UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADO

"Aprendizaje fin a fin para la conducción autónoma de vehículos domésticos usando visión artificial y redes neuronales convolucionales"

POSTULANTE: JOSE EDUARDO LARUTA ESPEJO
TUTOR: JAVIER SANABRIA GARCIA
D.A.M.: GONZALO SAMUEL CABA MORALES

LA PAZ, AGOSTO 2018

Índice general

Capítulo 1

Introducción

1.1. Antecedentes

El primer intento de desarrollo de un sistema de conducción autónomo "fin a fin" fue llevado a cabo por la Agencia de Proyectos de Investigación Avanzada en Defensa de los Estados Unidos (DARPA) con un proyecto conocido como el Vehículo Autónomo de DARPA o DAVE [1] en el cual un vehículo radio controlado a escala tenía la tarea de conducir a través de un entorno escabroso. El vehículo DAVE fue entrenado a partir de cientos de horas de conducción humana en entornos similares pero no idénticos. Los datos de entrenamiento incluyeron imágenes de dos cámaras de video y comandos de control generados por un operador humano.

Paralelamente a este esfuerzo realizado por el DARPA y debido a la limitada capacidad computacional de la época, los avances en las distintas tareas que componen la conducción autónoma se han enfocado en el tratamiento de las señales y datos provenientes de los sensores con algoritmos de procesamiento básicos llegando a crearse implementaciones efectivas basadas en un flujo de trabajo descrito a continuación.

1.1.1. Sistemas de Conducción Autónoma

Un sistema de conducción autónoma es una combinación de varios componentes o subsistemas donde las tareas de percepción, toma de decisiones y operación de un vehículo son desarrolladas por un sistema electrónico en lugar de un conductor humano.

El primer hito en el desarrollo de un sistema completamente autónomo vino con la organización del DARPA "Grand Challenge" en el cual equipos de varias universidades, institutos de investigación y empresas tuvieron que enfrentar el difícil reto de desarrollar un sistema capaz de controlar un vehículo doméstico a través de una carretera ripiada en medio del desierto de Arizona. Dentro las 2 versiones del Darpa Grand Challenge destacaron los proyectos de universidades como Stanford con el robot Stanley [2] que fue el primer vehículo en recorrer mas de 170 kilómetros en una carretera ripiada de manera completamente autónoma.

El éxito de los proyectos que participaron en el grand challenge sentó un gran precedente

Figura 1.1: Stanley, el vehículo autónomo de Stanford que ganó la competencia DARPA Grand Challenge en 2005. Fuente: stanford.edu

en el desarrollo de lo que ahora se conoce como *Self Driving Car* o vehículo autónomo. De hecho, muchos de los equipos participantes de este concurso se constituyen en la actualidad como existosas empresas de desarrollo o coadyuvan en iniciativas privadas de gigantes de la tecnología como Google, Uber o Nissan.

Sin embargo, debido al creciente interés tanto en investigación como económico en los sistemas de conducción autónoma, la Sociedad de Ingenieros en Automoción (SAE por sus siglas en inglés) ha elaborado un estándar donde se detallan distintos aspectos concernientes. La regulación define varios niveles de autonomía en vehículos terrestres, aéreos y acuáticos yendo desde un control completamente manual, normalmente observado en vehículos completamente mecánicos, pasando por asistencias al control hasta llegar a un vehículo completamente autónomo en todas sus tareas

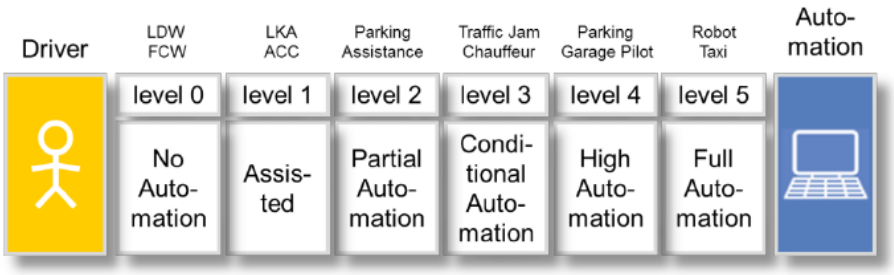

Figura 1.2: Niveles de automatización en la conducción según SAE. Fuente: researchgate

La creación de estándares y regulación ha tenido como consecuencia que, en la actualidad, existan varias iniciativas en el desarrollo de los *self Driving Cars*, siendo una de las más importantes la empresa Waymo, dependiente de Google a través de su empresa Pública Alphabet. Waymo, ha aprovechado el uso de tecnologías emergentes de sensado como el LIDAR para mejorar el mapeo y la navegación a través de algoritmos de fusión de sensores. Aparte de Alphabet, existen diversas iniciativas privadas en el desarrollo de vehículos autónomos

con fines comerciales como los Self Driving Cars de Uber, Toyota, BMW, Ford, entre otros.

Figura 1.3: El vehículo autónomo de Waymo. waymo.com

Una de las tareas más importantes dentro de un *self driving car* es la detección y mantención del carril del vehículo. Fabricantes de vehículos automotores han incluido con éxito sistemas de asistencia al conductor para la mantención del carril usando cámaras digitales y visión artificial para poder detectar la posición del automóvil con respecto al carril. Estos sistemas se consideran fundamentales en sistemas de conducción autónoma. Durante las últimas dos décadas, se han desarrollado distintos tipos de sistemas y enfoques para resolver el problema de la mantención de carril.

1.1.1.1. Arquitectura general de un sistema de conducción autónoma

En general, la arquitectura de un sistema de conducción autónoma se puede entender como la integración de varios módulos o subsistemas funcionales que operan en coordinación tal como se puede observar en la Figura(1.4).

Normalmente, este tipo de sistemas cuenta con una etapa de adquisición de datos y entrenamiento que servirá para alimentar una base de conocimiento o reglas en las que se basará el módulo de inferencia y control autónomo. Asimismo, tanto el subsistema de adquisición de datos y entrenamiento como el subsistema de inferencia y control autónomo interactuan directamente con el subsistema de control y actuación del vehículo.

Los mayores esfuerzos se han enfocado principalmente al desarrollo del subsistema de inferencia y control autónomo ya que es el que define el rendimiento de un sistema de conducción autónoma en sí. Este subsistema, a su vez, puede ser analizado como un conjunto de varios módulos que interactúan entre sí de manera secuencial, como se puede apreciar en la Figura(1.5).

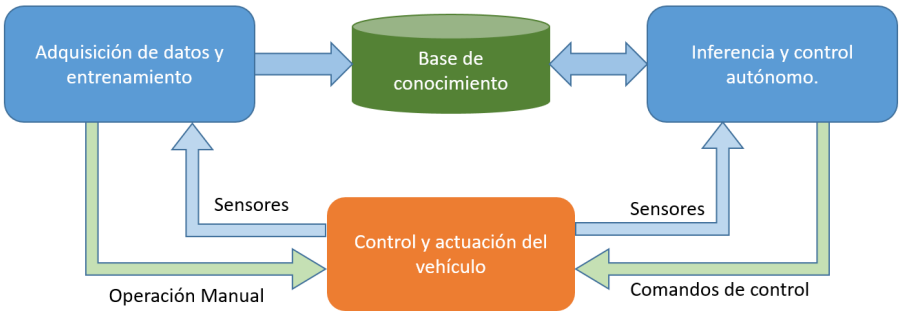

Figura 1.4: Arquitectura de un sistema de conducción autónoma. Fuente: Elaboración propia.

1.1.1.2. Sistemas de aprendizaje fin a fin

Tradicionalmente, los sistemas de aprendizaje requieren de múltiples etapas de procesamiento que interactuan entre sí, como se muestra en la Figura(1.5). Por su parte, los sistemas de aprendizaje fin a fin intentan condensar estas etapas de procesamiento y reemplazarlas usualmente con una sola red neuronal. Estos sistemas han demostrado ser altamente efectivos en contraste a los enfoques tradicionales, principalmente porque abstrae y resumen el diseño de las etapas intermedias de un sistema de aprendizaje tradicional con una sola etapa. La desventaja de los sistemas de aprendizaje fin a fin radica en la necesidad de grandes cantidades de datos de entrenamiento en comparación con los enfoques tradicionales, sin embargo, gracias a la gran disponibilidad de datos de entrenamiento y la accesibilidad de instrumentos y herramientas de adquisición de datos, esta desventaja no representa una dificultad de gran magnitud en el desarrollo de sistemas fin a fin.

Los sistemas de aprendizaje fin a fin se han explorado de manera exitosa en los últimos años, esto debido a la creciente disponibilidad de sistemas de cómputo de alta concurrencia, en especial las Unidades de Procesamiento Gŕafico de propósito general o GPGPU por sus siglas en inglés. Esta disponibilidad ha logrado que se puedan entrenar redes neuronales completas en una estación de trabajo que no consume demasiada energía. Una de las empresas pioneras en GPGPU es Nvidia con su herramienta CUDA, que ha permitido el desarrollo de algoritmos de entrenamiento e inferencia para redes neuronales de manera sencilla. Es precisamente Nvidia que ha demostrado que los sistemas de aprendizaje fin a fin pueden tener éxito con el desarrollo de un prototipo y arquitectura de vehículo autónomo [3].

1.2. Justificación del Proyecto

1.2.1. Justificación académica

Desde el punto de vista académico, el proyecto se justifica en el entendido del uso de técnicas y procedimientos de ingeniería para el análisis y diseño de un sistema de aprendizaje

"fin a fin" usando redes neuronales y una plataforma para el entrenamiento y despliegue del mismo. Tales técnicas y procedimientos incluyen la definición de la arquitectura de la red neuronal, el entrenamiento y el análisis del rendimiento de la misma. Así como también el dimensionamiento de los componentes de cómputo embebido para el prototipo y la implementación de los sistemas de control electrónico de bajo nivel para el mismo. Tales técnicas y procedimientos se corresponden de manera integral con los conocimientos adquiridos a lo largo de la carrera de Ingeniería Electrónica en sus distintas asignaturas.

1.2.2. Justificación tecnológica

Dada la creciente relevancia de los sistemas autónomos en la actualidad, el proyecto se justifica desde el punto de vista tecnológico dado que se presenta la aplicación de nuevas herramientas y plataformas de software para el desarrollo de sistemas de autónomos, visión por computador y redes neuronales convolucionales, que representan áreas vigentes en la investigación tecnológica hoy en día.

El sistema desarrollado se constituirá a su vez en una plataforma de desarrollo sobre el cual se podrá extender su funcionalidad y mejorar sus resultados usando herramientas de software de fácil acceso y aprendizaje presentando la posibilidad de continuar y extender la investigación en sistemas de conducción autónoma, robótica móvil, visión por computador y aprendizaje profundo.

1.2.3. Justificación técnica

Desde el punto de vista de las técnicas aplicadas, el proyecto se justifica dado que se pretende presentar una técnica alternativa al enfoque tradicional en el desarrollo de sistemas de aprendizaje, presentando el desarrollo de un sistema de aprendizaje fin a fin, que facilitará su análisis, diseño, entrenamiento y puesta en marcha en futuros proyectos de investigación y aplicaciones en distintas áreas de la ingeniería.

La propuesta de la nueva técnica de aprendizaje fin a fin representa un avance en relación al desarrollo de sistemas tradicionales por su impacto en el requerimiento de recursos y de conocimiento específico requerido.

1.3. Análisis de la problemática y planteamiento del problema

1.3.1. Análisis de la problemática

Se han estudiado diferentes enfoques para lograr solucionar la tarea de conducción autónoma para vehículos domésticos usando sistemas de aprendizaje. Normalmente, la salida del sistema se expresa como una serie de comandos de control de aceleración y dirección del volante del vehículo. Estos comandos se pueden obtener de diversas maneras dependiendo el nivel de robustez y abstracción que el sistema requiere. Muchos sistemas se basan en la fusión

de distintos tipos de sensores y fuentes de información como ser mapas satelitales, GPS, sensores láser y cámaras. La combinación de esta información es procesada y fusionada mediante distintos algoritmos de filtrado tales como el filtro de kalman. La característica de este tipo de sistema es que se puede expresar como una serie de etapas de procesamiento mediante el cual la información fluye y se transforma, cada una de las etapas es diseñada e implementada en base a conocimiento específico y con requerimientos y limitaciones específicas de la tarea que realiza tal como se puede apreciar en la Figura(1.5).

Si bien el enfoque anteriormente mencionado ha logrado conseguir importantes avances y resultados muy prometedores, involucra un gran esfuerzo a la hora de diseñar cada una de las etapas independientemente para luego hacer que funcionen todas juntas y cumplan la tarea asignada. Este proceso usualmente requiere de un equipo de expertos que sea capaz de realizar las tareas de diseño de las etapas o módulos del sistema y el de la integración de los módulos en un solo sistema funcional. Este enfoque, pese a que ha demostrado ser una forma efectiva de trabajo para diversos problemas, tiene la desventaja de requerir muchos recursos y tiempo para poder lograr un sistema funcional.

1.3.2. Planteamiento del problema

De acuerdo a lo establecido anteriormente, se puede considerar a la etapa de inferencia y control autónomo de un sistema de conducción autónoma como un sistema de procesamiento de información que consta de varias etapas secuenciales que transforman la información de acuerdo a parámetros previamente establecidos. Se debe tener en cuenta varios aspectos concernientes tanto al diseño como a la implementación de dichos tipos de sistemas.

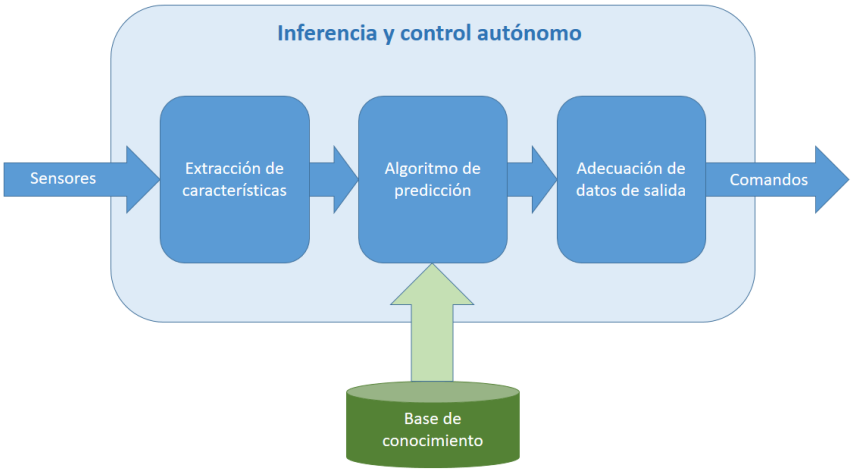

Figura 1.5: Componentes del subsistema de inferencia y control autónomo. Fuente: Elaboración propia.

En el área de visión por computadora para tareas de conducción autónoma, normalmente

se sigue el siguiente flujo en el desarrollo un sistema o prototipo:

1. **Extracción de características.** Esta etapa incluye el preprocesamiento y transformación de la imagen en un conjunto de características de distinta índole. Estas características se suelen llamar también descriptores y sirven para describir los aspectos más relevantes de la imagen para la tarea final, por ejemplo, la detección de bordes. La extracción de características también se usa para reducir la dimensionalidad inicial de la imagen a una más tratable y amigable con la capacidad de procesamiento computacional disponible. Las características o descriptores a usarse se definen manualmente por medio de conocimiento experto y se afinan de la misma manera.

2. **Algoritmo de predicción.** Esta etapa incluye típicamente un algoritmo de aprendizaje previamente entrenado con un conjunto de datos adecuado, permite realizar distintas tareas de alto nivel sobre los descriptores obtenidos de la imagen. Estas descripciones de alto nivel incluyen normalmente tareas de detección, clasificación o regresión. Los algoritmos de aprendizaje incluyen típicamente algoritmos básicos, tales como árboles de decisión, regresión lineal o máquinas de soporte vectorial ya que deben realizar la tarea de predicción en un conjunto de dimensionalidad relativamente baja (los descriptores).

3. **Adecuación de los datos de salida.** La información extraída de la anterior etapa debe procesarse para poder ser traducida a comandos de control que actuen directamente con las etapas de bajo nivel del vehículo, es decir la etapa de actuación y potencia. En esta etapa se suele incluir algún algoritmo de control realimentado para el control de motores así como también algoritmos de fusión de distintas fuentes de información para obtener finalmente una señal de comando para los actuadores.

Como se ha podido observar, el flujo de trabajo en un sistema de conducción autónomo se compone de varias etapas secuenciales que se deben realizar con conocimiento y experiencia específica en cada una de las mismas.

Por su parte, otra de las dificultades con este acercamiento, al reto de la conducción autónoma es el de la reducida flexibilidad del sistema. En otras palabras, si se quisiera modificar el sistema para agregar requerimientos o expandir la funcionalidad del mismo, se debe realizar una modificación a la etapa específica y evaluar el impacto de las modificaciones en todo el sistema en su conjunto. Esto dificulta de manera sustancial la reutilización de diversos componentes en sistemas similares.

Finalmente, la exagerada complejidad y conocimientos requeridos para poder implementar un sistema experimental de esta naturaleza hace prácticamente imposible su desarrollo por equipos de investigación pequeños o investigadores individuales. Dada la importancia y la potencialidad de los sistemas de conducción autónoma es escencial reducir esta dificultad de implementación y experimentación.

En conclusión, el desarrollo de un sistema de conducción autónoma presenta tres principales dificultades a la hora de ser abordado:

1. Conocimiento experto de cada una de las etapas involucradas en el sistema.

2. Poca flexibilidad en el diseño y la implementación del sistema una vez establecido y probado.

3. El tiempo y recursos necesarios para poder diseñar e implementar un sistema de tal naturaleza lo hace privativo para equipos de investigación pequeños o con poco presupuesto.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar un sistema de aprendizaje "fin a fin" capaz de generar de comandos de control de vehículos domésticos basado en visión artificial y redes neuronales convolucionales para facilitar el diseño e implementación de sistemas de conducción autónoma.

1.4.2. Objetivos Específicos

Para alcanzar el objetivo general será necesario:

- Estudiar los aspectos concernientes al desarrollo de sistemas de conducción autónoma y sistemas de aprendizaje.
- Analizar los requerimientos de un sistema de conducción autónoma capaz identificar y mantener su carril mientras se conduce.
- Diseñar la arquitectura de un sistema de conducción autónoma en base a los requerimientos previamente establecidos.
- Diseñar el subsistema de adquisición de datos y entrenamiento para tareas de conducción autónoma.
- Diseñar el subsistema de control y actuación para la conducción autónoma de un vehículo con características similares a las de un vehículo doméstico real.
- Diseñar el subsistema de inferencia y control autónomo basado en el uso de redes neuronales convolucionales.
- Analizar los resultados del entrenamiento e implementación del subsistema de inferencia y control autónomo.
- Realizar pruebas de rendimiento y análisis comparativos en el sistema implementado.

1.5. Alcance

El presente proyecto de grado cubre los siguientes aspectos dentro de su alcance:

- El enfoque del estudio de sistemas de conducción autónomos de vehículos terrestres con el modelo de Ackermann.
- Investigación de arquitecturas y plataformas de software para el diseño y despliegue de robots móviles y tareas de conducción autónoma.
- El desarrollo del sistema se contempla en el marco de un proyecto académico y, por tanto, será implementado usando herramientas de software comúnmente utilizadas en investigación de sistemas autónomos.

El alcance detallado previamente estará acotado a su vez por una serie de supuestos.

1.6. Límites

El sistema, por su parte, contará con ciertas restricciones detalladas a continuación:

- La tarea de conducción autónoma estará enfocada exclusivamente al seguimiento y mantención del carril basado en imágenes provenientes de una cámara sin considerar el reconocimiento e interpretación de otro tipo de información como señales de tránsito, cruces e intersecciones o la presencia de peatones, ciclistas, animales y otros objetos en la ruta.
- El prototipo a escala servirá solamente para un análisis superficial de la dinámica de un vehículo automotor doméstico tomando como punto de inicio modelos matemáticos simplificados y limitaciones de rangos de trabajo dentro de dichos modelos.
- El diseño de la arquitectura de la red neuronal estará orientado a tareas de aprendizaje supervisado y aproximación de funciones y limitado por la capacidad de procesamiento disponible en el momento de la realización del presente proyecto.

Capítulo 2

Marco Teórico

2.1. Sistemas de Conducción Autónoma

Un sistema de conducción autónoma es una combinación de varios componentes o subsistemas donde las tareas de percepción, toma de decisiones y operación de un vehículo son desarrolladas por un sistema electrónico en lugar de un conductor humano. Usualmente, un sistema de conducción autónoma incluye varios subsistemas de automatización que operan de manera conjunta y coordinada para poder tomar el control total o parcial del vehículo.

En algunas ocasiones, la automía del control se implementa de manera condicional, es decir, que el sistema toma el control del vehículo para ciertas situaciones pero no todo el tiempo como por ejemplo sistemas de estabilización de frenos o prevención de impactos. Este tipo de sistemas se ha ido desarrollando e implementando en vehículos comerciales de manera paulatina pero todavía no existe un vehículo completamente autónomo circulando por las calles o carreteras. Notese que los términos autonomía y automatización se usan de manera intercambiable en este contexto.

2.1.1. Niveles de Autonomía

Debido al creciente interés e inversión en el desarrolo de sistemas de conducción autónoma se ha establecido una manera de categorizar los niveles de automatización de la conducción por parte de la Sociedad de Ingenieros en Automoción (SAE, por sus siglas en inglés) en la que se definen seis niveles de automatización en vehículos terrestres, acuáticos y aéreos.

2.1.1.1. Nivel 0: Sin automatización

El conductor está en completo control de todas las funciones del vehículo en todo momento, no existe intervención de ningún sistema automatizado en el control. Sistemas de alerta de colisión o pérdida de carril entran en esta categoría.

2.1.1.2. Nivel 1: Conducción asistida

El conductor tiene el control del vehículo, pero el sistema puede modificar la aceleración o dirección del mismo. Los sistemas de control de velocidad de crucero caen en esta categoría.

2.1.1.3. Nivel 2: Automatización parcial

El conductor debe poder ser capaz de tomar el control del vehículo si ciertas se necesitan ciertas correcciones, pero ya no está en control de la aceleración y dirección del vehículo directamente. Es importante resaltar que desde los niveles 0 al 2 el conductor no puede estar distraido en ningún momento de la conducción. Los sistemas de parqueo automático representan un buen ejemplo de sistemas de Nivel 2.

2.1.1.4. Nivel 3: Automatización condicional

El sistema automatizado tiene el control del vehículo, tanto de la aceleración, dirección así como también del monitoreo del entorno bajo condiciones específicas. El conductor debe estar preparado para intervenir cuando el sistema así lo requiera, por tanto, se permiten distracciones ocasionales. Uno de los sistemas recientemente implementados que cae en esta categoría es el sistema *autopilot* de los vehículos de Tesla Motors.

2.1.1.5. Nivel 4: Automatización elevada

El sistema está en completo control del vehículo y la presencia humana ya no es necesaria, sin embargo, la operación autónoma del vehículo está limitada a condiciones específicas. Si las actuales condiciones del entorno sobrepasan las fronteras de rendimiento definidas, el vehículo puede desplegar un protocolo o secuencia de emergencia. Actualmente el desarrollo de vehículos autónomos o *self driving cars* se enfoca en este nivel.

2.1.1.6. Nivel 5: Automatización completa

El sistema está en completo control del vehículo y la presencia humana no es necesaria en absoluto. El sistema es capaz de proveer las mismas características que en el Nivel 4, pero en esta ocasión puede operar al vehículo en todas las condiciones. En este nivel, el conductor pasa a ser un pasajero en el vehículo. Actualmente, no existen sistemas que operen en este nivel.

La relación entre la responsabilidad del sistema y el conductor en los distintos niveles se puede apreciar en la Tabla 2.1:

2.1.2. Arquitectura de un sistema de conducción autónoma

2.2. Visión por computador

-

<table>
<tr><th>Nivel SAE</th><th>Denominación</th><th>Ejecución de aceleración y dirección</th><th>Monitoreo del entorno</th><th>Responsable en condiciones difíciles</th><th>Modos de conducción</th></tr>
<tr><td>0</td><td>Sin Automatización</td><td>Humano</td><td rowspan="2">Humano</td><td rowspan="3">Humano</td><td>Ninguno</td></tr>
<tr><td>1</td><td>Conducción asistida</td><td>Humano y sistema</td><td rowspan="3">Algunos Modos</td></tr>
<tr><td>2</td><td>Automatización parcial</td><td rowspan="4">Sistema</td><td rowspan="3">Sistema</td></tr>
<tr><td>3</td><td>Automatización condicional</td><td rowspan="2">Sistema</td></tr>
<tr><td>4</td><td>Automatización elevada</td><td>Varios Modos</td></tr>
<tr><td>5</td><td>Automatización completa</td><td>Todos los Modos</td></tr>
</table>

Cuadro 2.1: Niveles de automatización según SAE. Fuente: SAE

2.2.1. Procesamiento de imágenes

2.2.2. Filtrado

2.3. Redes Neuronales Artificiales

2.3.1. Aprendizaje Automático

El aprendizaje automático es un subcampo de la inteligencia artificial que intenta extraer patrones mediante un proceso de *aprendizaje* a partir de datos [4]. Este proceso de aprendizaje se define de acuerdo a una **tarea específica** T que intenta aprenderse en base a **experiencia pasada** E tomando como referencia una **medida de rendimiento** P . Dentro de esta definición, se puede listar varios ejemplos de tareas de aprendizaje que usualmente se resuelven usando los conceptos del aprendizaje automático o también llamado *machine learning*:

- **Un algoritmo de aprendizaje que pueda jugar ajedrez:**
  - **Tarea** T**:** Jugar Ajedrez.
  - **Medida de Rendimiento** P**:** Porcentaje de partidas ganadas contra el oponente.
  - **Experiencia** E**:** Información de varias partidas de práctica.
- **Un algoritmo de aprendizaje que pueda reconocer dígitos manuscritos:**
  - **Tarea** T**:** Reconocer y clasificar dígitos manuscritos dentro de una imagen.
  - **Medida de Rendimiento** P**:** Porcentaje de dígitos correctamente clasificados.
  - **Experiencia** E**:** Base de datos de imágenes de dígitos con sus etiquetas correspondientes.
- **Un algoritmo de aprendizaje que pueda reconocer la voz:**
  - **Tarea** T**:** Extraer una secuencia de palabras de una grabación de voz.

- **Medida de Rendimiento** P**:** Porcentaje de palabras correctamente predichas.
- **Experiencia** E**:** Grabaciones de voz con una transcripción correspondiente.

Esta definición de aprendizaje es lo suficientemente amplia como para englobar todas las tareas que el campo del aprendizaje automático intenta resolver en la actualidad. Sin embargo, debido a su naturaleza, se pueden clasificar las tareas de aprendizaje en tres grandes categorías que tienen características particulares: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

La diferencia entre estos tres tipos de problemas surge de la distinta naturaleza de la experiencia E disponible para el entrenamiento. A continuación, se procede a detallar cada uno de ellos.

2.3.1.1. Aprendizaje supervisado

En el caso de las tareas de aprendizaje supervisado, la experiencia constituye un conjunto de datos o *dataset* que contiene ejemplos con *características* y cada ejemplo está asociado con una *etiqueta*. Por ejemplo, un conjunto de datos de flores donde cada registro contiene datos de la flor (características) y la especie a la que pertenece (etiqueta). Dentro de los algoritmos que atacan problemas de aprendizaje supervisado se pueden encontrar 2 grandes categorías.

2.3.1.1.1. Clasificación Las tareas de clasificación tienen como característica el hecho de que la etiqueta de cada ejemplo en el conjunto de datos pertenece a una categoría o, en otras palabras, tiene una naturaleza discreta y finita. Por ejemplo, en el caso de la clasificación de las flores mencionado anteriormente, la etiqueta solamente puede pertenecer a un conjunto finito de especies de flores y cada ejemplo pertenece a una de estas especies.

2.3.1.1.2. Regresión En las tareas de regresión, las etiquetas pertenecen a un conjunto de números reales o de naturaleza contínua. En este caso, las etiquetas no se asocian con categorías sino más bien con otro tipo de variables. Un ejemplo muy conocido es el de la tarea de la predicción del precio de una casa en base a sus características, el precio de una casa no puede categorizarse porque representa un número que puede tener infinitos valores dentro de un rango definido.

En las tareas del aprendizaje supervisado, se puede considerar cada ejemplo como una descripción de una situación (características) en conjunto con una especificación (etiqueta), cada uno de los ejemplos dentro el conjunto de datos son eventos independientes y se pueden analizar por separado. En este sentido la tarea del algoritmo es generalizar la respuesta para casos no presentes en el conjunto inicial de datos.

2.3.1.2. Aprendizaje no supervisado

En las tareas del aprendizaje no supervisado la experiencia contenida en el conjunto de datos tiene la característica de no poseer ninguna etiqueta, por tanto, usualmente se intenta buscar una estructura escondida dentro el conjunto de datos o, dicho de otra manera, se

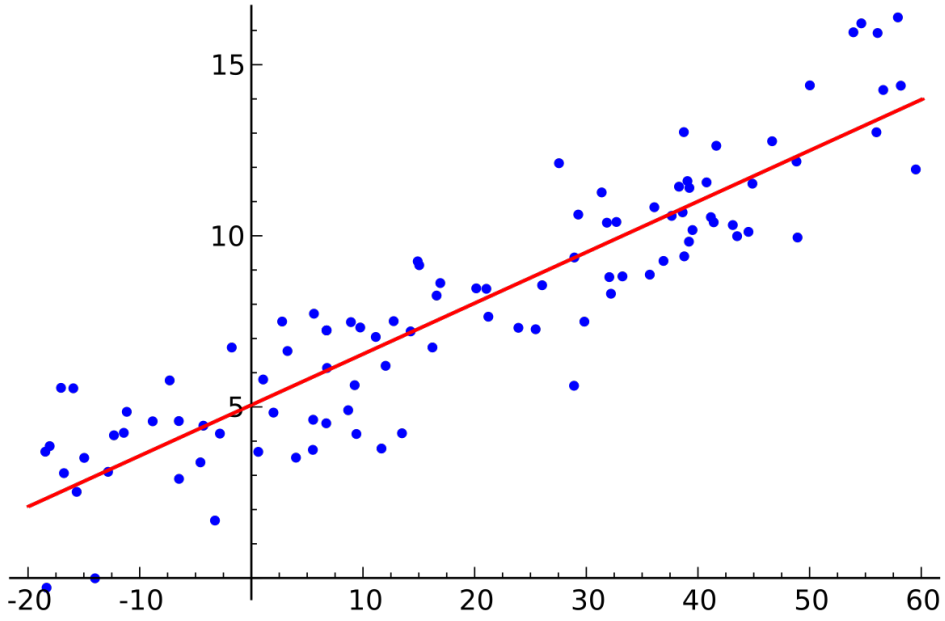

Figura 2.1: Regresión lineal, una tarea de aprendizaje supervisado. Fuente: [5]

buscan patrones que puedan presentarse en dichos datos. Estos patrones pueden aprovecharse para extraer información relevante de la naturaleza de datos de muy alta dimensionalidad, información que normalmente no es trivial de encontrar o visualizar por una persona. Entre algunas de las tareas más comunes dentro del aprendizaje no supervisado, se pueden listar:

2.3.1.2.1. Clustering Refiere a la tarea de separar y agrupar los datos en un número finito de conjuntos o *clusters*. Los *clusters* normalmente denotan una estructura oculta dentro de los datos y proporcionan información acerca de la similaridad entre ejemplos del conjunto de datos.

2.3.1.2.2. Reducción de dimensionalidad Uno de los problemas con las bases de datos y conjuntos de datos disponibles es que poseen una dimensionalidad bastante alta haciendo prácticamente imposible para un humano poder visualizar o encontrar patrones e información útil en los mismos. Este problema se suele tratar con algoritmos de reducción de dimensionalidad, en la que se encuentra una representación estimada de los datos pero con menos dimensiones. Uno de los algoritmos más conocidos y usados en esta categoría es el análisis de componente principal o PCA, por sus siglas en inglés, en el que se encuentra una representación de los datos en una menor dimensión usando proyecciones ortogonales.

2.3.1.2.3. Estimación de probabilidad Muchos conjuntos de datos son obtenidos de distintas fuentes y a lo largo de varios intervalos de tiempo, en este entendido, es muy útil conocer o aproximar la distribución de probabilidad de los datos para luego poder realizar

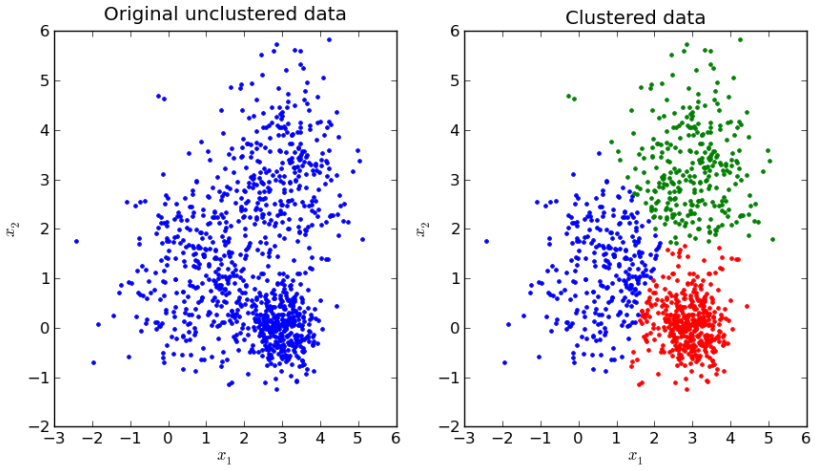

Figura 2.2: Clustering, una tarea de aprendizaje no supervisado. Fuente: [6]

predicciones o tratarlos con algún modelo en específico.

2.3.1.3. Aprendizaje por refuerzo

En las tareas de aprendizaje por refuerzo se toma en cuenta la interacción de un agente con su entorno y la forma en la que las acciones que toma dicho agente afectan a su entorno y se materializan en una recompensa o castigo [7]. Formalmente se pueden definir ciertos elementos que componen una tarea de aprendizaje por refuerzo:

- **Agente.** Es la entidad que interactúa con el entorno. El agente se comunica con el entorno mediante acciones.
- **Política.** Representan la forma de actuar del agente en base al conocimiento que ha adquirido.
- **Recompensa.** Es la función que define la efectividad del agente de cumplir el objetivo deseado, normalmente, el aprendizaje se enfoca en maximizar la recompensa que el agente puede obtener.

2.3.2. Aprendizaje Profundo

Dentro del campo de la inteligencia artificial y el aprendizaje automático se han implementado diversos tipos de algoritmos con éxito en los tipos de tareas de aprendizaje mencionados anteriormente. La base teórica y los detalles de implementación de estos algoritmos son muy variados, sin embargo, las redes neuronales artificiales han experimentado un incremento en el interés en la investigación y en las aplicaciones muy importante. Tal es el éxito de las mismas que se ha creado un subcampo exclusivo llamado aprendizaje profundo o *deep learning*.

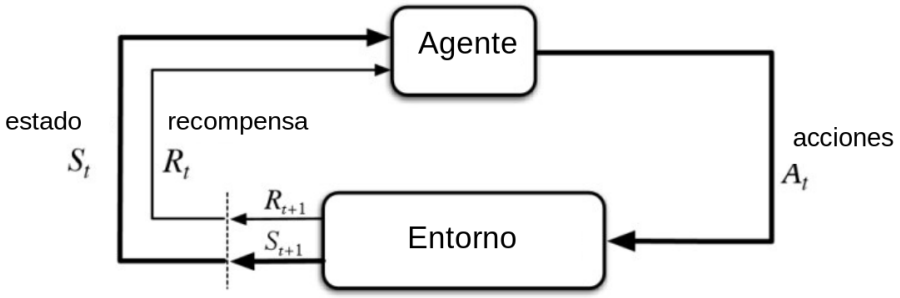

Figura 2.3: Esquema de una tarea de aprendizaje por refuerzo. Fuente: [8]

El aprendizaje profundo es un campo de la inteligencia artificial que se encarga de estudiar exclusivamente a las redes neuronales artificiales, sus componentes, arquitectura y aplicaciones. El impresionante rendimiento de estos algoritmos reside principalmente en el concepto de la representación que generan a partir de los datos que se procesan.

El aprendizaje profundo resuelve el problema del aprendizaje de representaciones al introducir representaciones que se expresan en términos de otras representaciones más simples. Además, permite a una computadora construir conceptos complejos a partir de conceptos más simples. Un ejemplo de la generación de estos conceptos o representaciones se puede apreciar en la Figura(2.4).

A continuación, se procede a definir los conceptos más importantes de redes neuronales artificiales con los cuales se podrá plantear la solución al problema de la conducción autónoma usando visión artificial.

2.3.2.1. Redes neuronales feedforward

Las redes neuronales feedforward o también llamadas perceptrón multicapa, son la base fundamental de los modelos de aprendizaje profundo. El objetivo de una red neuronal feedforward es el de aproximar una función f^*. Por ejemplo, para una tarea de clasificación, $y = f^*(\mathbf{x})$ mapea una entrada $\mathbf{x}$ a una categoría y. Una red neuronal feedforward define un mapeo $\mathbf{y} = f(\mathbf{x}, \mathbf{W})$ y aprende el valor de los parámetros $\mathbf{W}$ que resulten en la mejor aproximación[9].

Este tipo de modelos son denominados feedforward debido a que la información fluye a por la función siendo evaluada desde $\mathbf{x}$, a través de distintos cálculos intermedios definidos por f, hasta llegar a la salida $\mathbf{y}$. No existen conexiones de retroalimentación en las que la salidas del modelo se inyecten de nuevo a sí mismo. Las redes neuronales que poseen este tipo de conexiones de retroalimentación son denominadas redes neuronales recurrentes.

Para definir una red neuronal feedforward se puede comenzar definiendo un modelo basado en una combinación lineal en conjunto con una función no lineal que toma la siguiente forma:

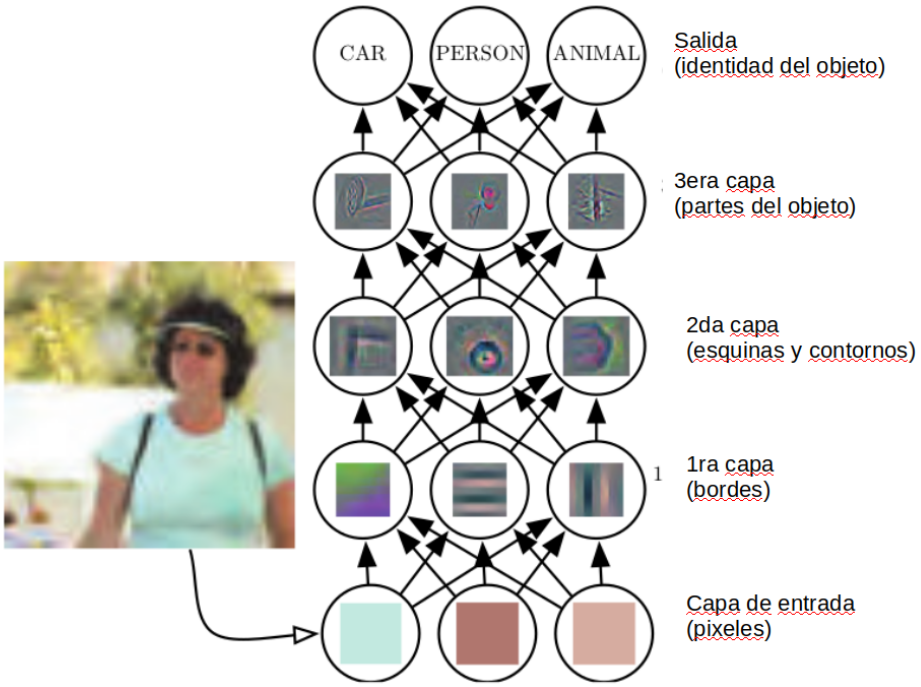

Figura 2.4: Ilustración de un modelo de aprendizaje profundo. Las representaciones se generan en las capas ocultas y corresponden con características de distintos niveles de complejidad. Fuente: [9]

$$y(\mathbf{x}, \mathbf{W}) = f\left(\sum_{j=1}^{M} w_j x_j\right) \tag{2.1}$$

donde $f()$ es una función de activación no lineal. Esto lleva al modelo básico de una red neuronal que puede ser descrita como una serie de transformaciones. Primero, se construyen M combinaciones lineales de las variables de entrada $x_1, \ldots, x_D$ donde D es la dimensión del vector de entrada $\mathbf{x}$:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.2}$$

donde $j = 1, \ldots, M$, y el superíndice (1) indican que los correspondientes parámetros se encuentran en la primera capa de la red. Los parámetros $w_{ji}^{(1)}$ se suelen conocer también con el nombre de *pesos* y los parámetros $w_{j0}^{(1)}$ con el nombre de *sesgos* o *biases*. Las cantidades a_j se conocen como *activaciones*, y cada una de ellas es luego transformada usando una función no lineal y derivable conocida como la *función de activación* $h()$ para luego obtener:

$$z_j = h(a_j) \tag{2.3}$$

Estas cantidades corresponden con la salida de la capa y también se suelen referir por el nombre de *unidades ocultas*. Las funciones no lineales $h()$ pueden escogerse dependiendo a diversos criterios de rendimiento o de comportamiento. Siguiendo a la Ecuación(2.1), las unidades ocultas se pueden volver a procesar con una combinación lineal y función de activación en una segunda capa:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.4}$$

donde $k = 1, \ldots, K$ y K corresponden con el número de salidas de la segunda capa. Finalmente, si se considera a esta capa como la capa de salida, podemos transformar las activaciones de la segunda capa con una función de activación. Normalmente, para una tarea de regresión, la función de activación es una función identidad, es decir $y_k = a_k$. Para una tarea de clasificación binaria, en cambio, la función de activación es una función sigmoide:

$$y_k = \sigma(a_k) \tag{2.5}$$

donde

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2.6}$$

Finalmente, se pueden combinar las etapas en una función general de la red que, para una salida sigmoidal, toma la siguiente forma:

$$y_k(\mathbf{x}, \mathbf{W}) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{2.7}$$

De esta manera, se define una red neuronal de dos capas a partir de la combinación lineal de las entradas y las unidades ocultas con los parámetros o pesos de la red transformados por funciones de activación no lineal. La arquitectura de la red definida en la Ecuación(2.7) se puede visualizar en la Figura(2.5) donde se observa claramente las relaciones que se han definido anteriormente en forma gráfica y la naturaleza del flujo en una sola dirección (feerforward) de los datos desde la entrada hasta la salida. En este caso, la red neuronal analizada es una red neuronal con una capa oculta.

2.3.2.2. Funcion de costo

La función de costo es una función que permite definir el rendimiento de una red neuronal con respecto a las predicciones esperadas a la salida. Un aspecto importante en el diseño de una red neuronal es la elección adecuada de la función de costo.

La función de costo o función de error, se define de manera similar al caso del ajuste de curvas. Es decir, se desea minimizar una suma de errores cuadráticos. Dado un conjunto de entrenamiento compuesto por un conjunto de vectores de entrada $\{\mathbf{x}_n\}$, donde $n = 1, \ldots, N$, junto con un conjunto de vectores objetivo $\{\mathbf{t}_n\}$ el objetivo es minimizar la función:

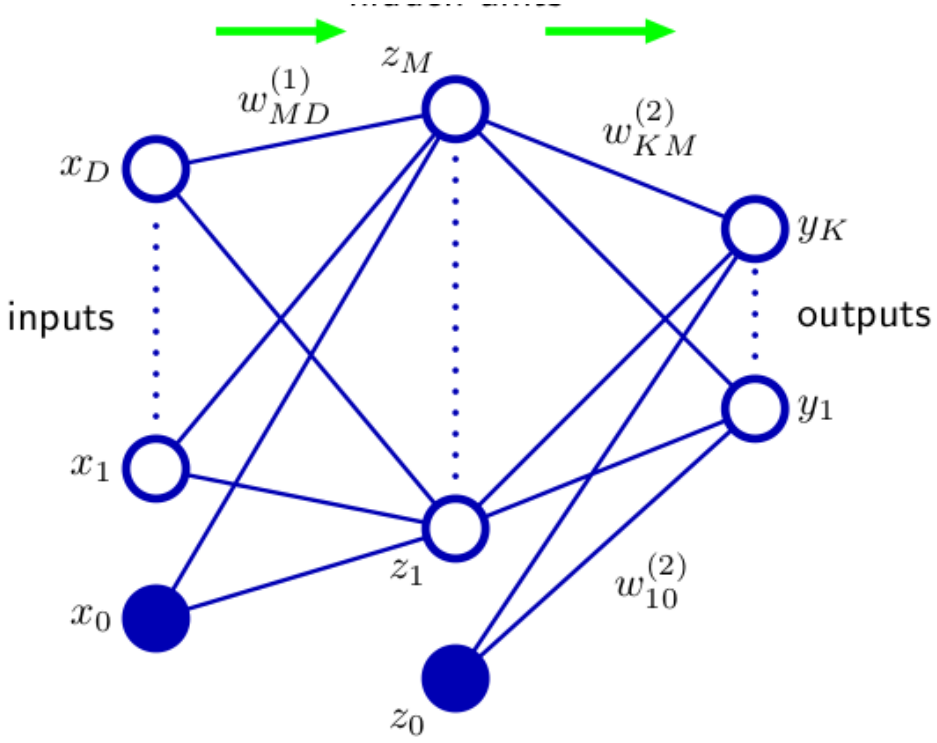

Figura 2.5: Diagrama de la red neuronal de dos capas correspondiente a la Ecuación(2.7). Fuente: [10]

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \tag{2.8}$$

donde el valor de $\mathbf{w}$ que minimice la función de error $E(\mathbf{w})$ será considerado como el mejor conjunto de parámetros sobre el cual el modelo puede generalizar.

2.3.2.3. Entrenamiento usando gradientes y retropropagación

Una vez definidos con claridad los componentes básicos de una red neuronal feedforward y cómo es el proceso del flujo de la información desde la entrada $\mathbf{x}$ hasta la salida $\mathbf{y}$, solamente queda especificar el procedimiento necesario para ajustar los parámetros o pesos de la red $\mathbf{W}$. Para este cometido, el método más usado es el de la retropropagación o *backpropagation*, popularizado a partir del paper de Rumelhart [11] y ampliamente usado en la actualidad en la mayoría de las redes neuronales. La retropropagación tiene el objetivo de encontrar los gradientes, en específico, se desea encontrar el gradiente de de la función de costo o error con respecto a los parámetros $\nabla_{\mathbf{W}} E(\mathbf{W})$. Si tomamos en cuenta que una red neuronal feedforward puede tener varias capas ocultas, la gradiente de la función de costo está en función de los parámetros y funciones de activación de cada capa, por tanto, se necesita usar la regla de la

cadena para poder encontrar estos gradientes intermedios.

Sea $y = g(x)$ y $z = f(g(x)) = f(y)$ dos funciones reales con argumento real, la regla de la cadena define lo siguiente:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.9}$$

Se puede generalizar la anterior expresión para casos fuera de una variable escalar donde $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, g mapea de $\mathbb{R}^m$ a $\mathbb{R}^n$ y f mapea de $\mathbb{R}^m$ a $\mathbb{R}$, si $\mathbf{y} = g(\mathbf{x})$ y $\mathbf{z} = g(\mathbf{y})$, entonces:

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i} \tag{2.10}$$

en notación vectorial, sería equivalente a lo siguiente:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z \tag{2.11}$$

donde $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ es el jacobiano $n \times m$ de g. Lo importante aqui es recordar que en una función multivariable, el gradiente proporciona la dirección hacia donde la función crece más rapidamente, esta intuición es fundamental para poder actualizar los pesos de la red en una etapa posterior.

El concepto fundamental de la retropropagación es encontrar estos gradientes de manera secuencial, partiendo desde la capa de salida hasta llegar a la capa de entrada.

2.3.2.3.1. Actualización de los parámetros de la red La importancia de hallar los gradientes de la red con respecto a los pesos reside en que son justamente los gradientes los que proporcionan la información de la evolución de los pesos con respecto de la función de error y en qué dirección se encuentra el mínimo. En la Figura(2.6) se puede ver cómo, para una combinación arbitraria de pesos $\mathbf{w}_C$ el gradiente de la función de error ∇E indica la direccion de máximo crecimiento de la superficie

La actualización de los parámetros dados los gradientes de la red se puede realizar de distintas formas, pero una de las más comunes es el algoritmo llamado *Descenso de gradiente*, donde, de forma iterativa, se va actualizando los pesos restando una medida del gradiente de la siguiente manera:

Dicho de otro modo, en el descenso de gradiente, se avanza un pequeño paso en la dirección opuesta al gradiente para avanzar hacia el mínimo:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \alpha \nabla E(\mathbf{w}^{(\tau)}) \tag{2.12}$$

donde el parametro $\alpha > 0$ se conoce como la razón de aprendizaje o *learning rate*. Después de cada actualización, el gradiente es recalculado para poder encontrar los nuevos pesos y el proceso se repite hasta encontrar el punto donde $\nabla E = 0$, lo que quiere decir que se ha llegado al mínimo. Usualmente, por la naturaleza de la superficie de la función de error y

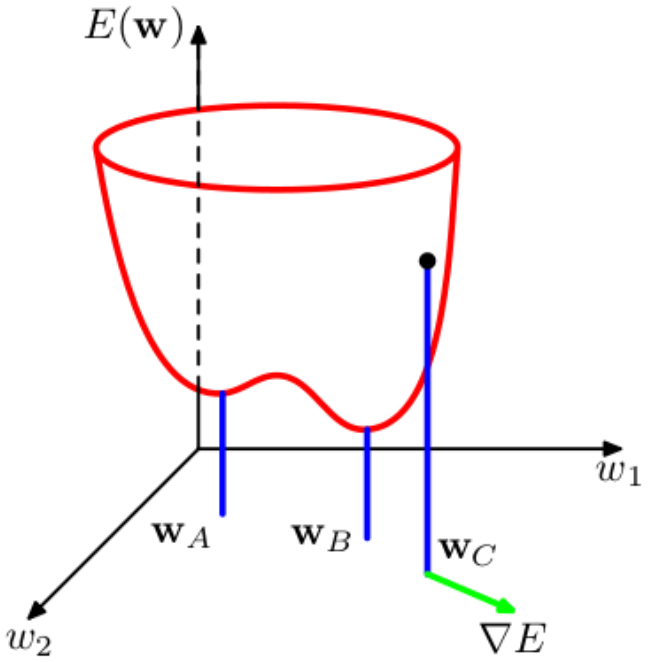

Figura 2.6: Visualización de la función de error $E(\mathbf{w})$ como una superficie. El punto $\mathbf{w}_A$ es un mínimo local y el punto $\mathbf{w}_B$ es el mínimo global. Fuente: [10]

diversos errores en el cálculo en una computadora, el número de iteraciones se limita en base a cierto parámetro de rendimiento.

Tradicionalmente, el algoritmo de descenso de gradiente se evalúa sobre todo el conjunto de datos de entrenamiento, este tipo de descenso de gradiente es llamado *batch gradient descent*, una *pasada* por todo el conjunto de entrenamiento se denomina usualmente una *época*. La principal desventaja es que la actualización de los pesos de la red neuronal solamente se actualiza una sola vez en cáda época; esto representa una dificultad en tiempo de procesamiento y memoria cuando se procesan conjuntos de datos muy grandes. Se han desarrollado versiones alternativas al *batch gradient descent* que mejoran su desempeño y son más tratables con conjuntos de datos muy grandes:

- ***Stochastic Gradient Descent.*** Refiere a que la actualización de los pesos en cada muestra procesada, es decir, que los pesos se actualizan n veces en una época.
- ***Mini-batch Gradient Descent.*** Es una mezcla del descenso de gradiente tradicional y el descenso de gradiente estocástico, en este caso, las muestras se alimentan al algoritmo en *mini-batches* o muestras de tamaño pequeño.

2.3.2.3.2. Adam Adam, abreviación de *Adaptive Moment Estimation*, es un algoritmo de optimización alternativo al descenso de gradiente que utiliza promedios tanto de los gradientes

como los momentos de segundo orden de los gradientes [12]. Dados los parámetros $\mathbf{w}^{(\tau)}$ y la función de error $E^{(\tau)}$, la actualización de parámetros de acuerdo a Adam se da mediante las siguientes expresiones:

$$\begin{aligned} m_w^{(\tau+1)} &= \beta_1 m_w^{(\tau)} + (1-\beta_1)\nabla_w E^{(\tau)} \\ v_w^{(\tau+1)} &= \beta_2 m_w^{(\tau)} + (1-\beta_2)(\nabla_w E^{(\tau)})^2 \\ \hat{m}_w &= \frac{m_w^{(\tau+1)}}{1-(\beta_1)^{(\tau+1)}} \\ \hat{v}_w &= \frac{v_w^{(\tau+1)}}{1-(\beta_2)^{(\tau+1)}} \\ w^{(\tau+1)} &= w^{(\tau)} - \alpha \frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon} \end{aligned} \tag{2.13}$$

Donde m_w representa a los momentos de primer orden y v_w representa a los momentos de segundo orden. Adam tiene los siguientes hiperparámetros:

- α es la razón de aprendizaje, análogo al caso del descenso de gradiente tradicional.
- β_1 es la razón de decaimiento exponencial para el estimado de los momentos de primer orden.
- β_2 es la razón de decaimiento exponencial para el estimado de los momentos de segundo orden.
- ϵ es un escalar pequeño usado para prevenir una posible división entre cero.

La introducción de los estimados de los momentos de primer y segundo orden de los gradientes permiten que se tome en cuenta el *ímpetu* con el que los gradientes cambian en el proceso de entrenamiento lo cual evita que pueda sobrepasar el mínimo y garantiza su convergencia a mayor velocidad que la de una actualización con un descenso de gradiente tradicional.

2.3.2.4. Funciones de activación

Se ha estudiado con mucho detalle la naturaleza de la función de activación $f()$, introducida en la Ecuación(2.1), analizando fundamentalmente dos aspectos: su contribución a la generación de representaciones internas en las capas ocultas de la red neuronal, así como también su comportamiento de sus gradientes con relación a la etapa de aprendizaje, tal como se ha visto en la Sección(2.3.2.3), donde se ha establecido claramente que el papel de la función de activación y sus gradientes es fundamental para el proceso de retropropagación y ajuste de los pesos de la red. Una guía con algunos criterios puede encontrarse en [13], de donde se rescatan los siguientes aspectos a la hora de escoger una función de activación:

- Que modele de forma no lineal la naturaleza de una representación interna fácil de interpretar.
- Que sea derivable en todo el rango de trabajo.

Tomando en cuenta lo anterior, se han generado diversas tendencias en la aplicación de funciones de activación y a continuación se analizarán las más relevantes para este proyecto.

2.3.2.4.1. Función sigmoide Esta función ha sido introducida en la Ecuación(2.6) y representa, históricamente, la función más utilizada en las primeras redes neuronales artificiales porque modela de manera aproximada, una respuesta característica de las neuronas del cerebro [14], pero sobre todo, porque posee las dos características mencionadas anteriormente: naturaleza no lineal y diferenciable. Otra característica llamativa, es que se puede interpretar a la salida como una función de probabilidad, pues sus valores van desde 0 a 1, como se puede apreciar en la Figura(2.7). Usualmente se incluye un parámetro adicional para controlar el *radio de activación* que hace que la función responda con más o menos sensibilidad a su entrada.

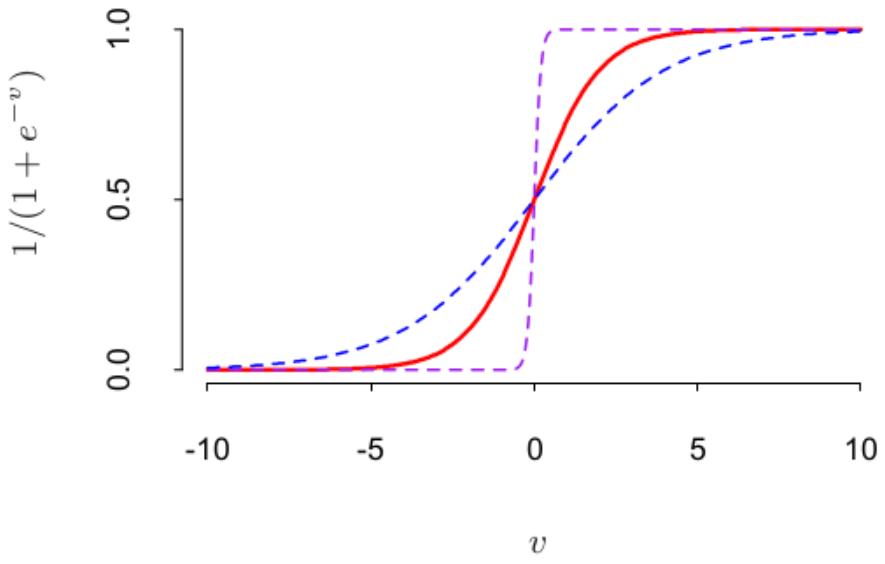

Figura 2.7: Gráfico de la función sigmoide $\sigma(v) = 1/(1+e^{(-v)})$ (curva roja), y dos variaciones con un radio de activación de la forma $\sigma(sv)$ con valores $s = 1/2$ (curva azul) y $s = 10$ (curva púrpura) . Fuente: [9]

Pese a las características anteriormente mencionadas, la función sigmoide tiene una gran desventaja: el llamado *desvanecimiendo de gradientes* [15]. Este fenómeno ocurre cuando la activación de una capa oculta tiene valores muy altos o muy negativos, se puede apreciar en la Figura(2.7), que para estos valores, la derivada tiene un valor muy pequeño, aproximándose a cero mientras más grandes sean los valores. Este fenómeno ocasiona que, mientras se realiza

la retropropagación de gradientes, dado que el gradiente tiene un valor muy bajo, el ajuste en los pesos sea mínimo, deteniendo así el aprendizaje de la neurona en la que ocurre el fenómeno.

Esta dificultad se presenta especialmente cuando la red neuronal se compone de varias capas ocultas y ha motivado el desarrollo de nuevas funciones de activación que no presenten esta limitación y puedan mantener valores de gradientes adecuados durante el entrenamiento.

Cabe resaltar que, pese a las dificultades con la propagación de los gradientes de la función sigmoide, ésta se suele utilizar bastante en la capa de salida, cuando se trata de clasificación binaria, ya que representa de manera adecuada la noción de probabilidad, lo cual es deseable en este tipo de modelos.

2.3.2.4.2. Función tangente hiperbólica La función tangente hiperbólica o $tanh()$ se define mediante la siguiente expresión:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.14}$$

Esta función también cumple con los requisitos de no linealidad y de ser derivable, pero en este caso, el rango de salida de la función $tanh()$ es desde -1 a 1. Considerando el concepto de función de probabilidad que ofrece la salida de la función sigmoide, la función tangente hiperbólica introduce el concepto de aceptación o negación de una premisa, donde una premisa aceptada se acerca a 1 y una premisa rechazada se acerca a -1, el valor de 0, indica indecisión. La función $tanh()$ se ha usado junto a su par sigmoide ampliamente en los inicios de las redes neuronales para las activaciones de las capas ocultas, pero, tal como se puede observar en la Figura(2.8), presenta la misma desventaja del desvanecimiendo de gradientes, dado que para valores muy grandes, la derivada se aproxima a cero.

En la actualidad, para implementaciones de redes neuronales profundas, el uso de las funciones sigmoide y tangente hiperbólica en las capas ocultas está ampliamente desaconsejado por los problemas anteriormente planteados. En su lugar se ha generado nuevos tipos de funciones de activación que presentan características bastante favorables para el aprendizaje y ajuste de pesos basados en gradientes.

2.3.2.4.3. Unidad Lineal Rectificada - ReLU Introducida por primera vez en el año 2010 por Nair y Hinton, las Unidades Lineales Rectificadas o ReLU, se propusieron para mejorar el rendimiento de un tipo especial de red neuronal: las máquinas restringidas de Boltzman (RMB, por sus siglas en inglés) [17], pero pronto, ganarían gran popularidad también en las redes neuronales feedforward y en redes neuronales convolucionales como se puede apreciar en [18].

La función ReLU se define de la siguiente manera:

$$g(x) = max\{0, x\} \tag{2.15}$$

Tal como se puede apreciar, la característica más importante de la función ReLU es su simplicidad pues es muy similar a una función identidad, la diferencia es que ReLU toma el

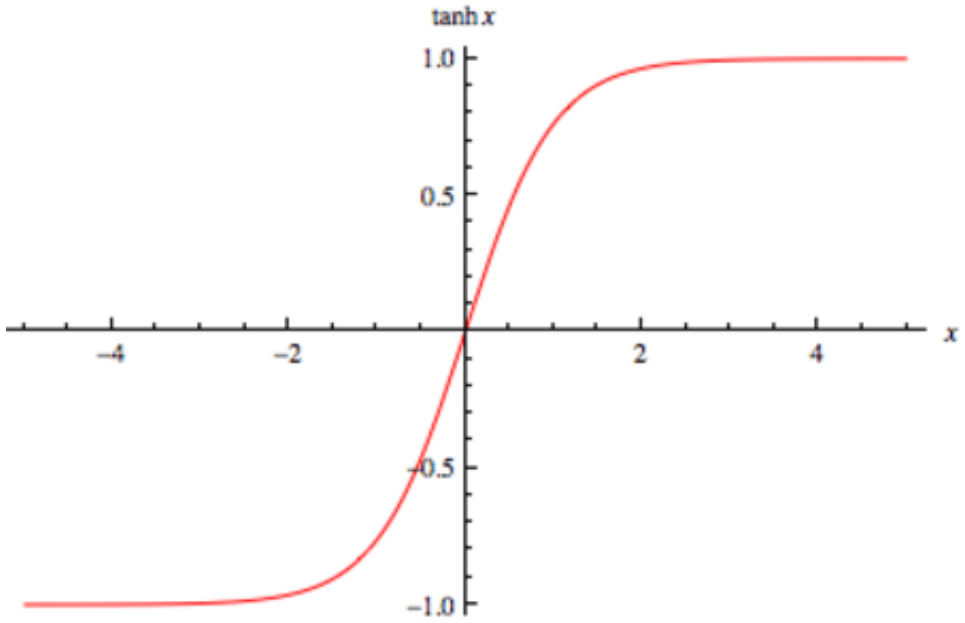

Figura 2.8: Gráfico de la función tangente hiperbólica $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Fuente: [16]

valor de cero para todos los valores de x que sean negativos. Esto resuelve el problema del desvanecimiendo de gradientes pues garantiza que el gradiente tenga un valor razonable si la entrada está activa y además sea consistente.

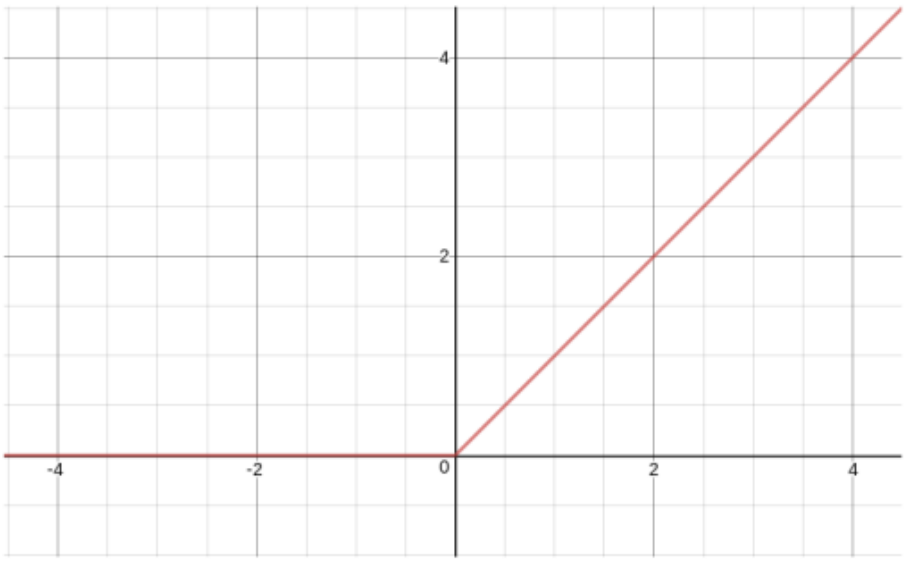

Figura 2.9: Gráfico de la ReLU. Fuente: [16]

Sin embargo, dado que la función ReLU tiene una gradiente nula para valores negativos es de vital importancia que se garantice la existencia de gradientes positivos, aunque sea pequeños durante la inicialización y las capas previas.

2.3.2.5. Diseño de Arquitectura e hiperparámetros de una red neuronal

Como se ha podido ver en las secciones anteriores, la característica más importante de una red neuronal es que es capaz de generar representaciones internas a partir de los datos y la retropropagación de los gradientes, lo que ha finalizado la era de la "ingeniería de características" donde se necesitaba conocimiento experto para elegir las representaciones adecuadas y cómo calcularlas, sin embargo, esto ha llevado a que el diseño de las redes se enfoque en otros aspectos como la arquitectura y otros parámetros externos que definen el modelo de la red. Por arquitectura de la red, se entiende a la estructura general de la red neuronal, el número de capas, el número de unidades o neuronas por cada capa y cómo las unidades y capas se conectan entre sí.

La gran parte de las redes neuronales estan organizadas en grupos de unidades llamadas capas, asimismo, las capas se ordenan de manera encadenada siendo cada capa una función de la capa que la precede, esto se ha establecido para una red de dos capas en la Ecuación(2.7) en su forma general, pero, se puede expresar el modelo de forma vectorial de la siguiente manera:

$$\mathbf{h}^{(1)} = g^{(1)}\left(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}\right) \tag{2.16}$$

$$\mathbf{h}^{(2)} = g^{(2)}\left(\mathbf{W}^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right) \tag{2.17}$$

En términos de redesn neuronales, la cantidad de unidades en cada capa se denomina el ancho o *width* y la cantidad de capas se denomina profundidad o *depth*.

2.3.2.5.1. Hiperparámetros Tanto la profundidad como el ancho de la red, son parámetros que se deben escoger en base a ciertos criterios, éstos parámetros son externos a los parámetros o pesos $\mathbf{W}$ de la red neuronal, por lo que se demoninan hiperparámetros. Aparte del ancho y profundidad, en el diseño de una red neuronal se consideran las siguientes variables:

- Razón de aprendizaje α.
- Funciones de activación.
- Tamaño del *mini-batch*.
- Número de épocas de entrenamiento.
- Algoritmo optimizador (Ejemplo: Descenso de gradiente).
- Función de costo.

Los hiperparámetros deberán ser escogidos de manera cuidadosa, pero debido a la gran complejidad y poca predictibilidad del comportamiento de una red neuronal profunda normalmente se eligen en conjunto con un proceso de prueba y error, analizando las curvas de aprendizaje y rendimiento en conjuntos de datos de validación y prueba.

2.3.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un tipo especializado de red neuronal que sirven para procesar datos de tipo "grilla"[9]. Algunos ejemplos de datos de tipo grilla que se pueden mencionar son los siguientes:

- **Series de tiempo.** Grilla de una dimensión tomados en intervalos regulares de tiempo.
- **Imágenes digitales.** Grilla de pixeles de dos o más dimensiones (Escala de grises, RGB).

Las también llamadas redes convolucionales, han demostrado un éxito impresionante en diversas aplicaciones prácticas especialmente en el campo de la visión por computador y el procesamiento de texto y lenguaje natural. El término "red neuronal convolucional" proviene del hecho de que en este tipo de redes neuronales se utiliza una operación matemática llamada **convolución**, siendo la convolución una operación lineal especializada para procesar datos de tipo grilla.

En los párrafos posteriores, se procede a describir la operación de convolución en el contexto de redes neuronales, pues, no siempre la definición de la misma corresponde con el concepto de convolución usado en distintos campos de la ciencia y la ingeniería.

2.3.4. Operación de convolución

En su forma más general, la convolución es una operación entre dos funciones reales y su definición se puede introducir usando el concepto de un promedio ponderado. Sea una función $x(t)$ dependiente del tiempo, tanto x como t son números reales; en este caso, la función x puede entenderse como una serie de medidas en un instante de tiempo t. Considérese una segunda función de ponderación $w(\tau)$ donde τ es la antiguedad de una medida. Si se aplica la función de ponderación en cada instante de tiempo, se puede obtener una nueva función definida por:

$$s(t) = \int x(\tau)w(t-\tau)d\tau \tag{2.18}$$

Esta operación es llamada la *operación de convolución* y es denotada tradicionalmente con un asterisco:

$$s(t) = (x * w)(t) \tag{2.19}$$

En el ejemplo de la ponderación, w debe ser una función de densidad de probabilidad válida, o la salida no podrá ser considerada como un promedio ponderado. Además, w también debe ser 0 para cualquier $t < 0$, esta última característica se denomina comunmente como

el principio de "causalidad". En general, la convolución está definida para cualquier función en la cual la integral anteriormente declarada esté definida y puede ser usada para otros propósitos aparte de promedios ponderados.

Hablando en términos de una red neuronal convolucional, el primer argumento (en el ejemplo, la función x) es comunmente referido como la **entrada**, y el segundo argumento (w, en el ejemplo) es referido como el **kernel**. La salida, a su vez, es normalmente referida como el **mapa de características**.

Por su parte, cuando se trata de señales digitales, como los datos en una computadora, el tiempo tiene una naturaleza discreta, es decir, que los datos estarán disponibles en intervalos regulares de tiempo. En este caso, el índice de tiempo t puede tomar solamente valores enteros y, entonces, es válido asumir que tanto x como w estan definidos solamente para valores enteros de t. De este modo, se puede definir la convolución discreta:

$$s(t) = (x * w)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau) \tag{2.20}$$

En el contexto de las aplicaciones de aprendizaje automático o, más específicamente, aprendizaje profundo, la entrada es usualmente un arreglo multidimensional de datos, y el kernel es usualmente un arreglo multidimensional de parámetros que se adaptan en el proceso de aprendizaje.

2.3.4.1. Procesamiento de imágenes con redes neuronales convolucionales

La operación de convolución se usa frecuentemente sobre datos con más de una dimensión. Las imágenes digitales son un perfecto ejemplo de un arreglo multidimensional de datos. Una imagen digital se representa mediante una matriz con filas y columnas, donde cada elemento se denomina pixel y contiene información acerca de la intensidad o luminancia, para una imagen en escala de grises o el nivel de color para distintos canales en una imagen a color. Si se toma el ejemplo de la imagen en escala de grises, se tiene una entrada o imagen bidimensional I con un kernel bidimensional correspondiente K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.21}$$

Dado que la convolución es conmutativa, se puede reescribir la ecuación 2.21 como:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.22}$$

Frecuentemente, la última fórmula es la más utilizada en librerías de aprendizaje profundo por su sencillez en la implementación en un sistema computacional, esto, dado que existe menos variación en el rango de valores válidos de m y n.

2.3.4.2. Aprendizaje de representaciones internas

Una de las preguntas clave en la visión por computador es el cómo generar una buena y significativa representación interna de una imagen, dado que la mayor parte de la imagen corresponde con pixeles que no aportan mucha información relevante a la tarea asignada. Por ejemplo, si se quisiera detectar un rostro dentro de una imagen, normalmente se suele encontrar una representación que ayude a aislar solamente las porciones de la imagen que pueden contener el rostro, tales como la búsqueda de contornos, bordes y características típicas de un rostro. Antes de la aparición de las redes convolucionales, estas representaciones se hallaban de manera manual y gracias al conocimiento de expertos en el área del procesamiento de imágenes. La definición de características y mapas de características era comunmente conocida como la *ingeniería de características*, en la cual los expertos creaban descriptores para tareas específicas con una gran inversión de tiempo en la sintonización fina de los mismos.

En contraste con el anterior enfoque, las redes convolucionales generan sus propias representaciones internas de manera automática gracias al aprendizaje de los parámetros de cada uno de los kernels que componen las distintas capas de la red neuronal. En principio, las redes convolucionales se inspiraron en el trabajo de Hubel y Wiesel sobre la corteza visual primaria de un gato[19]. En dicho trabajo, se logró identificar células simples que respondían de manera sobresaliente a distintas orientaciones con campos receptivos locales. Éstas células receptivas simples se pueden corresponder con los kernels de convolución usados en las redes convolucionales por la sencillez y la localidad de su campo de receptividad.

Posteriormente, las redes convolucionales ganaron una gran popularidad debido a su rendimiento en tareas de clasificación de imágenes y detección y reconocimiento de objetos en imágenes. El primer hito de su capacidad para procesar imágenes de manera efectiva fue en concurso de clasificación de imágenes de ImageNet, donde el equipo de Geoffrey Hinton logró sobrepasar el mejor resultado en precisión de clasificación por un gran márgen usando una arquitectura de red convolucional [18]. En este trabajo, se pudo apreciar con gran detalle las ventajas del enfoque del aprendizaje de representaciones internas en una red convolucional.

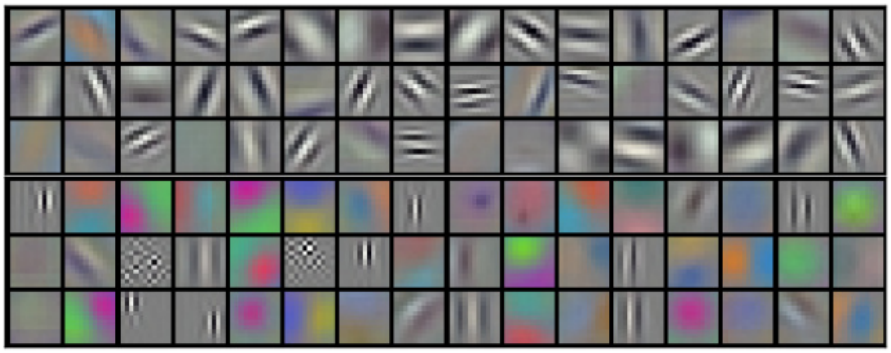

Figura 2.10: Kernels convolucionales de tamaño $11 \times 11 \times 3$ en la primera capa convolucional. Fuente: [18]

Tal como se puede apreciar en la Figura(2.10), en la primera capa convolucional, los kernels de convolución corresponden con representaciones básicas en una imagen como la búsqueda de bordes en distintas orientaciones, esto va acorde a lo establecido anteriormente en el modelo de la corteza visual de un gato. Puede decirse entonces que las redes convolucionales emulan, en cierto modo, al proceso biológico de visión en animales.

2.3.5. Sistemas de Aprendizaje Fin a Fin

2.4. Modelo cinemático del vehículo

-

2.4.1. Ecuaciones de movimiento

Bibliografía

[1] Y. LeCun, E. Cosatto, J. Ben, U. Muller, and B. Flepp, "Dave: Autonomous off-road vehicle control using end-to-end learning," Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www. cs. nyu. edu/yann/research/dave/index. html, Tech. Rep., 2004.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[4] T. M. Mitchell, *Machine Learning*. PN, 1990. [Online]. Available: https://www.amazon.com/Machine-Learning-Tom-M-Mitchell/dp/1259096955?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1259096955

[5] "Linear regression," Nov 2018. [Online]. Available: https://en.wikipedia.org/wiki/Linear_regression

[6] N. Viswarupan, "K-means data clustering – towards data science," Jul 2017. [Online]. Available: https://towardsdatascience.com/k-means-data-clustering-bce3335d2203

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[8] S. Bhatt, "5 things you need to know about reinforcement learning," Mar 2018. [Online]. Available: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006. [Online]. Available: https://www.ebook.de/de/product/5324937/christopher_m_bishop_pattern_recognition_and_machine_learning.html

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] H. N. Mhaskar and C. A. Micchelli, "How to choose an activation function," in *Advances in Neural Information Processing Systems*, 1994, pp. 319–326.

[14] S. Narayan, "The generalized sigmoid activation function: competitive supervised learning," *Information sciences*, vol. 99, no. 1-2, pp. 69–82, 1997.

[15] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[16] T.-y. Wang, "From perceptron to deep learning," Jan 2016. [Online]. Available: http://databeauty.com/blog/2018/01/16/From-Perceptron-to-Deep-Learning.html

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] Y. LeCun, K. Kavukcuoglu, C. Farabet *et al.*, "Convolutional networks and applications in vision." in *ISCAS*, vol. 2010, 2010, pp. 253–256.