

Se recomienda que todos los archivos antes mencionados deben estar almacenados en la misma carpeta de Google Drive, junto con el archivo .zip que contiene las imágenes y sus etiquetas.

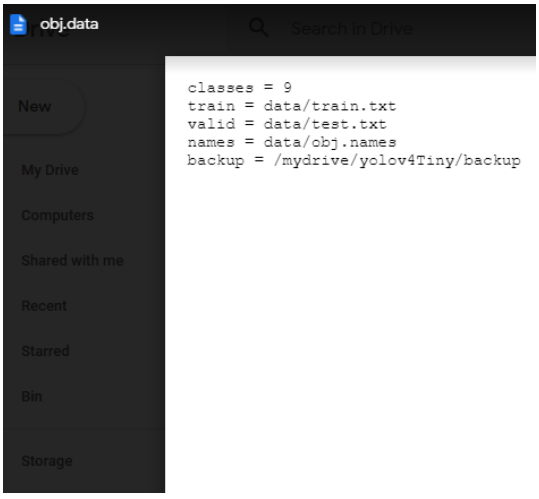

Figura 3: Obj.names ejemplo gráfico

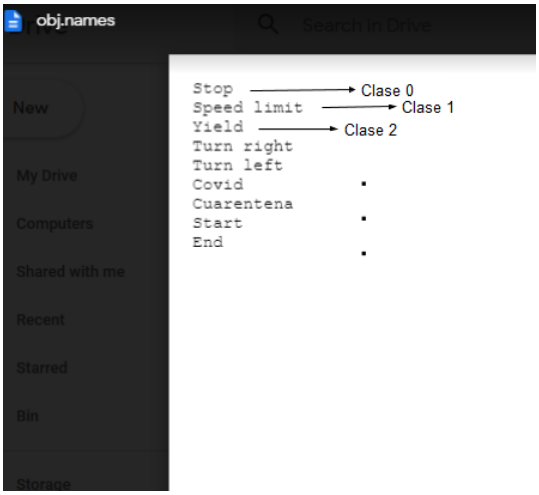

Figura 4: Obj.data ejemplo de configuración

Justificación y explicación del modelo

Modelo creado para el movimiento del robot sobre la carretera

Se elige utilizar estos modelos porque es una de las herramientas que trae el jetbot, es una manera probada de poner a funcionar la inteligencia artificial en el robot, aunque claro, antes de definir que esta iba a ser nuestra opción definitiva realizamos una investigación.

En la búsqueda encontramos otros modelos que utilizan jetson nano, como el jetracer que está diseñado para carreras o el donkey car. El problema es que la mayor parte de estos vehículos tiene una variedad distinta de sensores, por ejemplo un sensor de línea, por lo tanto no es una solución viable ya que el único elemento que tenemos con el jetbot para la detección de la carretera que debe seguir, es la cámara.

Existía una opción de 'reinforcement learning', la cual puede ser una buena opción para un futuro con respecto al robot, porque después de su aprendizaje inicial continúa aprendiendo cada vez que recorre el camino, sin embargo, para empezar con este método es necesario un set de datos de 1000 a 10000 fotos, mientras que el que utilizamos puede empezar a moverse por la carretera con apenas 120 fotos.

Ahora bien, con respecto a la razón por la que se utiliza un framework como pytorch es porque con este es más sencillo e intuitivo hacer deep learning [3]. Por otra parte, con respecto a la red pre-entrenada de resnet 18, como se mencionó anteriormente está entrenada con muchísimas imágenes, por lo tanto aprende fácilmente a partir de tener menos imágenes.

Con respecto a los epochs utilizados, se trató de una prueba y error,

las primeras veces se utilizaron alrededor de 70 epochs, pero su precisión no era muy buena, por esto se probaron diferentes combinaciones, en algún momento probamos 300 y sin embargo, el modelo ya no mejoraba, por lo tanto, decidimos dejarlo con 150 epochs.

Modelo creado para el análisis de imágenes

Como parte de las soluciones que se buscaron para resolver el problema planteado, en primera instancia se creó una CNN con Keras y Tensorflow, las cuales son bibliotecas que facilitan la creación de este tipo de redes convolucionales, pero debido a que los reultados arrojados por el modelo creado no fueron los esperados, se debió cambiar el plan y empezar a investigar sobre Yolo. Una vez que se inició la creación del modelo con Yolo, se logró comprender más sobre la diferencia entre el reconocimiento de imágenes y la detección de objetos en imágenes. Además, una vez probado el nuevo modelo, se descubrió que los resultados fueron muy acertados, logrando este modelo reconocer imágenes hasta de la web, lo que el anterior modelo no lograba, ya que no era capaz de reconocer imágenes que no fueran similares a las que se usaron en el entrenamiento. Debido a que el Jetson Nano no respondió bien al reconocimiento de las señales con el modelo construido en Yolov3, se tuvo que entrenar un nuenvo modelo en Tiny-yolov3, es cual evidentemente es más liviano y más fácil de procesar por el robot. Dicho esto, dado a todas las investigaciones, recomendaciones y pruebas realizadas, la solución propuesta a través de un modelo construido con Tiny-yolov3 y un dataset personalizado, está respaldada por la eficiencia del modelo, el tiempo de entrenamiento y pocos recursos de procesamiento.

Análisis de resultados

Mediante la solución planteada para el problema del movimiento del Jetbot, se obtuvo el recorrido completo en una carretera de 177x161 cm, esto sin identificar las señales durante la realización del recorrido por la pista, debido a que se enfrentaron dos problemas con ese objetivo, uno es que el modelo para la detección de señales junto con el modelo de detección del camino es muy pesado para la GPU, por lo tanto, a la hora de unir ambos modelos, los frames de la cámara empiezan a procesarse muy lento. El otro problema es que, no cuenta con una cámara con sensor de profundidad, por lo tanto al momento en el que el jetbot ve una señal, aunque esté muy lejos o sea la primera vez que la ve, intenta cambiar de dirección o detenerse según sea la acción que deba realizar.

En relación con otro de los objetivos, se logró implementar que el Jetbot caminara libremente evitando colisionar con paredes u objetos que se encontraban bloqueando su camino, esto se pudo realizar de manera bastante precisa y con un set de datos de poca extensión.

En cuanto al reconocimiento de señales, se logró obtener una precisión del 98 % para identificar cada señal de tránsito o las señales que le indican el inicio y el final del recorrido, reaccionando en tiempo real y correctamente según lo que va visualizando, el único problema para obtener un 100 % de este resultado es que el robot aunque detecta muy bien con el modelo las señales de tránsito, tiene problemas para detectar a la muñeca y al carro, ya que el set de datos no contiene imágenes de estos dos objetos tomadas por la misma cámara del robot, como se hizo en las demás clases.

Bibliografía

[1] AlexeyAB. Yolo-v4 and yolo-v3/v2 for windows and linux, nd.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Nikhil Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

[4] X. Ou, P. Yan, Y. Zhang, B. Tu, G. Zhang, J. Wu, and W. Li. Moving object detection method via resnet-18 with encoder–decoder structure in complex scenes. *IEEE Access*, pages 108152–108160, 2019.

[5] Joseph Redmon. Darknet: Open source neural networks in c, 2013–2016.

[6] Piotr Skalski. Make sense, nd.