CONDUCCIÓN AUTÓNOMA MEDIANTE ANN

A PREPRINT

Vanessa Fernández Martínez
v.fernandezmarti@alumnos.urjc.es

José María Cañas Plaza
jmplaza@gsyc.es

Francisco Miguel Rivas Montero
franciscomiguel.rivas@urjc.es

November 19, 2019

ABSTRACT

La conducción autónoma es una tarea que podría superar en seguridad a los conductores humanos. En este documento se presenta el estado del arte sobre conducción autónoma mediante Redes Neuronales Convolucionales (CNN). Se describirán diferentes bases de datos para conducción autónoma como Comma.ai [15] o Udacity [16], así como diferentes arquitecturas de redes neuronales empleadas para el mismo problema, como puede ser *PilotNet* [4] o *ControlNet* [6]. Además, se describirán diferentes simuladores para conducción autónoma.

1 Introducción

Los humanos somos capaces de mirar a la carretera y saber al instante si el coche que conducimos está en una curva o una recta, si hay coches alrededor y cómo interactúan entre ellos. En función a la situación en la que nos encontramos sabemos qué acciones llevar a cabo para lograr una buena conducción. Sin embargo, este procedimiento es más complicado para los ordenadores.

Uno de los problemas que se está estudiando ampliamente en visión artifial en la última década es la conducción autónoma. La conducción autónoma es la acción por la cual un vehículo es capaz de desplazarse sin la intervención humana, aprendiendo las normas de circulación y llevándolas acabo. Para ello los vehículos necesitan sensores (cámaras, LIDAR, radar, etc) que les permitan conocer el entorno en el que están, así como algoritmos que les permitan tomar decisiones.

Hoy en día, cada vez existen más fabricantes de vehículos que incorporan tecnologías de conducción autónoma. Existe un estándar elaborado por la Sociedad de Ingenieros Automotrices (SAE), conocido como J3016 [23], que establece los niveles de conducción autónoma según la capacidad del vehículo.

- Nivel 0: No hay automatización de la conducción. Las tareas de conducción son realizadas en su totalidad por el conductor.
- Nivel 1: Asistencia al conductor. El vehículo tiene algún sistema de automatización de la conducción, ya sea para el control de movimiento longitudinal o el movimiento lateral, pero no ambas cosas a la vez. El conductor realiza el resto de tareas de conducción.
- Nivel 2: Automatización parcial. El vehículo es capaz de actuar de forma independiente dentro de escenarios controlados y en situaciones específicas de conducción. El conductor debe seguir prestando atención a lo que ocurre a su alrededor para evitar posibles riesgos.
- Nivel 3: Automatización condicional. El coche conduce completamente solo y el conductor controla que todo funcione correctamente. El coche también puede ser conducido de forma habitual en cualquier momento.
- Nivel 4: Alta autonomía. El sistema cuenta con los sistemas de automatización presentes en el nivel 3 y con sistemas de detección de objetos y eventos. Además, es capaz de responder ante ellos. El sistema de automatización de la conducción posee un sistema de respaldo para actuar en caso de fallo del sistema principal y poder conducir hasta una situación de riesgo mínimo.
- Nivel 5: Autonomía total. El sistema cuenta con todos los beneficios del sistema de automatización del nivel 4. Sin embargo, en este caso el vehículo podría seguir conduciendo en todo momento o circunstancia.

Ejemplos importantes de conducción autónoma son: el DARPA Grand Challenge y el Urban Challenge. El DARPA Grand Challenge, organizado en 2004 y 2005 en Estados Unidos, fue una carrera de vehículos autónomos que debían recorrer 120 kms por el desierto de Nevada sin intervención humana y disponiendo únicamente de un listado de puntos intermedios entre el principio del circuito y el final. El Urban Challenge, organizado en 2007, fue una carrera de vehículos autónomos por zona urbana en la que debían recorrer 96 km en menos de 6 horas.

Hoy en día el principal obstáculo para la conducción autónoma no se deriva de las limitaciones de la tecnología, sino de factores políticos, jurídicos, de regulación, de infraestructura y de responsabilidad que se deben abordar. A pesar de estas dificultades la investigación ha hecho muchos avances.

En la actualidad se están desarrollando sistemas que toman decisiones empleando una red neuronal profunda, la cual recibe información del entorno mediante diferentes sensores (LIDAR, radas, cámaras, etc.). A partir de los datos recogidos por los sensores la red predice unos valores de salida que serán los empleados para la conducción. Las decisiones tomadas por la red neuronal están determinadas por los datos empleados durante el entrenamiento de la red. Por lo tanto, cuanto más representativo sea el conjunto de datos, mejor rendimiento se espera que tenga la red, ya que conocerá todas las situaciones posibles en las que puede estar el vehículo.

En las próximas secciones se describirán los diferentes datasets y algoritmos empleados en la conducción autónoma, así como diferentes simuladores empleados en la investigación de la conducción autónoma.

2 Bases de datos para conducción autónoma

La conducción autónoma pretende que un vehículo sea capaz de conducir sólo en base a los datos proporcionados por determinados sensores. En concreto, la cámara es el sensor más empleado en las diferentes redes neuronales que veremos en la siguiente sección. Dado que queremos que el vehículo sea capaz de conducir bajo diferentes circunstancias, es decir, en diferentes entornos y diferentes iluminaciones, necesitaremos entrenar el modelo con un conjunto de imágenes representativo. Por ello, a lo largo de los últimos años han surgido diferentes datasets con el fin de solucionar este problema.

2.1 Comma.ai

La startup de conducción autónoma Comma.ai creó en 2016 un conjunto de datos [15] que permite probar modelos para controlar un vehículo autónomo. El conjunto de datos consta de 11 videoclips grabados a 20 Hz de una cámara Point Grey colocada en el parabrisas de un Acura ILX 2016. El conjunto de datos es un archivo zip comprimido de 45 GB.

El conjunto de datos consta de un total de 7.25 horas de datos de conducción. Los *frames* de vídeo tienen un tamaño de 160 x 320 píxeles. Junto a los archivos de video hay un conjunto de medidas de sensores donde se registran medidas como la velocidad, la aceleración, el ángulo de giro, la ubicación del GPS y los ángulos del giroscopio.

También registran los *time stamps* en los que se midieron estas medidas de los sensores y los *time stamps* en que se capturaron los *frames* de la cámara. Los datos los sensores se capturan en bruto y los *frames* de la cámara en archivos HDF5 para que sean fáciles de usar en el aprendizaje automático y el software de control.

2.2 Udacity

Inicialmente, Udacity [16] poseía 40 GB de datos públicos para facilitar que las personas fueran capaces construir modelos competitivos sin acceso al tipo de datos de conducción que Tesla o Google poseen. Sin embargo, debido a que los modelos de aprendizaje profundo necesitan muchos datos, la compañía publicó 183 GB adicionales de datos de conducción.

El conjunto de datos de Ucadity [17] consta de 223 GB de datos. Se grabaron datos de más de 70 minutos de conducción en días soleados y nublados, repartidos en dos días en Mountain View. La variedad de imágenes aumentará la calidad de los resultados y proporcionará a los participantes datos más realistas para poder trabajar, ya que este conjunto de datos representa mejor los desafíos de la conducción en el mundo real y las condiciones variables de la carretera. Los datos almacenados constan de latitud, longitud, marcha, freno, aceleración, ángulos de dirección y velocidad.

3 Simuladores

Un vehículo es caro, lo que implica que muchas investigaciones sobre conducción autónoma solamente estén disponibles para centros de investigacón y corporaciones. Cuando se emplea un vehículo puede que algo falle al probarlo, pudiendo

Figure 1: Simulador CARLA.

incluso romperse el vehículo. Hoy en día existen numerosos simuladores, lo que permite a cualquier persona crear, programar y probar infinidad de vehículos y escenarios de forma segura y económica. Algunos de los simuladores más empleados se explicarán a continuación.

3.1 CARLA

CARLA [18] [20] es un simulador de código abierto para la investigación de conducción autónoma. Se ha desarrollado desde cero para respaldar el desarrollo, el entrenamiento y la validación de sistemas de conducción autónomos. Además, admite diferentes conjuntos de sensores y condiciones ambientales.

CARLA (Figura 1) simula un mundo dinámico y proporciona una interfaz simple entre el mundo y un agente que interactúa con el mundo. Para llevar a cabo esta funcionalidad, CARLA está diseñado como un sistema cliente-servidor, donde el servidor ejecuta la simulación y renderiza la escena. La API del cliente se implementa en Python y es responsable de la interacción entre el agente autónomo y el servidor a través de sockets. El cliente envía comandos y metacomandos al servidor y recibe las lecturas del sensor. Los comandos (dirección, aceleración y frenado) controlan el vehículo. Los metamandatos controlan el comportamiento del servidor y se utilizan para restablecer la simulación, cambiar las propiedades del entorno (condiciones climáticas, iluminación y densidad de automóviles y peatones) y modificar el conjunto de sensores.

CARLA presenta las siguientes características:

- Escalabilidad a través de una arquitectura multi-cliente servidor: varios clientes en el mismo nodo o en diferentes nodos pueden controlar diferentes actores.
- Permite a los usuarios controlar todos los aspectos relacionados con la simulación (generación de tráfico, comportamientos de peatones, climas, sensores, etc).
- Los usuarios pueden configurar diversos conjuntos de sensores (LIDAR, cámaras, sensores de profundidad, GPS, etc).
- Permite deshabilitar la representación para ofrecer una ejecución rápida de la simulación del tráfico y los comportamientos de la carretera para los que no se requieren gráficos.
- Se pueden crear mapas siguiendo el estándar OpenDrive a través de herramientas como RoadRunner.
- Los usuarios pueden definir diferentes situaciones de tráfico.
- Integra ROS.

3.2 Gazebo

Gazebo [19] (Figura 2) es un simulador 3D de código abierto distribuido bajo licencia Apache 2.0. Este simulador se ha utilizado en ámbitos de investigación en robótica e Inteligencia Artificial. Es capaz de simular robots, objetos y sensores en entornos complejos de interior y exterior. Posee gráficos de gran calidad y un robusto motor de físicas (masa del robot, rozamiento, inercia, amortiguamiento, etc.). Fue elegido para realizar el DARPA Robotics Challenge (2012-2015) y está mantenido por la Fundación Robótica de Código Abierto (OSRF).

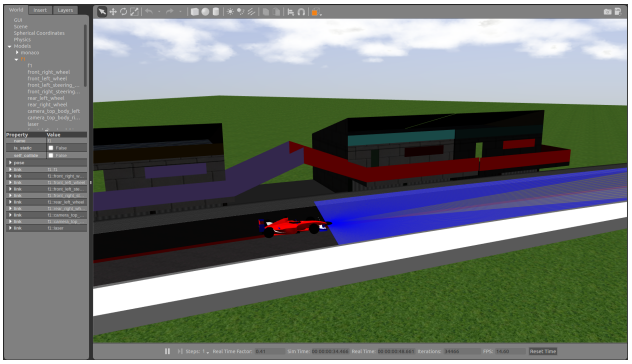

Figure 2: Simulador Gazebo.

Figure 3: Simulador Udacity's Self-Driving Car Simulator.

Los modelos de robots que se emplean en la simulación son creados mediante algún programa de modelado 3D (Blender, Sketchup, etc). Estos robots simulados necesitan ser dotados de inteligencia para lo cual se emplean los plugins. Estos plugins pueden dotar al robot de inteligencia u ofrecer la información de sus sensores a aplicaciones externas y recibir de éstas comandos para los actuadores de los robots.

3.3 Udacity's Self-Driving Car Simulator

Udacity's Self-Driving Car Simulator [16] [21] fue construido para Udacity's Self-Driving Car Nanodegree con el objetivo de que los estudiantes pudieran aprender cómo entrenar modelos de aprendizaje profundo que permitieran a los vehículos conducir de forma autónoma. Este simulador es de código abierto y requiere Unity.

El simulador de Udacity (Figura 3) permite al usuario seleccionar la escena deseada así como el modo de conducción en la pantalla principal. Existen dos modos de conducción: *Training Mode* y *Autonomus Mode*. En el modo *Training Mode* el coche se conduce manualmente mediante el teclado o el ratón y se almacenan los datos de conducción y las imágenes de las cámaras que posee el vehículo. Los datos grabados con este modo se pueden emplear para entrenar un modelo de aprendizaje automático. En el modo *Autonomous Mode* se puede probar el modelo de aprendizaje automático creado y comprobar su rendimiento en ejecución.

Técnicamente, el simulador actúa como un servidor desde el cual el programa puede conectarse y recibir un flujo de imágenes. Se puede crear un programa de Python que emplea un modelo de aprendizaje automático para procesar las imágenes de la carretera para predecir las mejores instrucciones de conducción y enviarlas de vuelta al servidor. Cada instrucción de conducción contiene un ángulo de dirección y un dato de aceleración, que cambia la dirección y la velocidad del automóvil.

Figure 4: Simulador Deepdrive.

3.4 Deepdrive 2.0

Deepdrive 2.0 [22] es un simulador de código abierto para Linux y Windows. Los simuladores actuales parecen vincularse a un hardware específico o no tienen forma de vincularse vehículos físicos. Deepdrive (Figura 4) para conseguir este propóstio, incluye una amplia gama de sensores, automóviles y entornos y facilita la transferencia a vehículos reales. Esto permitirá que un mayor número de personas utilice únicamente el simulador para hacer pruebas cosntantes.

Presenta algunas características únicas respecto a otros simuladores de código abierto:

- El *frame rate* es más elevado al emplear varias cámaras, ya que emplea memoria compartida en lugar de sockets y transferencia asíncrona.
- La superficie de la carretera no es plana, sino que incluye colinas, curvas y la anchura de la carretera varía.
- El mapa, los automóviles, la iluminación, etc. son gratuitos y son modificables en Unreal.

4 Redes neuronales para conducción autónoma

La conducción autónoma no posible sin un algoritmo que tome decisiones. En esta sección se describirán diferentes arquitecturas de redes neuronales empleadas en la conducción autónoma.

4.1 Redes neuronales convolucionales

El aprendizaje de extremo a extremo para conducción autónoma se ha explorado desde finales de los años ochenta. The Autonomous Land Vehicle in a Neural Network (ALVINN) [1] se desarrolló para aprender ángulos de dirección a partir de una cámara y las medidas proporcionadas por un láser mediante una red neuronal con una sola capa oculta. Basados en esta idea de redes de extremo a extremo (dada una imagen o imágenes se preciden ángulos de dirección), exiten mútiples aproximaciones [2] [3] [8] de las cuales veremos algunas a continuación.

Un buen ejemplo de red de extremo a extremo es la red PilotNet [3] [4] creada por Nvidia. En [3] se describe dicha red con detalle. Es una red neuronal convolucional (CNN) que mapea píxeles en crudo de una sola cámara frontal a comandos de dirección directamente. El comando propuesto por la CNN se compara con el comando deseado para la imagen en concreto y los pesos de la red se van ajustando para aproximar la salida de la red a la salida deseada. El ajuste de los pesos se realiza empleando *back propagation*.

La red PilotNet (Figura 5) consta de 9 capas, que incluyen una capa de normalización, 5 capas convolucionales y 3 capas *fully-connected*. La imagen de entrada se divide en planos YUV y se pasa a la red. Las capas convolucionales las diseñaron para realizar la extracción de características y las eligieron a través de experimentos que variaban las configuraciones de capas. Las dos primeras capas convolucionales usaban un *stride* de 2x2 y un kernel 5x5, mientras que las 3 últimas capas usaban un *non-stride* y un kernel 3x3. Las 3 capas *fully-connected* fueron diseñadas para funcionar como un controlador de la dirección, pero no es posible saber exactamente qué partes de la red funcionan principalmente como extractor de características y cuáles sirven como controlador. El sistema aprende automáticamente las representaciones internas, como la detección de características útiles de la carretera.

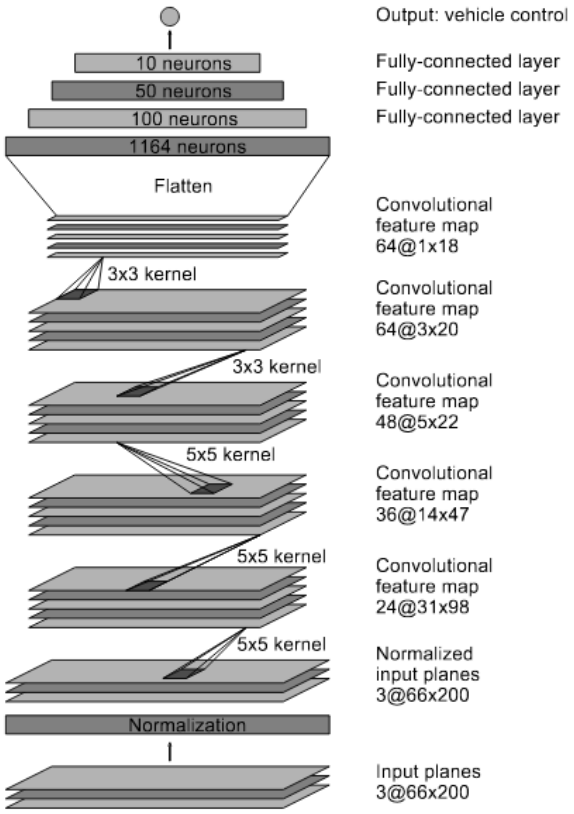

Figure 5: Arquitectura Pilotnet.

El objetivo de [4] es explicar lo que PilotNet aprende y cómo toma sus decisiones. Con este fin, se desarrolla un método para determinar qué elementos en la imagen de la carretera influyen más en la decisión de la dirección de PilotNet. Llaman a estas secciones de imagen salientes los objetos salientes. Se puede encontrar un informe detallado del su método de detección de saliencia en [9].

La idea central de [4] para discernir los objetos salientes es encontrar partes de la imagen que corresponden a ubicaciones donde los mapas de características tienen las mejores activaciones. Las activaciones de los mapas de nivel superior se convierten en máscaras para las activaciones de niveles inferiores utilizando el siguiente algoritmo:

1. En cada capa, las activaciones de los mapas de características se promedian.
2. El mapa con el promedio más alto se escala según el tamaño del mapa de la capa de abajo. El aumento de escala se realiza mediante una deconvolución. Los parámetros (*filter size* y *stride*) utilizados para la deconvolución son los mismos que se emplearon en la capa convolucional utilizada para generar el mapa. Los pesos de la deconvolución se establecen en 1.0 y los sesgos en 0.0.
3. El mapa promediado aumentado de un nivel superior se multiplica después con el mapa promediado de la capa de abajo (ahora son del mismo tamaño). El resultado es una máscara de tamaño intermedio.
4. La máscara intermedia se escala al tamaño de los mapas de la capa inferior de la misma manera que en el paso 2.
5. El mapa intermedio mejorado se multiplica de nuevo con el mapa promediado de la capa de abajo. Se obtiene una nueva máscara intermedia.
6. Los pasos 4 y 5 se repiten hasta que se alcanza la entrada. La última máscara que es del tamaño de la imagen de entrada se normaliza al rango 0-1 y se convierte en la máscara de visualización final.

Figure 6: Ejemplos de objetos salientes para varias imágenes de entrada.

Esta máscara de visualización muestra qué regiones de la imagen de entrada contribuyen más a la salida de la red. Estas regiones identifican los objetos salientes. En la Figura 6 se pueden ver ejemplos de objetos salientes para varias imágenes de entrada.

Los resultados muestran que PilotNet aprende a reconocer objetos relevantes en la carretera y que es capaz de mantener el vehículo en el carril con éxito en una amplia variedad de condiciones, independientemente de si las marcas del carril están presentes en la carretera o no.

En [14] se propone una nueva arquitectura de red, llamada TinyPilotnet, que se deriva de la red Pilotnet [3] [4]. La red TinyPilotnet (Figura 7) está compuesta por una capa de entrada, en la que se introducirán imágenes de resolución 16x32 y un único canal, seguida por dos capas convolucionales de kernel 3x3, y una capa *dropout* configurada al 50% de probabilidad para agilizar el entrenamiento. Finalmente, el tensor de información se convierte en un vector que es conectado a dos capas *fully-connected* que conducen a un par de neuronas, cada una de ellas dedicada a predecir los valores de dirección y aceleración respectivamente. La imagen de entrada tiene un solo canal formado por el canal de saturación del espacio de color HSV.

En [13] se presenta un enfoque de red neuronal profunda que emplea cámaras de eventos (sensores de inspiración biológica que no adquieren imágenes completas a una velocidad de *frames* fija, sino que tienen píxeles independientes que solo producen cambios de intensidad de forma asíncrona en el momento en el que ocurren) para predecir el ángulo de giro de un vehículo. Los eventos se convierten en *frames* de eventos por acumulación de píxeles en un intervalo de tiempo constante. Posteriormente, una red neuronal profunda los asigna a los ángulos de dirección.

En este artículo inicialmente apilan los *frames* de eventos de diferente polaridad, creando una imagen de eventos 2D. Después, implementan una serie de arquitecturas ResNet, es decir, ResNet18 y ResNet50. Estas redes son utilizadas como extractores de características para el problema de regresión, considerando solo las capas convolucionales. Para codificar las características de la imagen extraídas de la última capa convolucional en un descriptor vectorizado, se emplea una capa *global average pooling* que devuelve la media del canal de las características. Después se agrega una capa *fully-connected* (con dimensionalidad 256 para ResNet18 y 1024 para ResNet50), seguida de una ReLU no lineal y una capa *fully-connected* unidimensional para generar el ángulo.

En [13] preciden ángulos empleando 3 tipos de entradas: 1. imágenes en escala de grises, 2. diferencia de imágenes en escala de grises, 3. imágenes creadas por la acumulación de eventos. Analizan el rendimiento de la red en función del tiempo de integración utilizado para generar las imágenes de eventos (10, 25, 50, 100 y 200 ms). Cuanto mayor es el tiempo de integración, mayor es la traza de eventos que aparecen en los contornos de los objetos. La red funciona mejor cuando se entrena con imágenes de eventos correspondientes a 50 ms, y el rendimiento se degrada para tiempos de

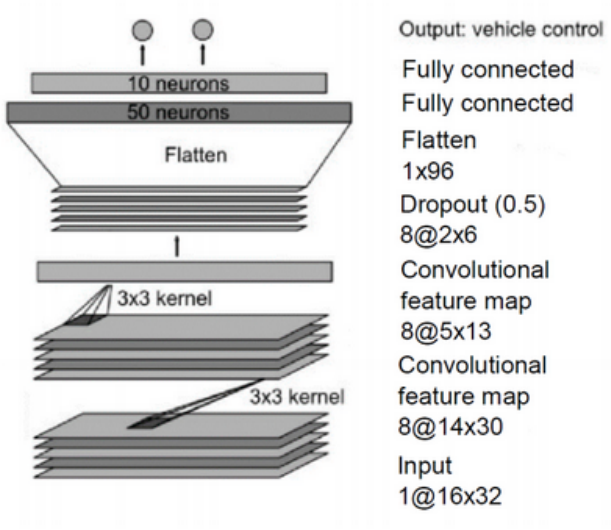

Figure 7: Arquitectura TinyPilotnet.

integración cada vez más grandes. Uno de los problemas que presentan las entradas que emplean imágenes en escala de grises es que a altas velocidades las imágenes se difuminan y la diferencia de imágenes se vuelve muy ruidosa.

En [12] se realiza un amplio estudio donde se analiza una red neuronal de extremo a extremo para predecir las acciones de dirección de un vehículo en base a las imágenes de una cámara, así como las dependencias temporales de entradas consecutivas y la diferencia entre redes de clasificación y redes de regresión.

La arquitectura principal que emplean es una variación de la arquitectura PilotNet, AlexNet o VGG19. Para AlexNet se elimina el *dropout* de las 2 capas densas finales y se reduce el tamaño de 500 y 200 neuronas. La capa de salida de la red depende de su tipo (regresión o clasificación) y para una red de clasificación del número de clases. Para el caso de clasificación, cuantifican las medidas del ángulo de dirección en valores discretos, que representan las etiquetas de la clase. Esta cuantificación es necesaria como entrada cuando se tiene una red de clasificación y permite equilibrar los datos a través de los pesos de la muestra. Esta ponderación actúa como un coeficiente para la tasa de aprendizaje de la red para cada muestra. El peso de una muestra está directamente relacionado con la clase a la que pertenece cuando se cuantifica. La ponderación de muestra se realiza para regresión y clasificación.

Se estudia la influencia de las especificaciones de cuantización de clase en el rendimiento del sistema. Estas especificaciones consisten en la cantidad de clases y la asignación del rango de entrada de estas clases. Se comparan redes con diferentes grados de granularidad, lo que influye en el rendimiento. Se compara un esquema de cuantificación de grano grueso de 7 clases con uno de grano fino de 17 clases, obteniendo mejores resultados con el de grano grueso.

Además, en este artículo se evalúan métodos que permiten que nuestro sistema aproveche la información de entradas consecutivas: un método que sigue una arquitectura de extremo a extremo y un método que emplea capas recurrentes (lo veremos en la siguiente subsección).

El método que emplea una CNN para la predicción, que llaman *stacked frames*, concatena varias imágenes de entrada consecutivas para crear una imagen apilada. La entrada a la red es esta imagen apilada (para la imagen t se concatenan las imágenes t-1, t-2, etc). El tamaño de entrada será la única variable que se modique, es decir, no se modifica la red. Por esta razón, las imágenes se concatenan en la dimensión de profundidad (canal) y no en una nueva dimensión. Por ejemplo, apilar 2 imágenes anteriores a la imagen RGB actual de 160 x 320 x 3 cambaría su tamaño a 160 x 320 x 9. Los resultados muestran un aumento en el rendimiento de las métricas con este método. Se cree que es debido a que la red puede hacer una predicción basada en la información promedio de múltiples imágenes. Para una sola imagen, el valor predicho puede ser o muy alto o muy bajo. En cambio, para imágenes concatenadas, la información combinada podría cancelarse entre sí, dando una mejor predicción promedio. Suponiendo que la red promedie la información, aumentar el número de imágenes podría hacer que la red perdiera la capacidad de respuesta. Por ello emplean 3 *frames* concatenados.

Además, en este artículo se demuestra cualitativamente que las métricas estándar que se emplean para evaluar redes no necesariamente reflejan con precisión el comportamiento de conducción de un sistema. Una matriz de confusión

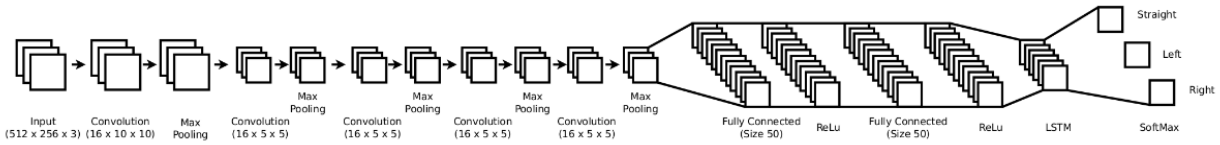

Figure 8: Estructura de red ControlNet.

prometedora puede dar como resultado un comportamiento de conducción deficiente, mientras que una matriz con mal aspecto puede dar como resultado un buen comportamiento de conducción.

4.2 Redes neuronales recurrentes

Las redes neuronales recurrentes (RNNs) representan una clase de redes neuronales artificiales que utilizan células de memoria para modelar la relación temporal entre los datos de entrada y, por lo tanto, aprender la dinámica subyacente. Con la introducción de las Long Short-Term Memory (LSTM), el modelado de relaciones a largo plazo se hizo posible dentro de RNN.

En múltiples investigaciones sobre conducción autónoma se ha aprovechado la capacidad de estas redes para poder aprovechar la información de imágenes consecutivas. Algunas de estas investigaciones las veremos a continuación.

Un ejemplo de investigación donde se emplean capas LSTM es la propuesta por [6]. En esta investigación se presenta un controlador reactivo basado en aprendizaje profundo que emplea una arquitectura de red simple que requiere pocas imágenes de entrenamiento. A pesar de esta estructura simple, su arquitectura de red, llamada ControlNet, supera a otras redes más complejas en múltiples entornos (entornos interiores estructurados y entornos exteriores no estructurados) utilizando diferentes plataformas robóticas. Es decir, el artículo se centra en el control reactivo, donde el robot debe evitar obstáculos que no están presentes durante la construcción del mapa.

ControlNet extrae imágenes RGB para generar comandos de control: gira a la derecha, gira a la izquierda y recto. La arquitectura de ControlNet consiste en alternar capas convolucionales con capas de *maxpooling* seguidas de capas *fully-connected*. Las capas convolucionales y la de *pooling* extraen información geométrica sobre el medio ambiente, mientras que las capas *fully-connected* actúan como un clasificador general. La capa LSTM permite al robot incorporar información temporal permitiéndole continuar moviéndose en la misma dirección sobre varios *frames*. La estructura de ControlNet (Figura 8) es:

- 2D Convolution, 16 filtros de tamaño 10x10
- Max Pooling, filtro de 3x3, stride de 2
- 2D Convolution, 16 filtros de tamaño 5x5
- Max Pooling, filtro de 3x3, stride de 2
- 2D Convolution, 16 filtros de tamaño 5x5
- Max Pooling, filtro de 3x3, stride de 2
- 2D Convolution, 16 filtros de tamaño 5x5
- Max Pooling, filtro de 3x3, stride de 2
- 2D Convolution, 16 filtros de tamaño 5x5
- Max Pooling, filtro de 3x3, stride de 2
- Fully connected, 50 neuronas
- ReLu
- Fully connected, 50 neuronas
- LSTM (5 frames)
- Softmax con 3 salidas

En [5] se propone una Convolutional Long Short-Term Memory Recurrent Neural Networks, conocido como C-LSTM (Figura 9), que es entrenable de extremo a extremo, para aprender las dependencias visual y temporal dinámica de la conducción. El sistema investigado está compuesto por una cámara RGB frontal y una red neuronal que consta de una

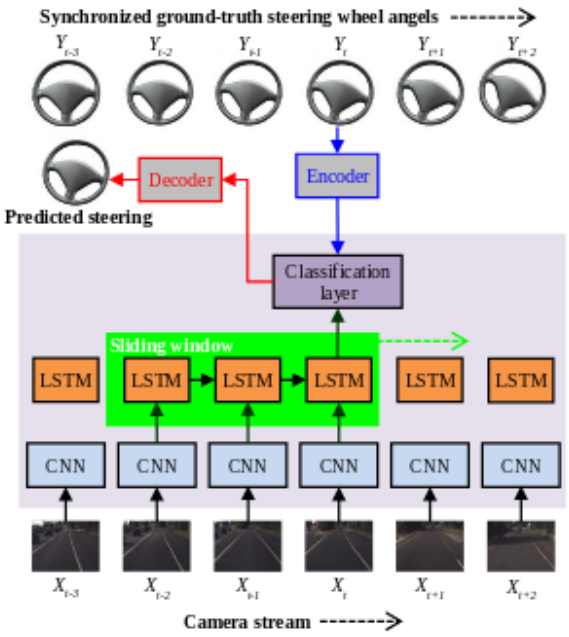

Figure 9: Arquitectura C-LSTM.

CNN y LSTM que estiman el ángulo del volante en función de la entrada de la cámrara. Las imágenes de la cámara se procesan fotograma a fotogama por la CNN. Las características resultantes luego se procesan dentro de la red LSTM para aprender las dependencias temporales. La predicción del ángulo de dirección se calcula a través de la capa de clasificación de salida después de las capas LSTM.

Aplican el concepto de *transfer learning*. La CNN está pre-entrenada en el conjunto de datos Imagenet. Luego, transfieren la red neuronal entrenada a otra específica enfocada en imágenes de conducción. Posteriormente, en la LSTM se procesa una secuencia de vectores de características de longitud fija w de la CNN. A su vez, las capas LSTM aprenden a reconocer las dependencias temporales que conducen a una decisión de dirección Yt basada en las entradas de Xt-w a Xt. Los valores pequeños de t conducen a reacciones más rápidas, pero la red aprende solo las dependencias a corto plazo y la suceptibilidad a los aumentos de *frames* mal clasificados individualmente. Mientras que los valores elevados de t conducen a un comportamiento más suave y, por tanto, predicciones de dirección más estables, pero aumenta las posibilidades de aprender dependencias erróneas a largo plazo.

El concepto de ventana deslizante permite a la red aprender a reconocer diferentes ángulos de dirección desde el mismo *frame* Xi pero en diferentes estados temporales de las capas LSTM. Tanto los pesos de la LSTM como de la CNN se comparten en diferentes pasos dentro de la ventana deslizante y, esto permite un tamaño de ventana arbitrariamente largo.

Plantean la regresión del ángulo de dirección como un problema de clasificación. Esta es la razón por la que el único número que representa el ángulo de dirección Yt está codificado como un vector de activaciones de las neuronas de la capa de clasificación. Utilizan una capa totalmente conectada con activaciones *tanh* para la capa de clasificación.

En esta propuesta para el entrenamiento de dominio "específico", la capa de clasificación de la CNN se reinicializa y se entrena con los datos de carretera de la cámara. El entrenamiento de la capa LSTM se lleva a cabo de manera múltiple, la red aprende las decisiones de dirección que están asociadas con los intervalos de conducción. La capa de clasificación y las capas LSTM emplean una mayor velocidad de aprendizaje porque se inicializan con valores aleatorios. La CNN y la LSTM se entrenan conjuntamente al mismo tiempo.

En [7] se propone un modelo basado en visión que mapea imágenes de entrada en ángulos de direccion usando redes profundas. Se segmenta la red en subredes. Es decir, los *frames* se introducen primero en una red de extracción de características, generando una representación de características de longitud fija que modela el entorno visual y el estado interno de un vehículo. Las características extraídas se envían a una red de predicción de dirección. En la subred de extracción de características emplea una Spatio-Temporal Convolution (ST-Conv) que cambia las dimensiones temporales y espaciales. Se emplea una capa *fully-connected* tras la ST-Conv para obtener un vector de características

de dimensión 128. Además, en la subred de extracción de características se introducen capas LSTM, para lo cual se emplea ConvLSTM. La subred de predicción de dirección propone concatenar acciones de dirección y de estado del vehículo con el vector de características de 128 dimensiones. Para ello se añade 1 paso de recurrencia entre la salida final y las dos capas *concat* justo antes/después de la LSTM. La capa *concat* antes de la LSTM agrega la velocidad, y el par de torsión y ángulo de rueda al vector de 128 dimensiones, formando un vector de 131 dimensiones. La capa *concat* después de LSTM está compuesta por un vector de características 128-d + salida de LSTM 64-d + salida final previa 3d.

En [8] se propone un modelo de atención visual para entrenar una red convolucional de extremo a extremo desde las imágenes hasta el ángulo de giro. El modelo de atención resalta las regiones de imagen que potencialmente influye en la salida de la red, de las cuales algunas son influencias reales y otras espureas. Su modelo predice comandos de ángulo de dirección continuos a partir de píxeles en bruto. El modelo predice el radio de giro inverso ût, pero se relaciona con el comando de ángulo de dirección mediante geometría de Ackermann.

En este método emplean una red neuronal convolucional para extraer un conjunto de vectores de características visuales codificadas, a las que se refieren como una característica convolucional cubo xt. Cada vector de características puede contener descripciones de objetos de alto nivel que permiten que el modelo de atención preste atención selectiva a ciertas partes de una imagen de entrada al elegir un subconjunto de vectores de características. Utilizan la red PilotNet [3] para aprender un modelo de conducción, pero omiten las capas de *maxpooling* para evitar la pérdida de información de ubicación espacial. Recopilan un cubo xt de características convolucionales tridimensionales de la última capa empujando la imagen preprocesada a través del modelo, y el cubo de características de salida se emplea como entrada de las capas LSTM. Utilizan una red LSTM que predice el radio de giro inverso y genera ponderaciones de atención en cada paso de tiempo t condicionado al estado oculto anterior y una característica convolucional actual xt. Asumen una capa oculta condicionada al estado oculto anterior y los vectores de características actuales. El peso de atención para cada ubicación espacial se calcula luego mediante una función de regresión logística multinomial.

El último paso de este método es un decodificador de grano fio en el que refinan un mapa de atención visual y detectan saliencias visuales locales. Aunque un mapa de atención del decodificador de grano grueso proporciona una probabilidad de importancia sobre un espacio de imagen 2D, el modelo debe determinar regiones específicas que causan un efecto casual en el rendimiento de la predicción. Obtienen una disminución en el rendimiento cuando se oculta una prominencia visual local en una imagen de entrada en bruto. En primer lugar, recopilan un conjunto consecutivo de pesos de atención e ingresan imágenes en bruto para los T pasos de tiempo especificados por el usuario. Luego, crean un mapa de atención, Mt. La red neuronal de 5 capas (basada en PilotNet) emplea una pila de filtros 5x5 y 3x3 sin ninguna capa *pooling*, y por tanto la imagen de dimensiones 80x160 se procesa para producir un cubo de características 10x20x64, conservando su relación de aspecto. Para extraer una prominencia visual local, primero muestrean aleatoriamente partículas de 2D con reemplazo sobre una imagen de entrada condicionada en el mapa de atención Mt. También emplean el eje de tiempo como la tercera dimensión para considerar las características temporales de las saliencias visuales, almacenando partículas espacio temporales 3D. Posteriormente, aplican un algoritmo de *clustering* (DBSCAN) para encontrar una prominencia visual local agrupando las partículas 3D en *clusters*. Para los puntos de cada grupo y cada *frame* de tiempo t, calculan el algoritmo *convex hull* para encontrar una región local de cada prominencia visual destacada.

En [12] además de aprovechar la información temporal concatenando *frames* se estudia la inclusión de capas recurrentes. Es decir, modifican su arquitectura para incluir capas LSTM, que permiten capturar información temporal entre entradas consecutivas. Las redes se entrenan con un vector de entrada que consiste en la imagen de entrada y una serie de imágenes anteriores, lo que se traduce en una ventana de tiempo. Comparan muchas variaciones de la arquitectura PilotNet [3]: (1) se cambia una o dos capas densas a capas LSTM, (2) se agrega una capa LSTM después de las capas densas, y (3) se cambia la capa de salida a LSTM. Todos los experimentos que realizan con redes LSTM demostraron que la incorporación de capas LSTM no aumentó ni redujo el rendimiento de la red.

En [14] se propone una nueva arquitectura basada en la arquitectura TinyPilotnet (Figura 7) para mejorar el rendimiento de la misma. Esta nueva red (Figura 10) está formada principalmente por 3 capas convolucionales de kernel 3x3, combinadas con capas *maxpooling*, seguidas por 3 capas LSTM convolucionales 5x5 y 2 capas *fully-connected*. Las capas LSTM producen un efecto de memoria, por lo que los ángulos de dirección y los valores de aceleración dados por la CNN están influenciados por los anteriores.

5 Conclusiones

En este estado del arte hemos visto diferentes conjuntos de datos, así como diferentes algoritmos de conducción autónoma. Unos obtendrán mejores resultados que otros en función del entorno, así como unos tendrán mayor velocidad de cómputo que otros. Además, hemos visto algunos simuladores que se pueden emplear para ver la eficiencia de las redes de conducción.

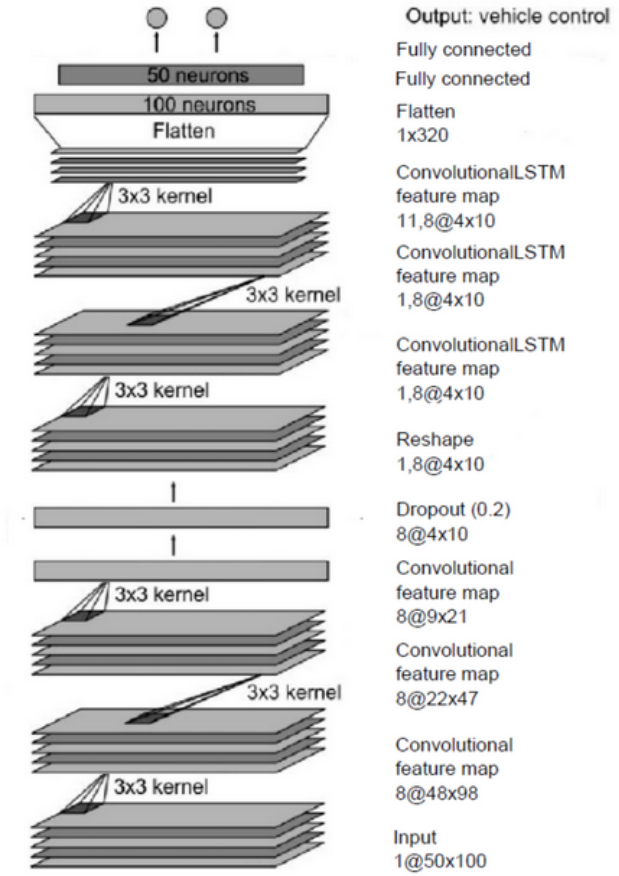

Figure 10: Arquitectura DeepestLSTM-TinyPilotnet.

En este TFM se propone la creación de una base de datos para conducción autónoma en el simulador Gazebo. Esta base de datos se basará en la solución propuesta para la asignaturá "Visión Robótica" del Máster de Visión Artificial para la práctica "Follow line" [24]. Además, se propone una investigación de redes extremo a extremo para conducción autónoma.

References

[1] Dean A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. *Technical report, Carnegie Mellon University. URL:https://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf*, 1989.

[2] Yann LeCun, Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp. Off-RoadObstacleAvoidancethroughEnd-to-EndLearning. *URL:http://yann.lecun.com/exdb/publis/pdf/lecun-dave-05.pdf*, 2005.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal,Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, Urs Muller. Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. *arXiv preprint arXiv:1704.07911*, 2017.

[5] Hesham M. Eraqi, Mohamed N. Moustafa, Jens Honer. End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies. *arXiv preprint arXiv:1710.03804*, 2017.

[6] Keith Sullivan, Wallace Lawson. Reactive Ground Vehicle Control via Deep Networks. *URL:https://pdfs.semanticscholar.org/ec17/ec40bb48ec396c626506b6fe5386a614d1c7.pdf*, 2017.

[7] Lu Chi, Yadong Mu. Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues. *arXiv preprint arXiv:1708.03798*, 2017.

[8] Jinkyu Kim, John Canny. Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention. *arXiv preprint arXiv:1703.10631*, 2017.

[9] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, Karol Zieba. VisualBackProp: efficient visualization of CNNs. *arXiv preprint arXiv:1611.05418*, 2017.

[10] Gustav von Zitzewitz. Survey of neural networks in autonomous driving. *URL:https://www.researchgate.net/publication/324476862_Survey_of_neural_networks_in_autonomous_driving*, 2017.

[11] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, Byron Boots. Agile Autonomous Driving using End-to-End Deep Imitation Learning. *arXiv preprint arXiv:1709.07174*, 2018.

[12] Jonas Heylen, Seppe Iven, Bert De Brabandere, Jose Oramas M., Luc Van Gool, Tinne Tuytelaars. From Pixels to Actions:Learning to Drive a Car with Deep Neural Networks. In *IEEE Winter Conference on Applications of Computer Vision*, 2018.

[13] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, Davide Scaramuzza. Event-based Vision meets Deep Learning on Steering Prediction for Self-driving Cars. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[14] del Egio J., Bergasa L.M., Romera E., Gómez Huélamo C., Araluce J., Barea R. Self-driving a Car in Simulation Through a CNN. *In: Fuentetaja Pizán R., García Olaya Á., Sesmero Lorente M., Iglesias Martínez J., Ledezma Espino A. (eds) Advances in Physical Agents. WAF 2018. Advances in Intelligent Systems and Computing, vol 855. Springer, Cham*, 2019.

[15] Eder Santana, George Hotz. Learning a Driving Simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[16] Udacity. Public driving dataset. *URL:https://eu.udacity.com/course/self-driving-car-engineer-nanodegree–nd013*, 2017.

[17] Udacity's Datasets. Public driving dataset. *URL:https://github.com/udacity/self-driving-car/tree/master/datasets*, 2016.

[18] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio López, Vladlen Koltun. CARLA: An Open Urban Driving Simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[19] Gazebo. Simulator. *URL:http://gazebosim.org/*, 2018.

[20] CARLA. An Open Urban Driving Simulator. *URL:http://carla.org/*, 2017.

[21] Udacity's Self-Driving Car Nanodegree. Self-Driving Simulator. *URL:https://github.com/udacity/self-driving-car-sim*, 2017.

[22] Deepdrive. Self-Driving Simulator. *URL:https://deepdrive.io/*, 2017.

[23] Levels of Driving Automation. Automate driving levels of driving automation are defined in new SAE International Standard J3016. *URL:https://www.smmt.co.uk/wp-content/uploads/sites/2/automated_driving.pdf*, 2017.

[24] Follow line. JdeRobot RoboticsAcademy. *https://github.com/JdeRobot/RoboticsAcademy/tree/master/exercises/follow_line*, 2019.