Instituto Tecnológico de Chihuahua II

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Dron Autónomo usando Redes Neuronales de Convolución

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

Michael Reyes

DIRECTOR DE TESIS

DR. HERNÁN DE LA GARZA GUTIÉRREZ

CODIRECTOR DE TESIS

M.C. RAFAEL SANDOVAL RODRÍGUEZ

CHIHUAHUA, CHIH., 2 DE FEBRERO DEL 2021

D i c t a m e n

Chihuahua, Chih., 27 de enero 2021

M.C. MARÍA ELENEA MARTÍNEZ CASTELLANOS
COORDINADORA DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de este conducto el comité tutorial revisor de la tesis para obtención de grado de Maestro en Sistemas Computacionales, que lleva por nombre "DRON AUTÓNOMO USANDO REDES NEURONALES DE CONVOLUCIÓN", que presenta el C. MICHAEL REYES, hace de su conocimiento que después de ser revisado ha dictaminado la APROBACIÓN de la misma. Sin otro particular de momento, queda de Usted.

Atentamente
La Comisión de Revisión de Tesis.

DR. HERNÁN DE LA GARZA GUTIÉRREZ
Directora de tesis

M.C. RAFAEL SANDOVAL RODRÍGUEZ
Co-Director

M.C. ARTURO LEGARDA SÁENZ
Revisor

DRA.MARISELA IVETTE CALDERA FRANCO
Revisor

Tabla de contenido

Tabla de ilustraciones

1. INTRODUCCIÓN

1.1 Introducción

La autonomía de las cosas o robots que son capaces de aprender y mejorar sus conocimientos como los humanos ha sido una de las ideas principales de la inteligencia artificial por varios años, ya que, de esta manera, los equipos aprenderían y darían mejor desempeño haciendo innecesaria la intervención humana constante en muchos procesos. Hasta años recientes tomaron auge los algoritmos de Reinforcement Learning (RL, aprendizaje reforzado) que permiten a una máquina actualizar sus conocimientos y mejorar su funcionamiento por sí misma.

La compañía de autos Tesla es un claro ejemplo de una compañía en busca de autos autónomos que permitan a los usuarios disfrutar de un viaje sin el estrés y preocupación de estar al volante. Además de esta compañía, los algoritmos de RL se han usado en gran variedad de proyectos y sistemas computacionales. La razón de ello, es debido a que permite a los sistemas conseguir mejores resultados y en menor tiempo que a las personas, así se puede usar ese capital humano liberado, en otro tipo de tareas más urgentes y por consecuencia, dar mejor uso a los recursos.

En años recientes el uso de drones autónomos para realización de tareas se ha hecho más frecuente, no solo para diversión, sino porque pueden hacer distintos tipos de tareas que ayudan a la humanidad. Entre ellas pueden estar la entrega de mercancía, vigilancia de propiedades, búsqueda de personas en zonas peligrosas, etc. Los drones o cuadrópteros son muy usados para todo tipo de escenarios ya sean comerciales o recreativos.

Debido a lo anteriormente mencionado el Instituto Tecnológico de Chihuahua II ha visto la posibilidad de incluir el uso de drones con sistemas de navegación autónomos para sus nuevos proyectos. Se buscó iniciar con los primeros pasos en construir y programar un dron autónomo como parte de lo que vendría a ser un proyecto escalable y adaptable a otros tipos de proyectos que requieran de él. Así como, generar y usar herramientas computacionales y conocimientos relacionados con el amplio campo del Machine Learning (ML, Aprendizaje de máquina).

1.2 Planteamiento del problema

Ya existen varios proyectos de drones autónomos haciendo uso de algoritmos de Machine Learning y visión computacional para distintas tareas. Como lo son, drones de seguidores de caminos en bosque para la búsqueda de personas. O también drones entrenados para las carreras de drones para mejorar el desempeño y velocidad, incluso se empieza a trabajar con la entrega de mercancías a domicilio por medio de drones. El Instituto Tecnológico de Chihuahua II no cuenta con ningún dron para uso de la institución, ni mucho menos un dron que aprenda por sí mismo a realizar una tarea, de manera que será la base para desarrollos futuros de mucho mayor alcance e importancia.

Esto generó la necesidad en la institución de adquirir o crear un dron autónomo adaptable que permita a los proyectos hacer uso de él y sentar las bases de un proyecto escalable que puede ser usado por más estudiantes de posgrado para ayudarlos en sus investigaciones y así

mismo, dar a la institución más herramientas para la proposición y creación de proyectos diversos. Lo cual lleva a desarrollar la siguiente pregunta de investigación:

¿Se puede crear un dron con un sistema de navegación autónomo haciendo uso del Reinforcement Learning?

1.3 Alcances y Limitaciones

- El dron podrá moverse del punto A al unto B, pero no podrá hacer maniobras complicadas.
- Estará limitada el área de navegación del dron a una habitación, lo cual implica que no se le permitirá hacer vuelos muy lejanos.
- El dron no podrá alejarse grandes distancias de la computadora que está dando sus órdenes de vuelo.
- El dron contará con la habilidad de aprender y actualizar su base de conocimientos previamente entrenada.
- El sistema de reinforcement learning será entrenado en un simulador.
- Adquisición del dron para la investigación, así como modificaciones que se vean necesarias al dron para su funcionamiento adecuado.
- Se requerirá la creación de un mapa que asemeje al entorno de vuelo.

1.4 Justificación

- Demostrar que es posible usar Reinforcement Learning y Aprendizaje de Máquina para entrenar a un dron a realizar maniobras en un ambiente controlado.
- La institución busca un proyecto que le permita ser usado como base para implementación de mejoras o fusión con otros proyectos.
- El dron proveerá de código abierto que dará a la institución acceso a él para su estudio posterior, o para hacer uso del código en distintos proyectos.

La simple palabra dron atrae, tiene un efecto especial no solo en los jóvenes sino en personas de más edad. Aunque son asociados en su mayoría con tareas de diversión y en aspectos de obtención de imágenes aéreas para fines variados, tienen un gran potencial para ser usados para tareas de apoyo y de servicio a la sociedad. Los drones usados para estas actividades o tareas, siempre cuentan con un operador que los controla con su central de mando. Sin embargo, en el Posgrado del Instituto, se ha contemplado el aplicar técnicas de inteligencia artificial, específicamente Reinforcement Learning para lograr que un dron aprenda a realizar una tarea de manera autónoma, considerándose como primer reto, que aprenda a desplazarse desde un punto origen (A), a un punto destino (B). Esto será el inicio de un camino hasta lograr que aprenda a realizar tareas cada vez más complejas y de mayor alcance en distancia y tiempo.

Con lo anterior se estará posicionando al Instituto como uno de los pioneros a nivel nacional en usar el aprendizaje reforzado (RL) para entrenamiento de drones autónomos.

De manera adicional se debe considerar que estas tecnologías no son algo fácil de adquirir o aprender. Uno de los principales usos de este proyecto se puede atribuir a sus partes, por ejemplo el código empleado para obtener video del simulador puede ser empleado en otros proyectos relacionados a la obtención de material u observación y testeo de redes neuronales, así mismo el uso de los scripts para control autónomo pueden usarse en proyectos donde se busquen entornos adecuados que permitan a estudiantes experimentar y observar resultados pertinentes a sus investigaciones, únicamente las herramientas de control y obtención de datos marcan su utilidad, el simulador puede ser cambiado. De igual manera, la red neuronal de convolución (CNN), lo cual permitirá probar diferentes entornos y practicar diferentes configuraciones entre otras necesidades que se presenten.

Lo anterior ya marca un cambio en el aprendizaje que esta investigación contribuye a los alumnos, a los profesores y algunos investigadores que inicien en el mundo de la inteligencia artificial, ahora también se cuentan con extensas librerías para la implementación de redes neuronales, como lo son keras, tensorflow, tflearn. Este proyecto contará con la implementación de una CNN y también con algunas de las técnicas usadas para controlar el dron de manera autónoma. Esta información servirá de ayuda a otros investigadores que busquen implementar proyectos nuevos en relación a drones controlados por conexión wifi, y comandos básicos que permitirán observar movimientos en el mundo real usando la cámara del dron para análisis a través de visión artificial.

1.5 Objetivos

En este apartado se definen los objetivos de la investigación.

1.5.1. Objetivo General

Desarrollar un algoritmo de Reinforcement Learning para su implementación en un dron, dándole la capacidad de aprender a volar y poder realizar movimientos en un espacio y llegar a su destino sin la necesidad de ser controlado por un humano.

1.5.2. Objetivos Específicos

- Desarrollo del algoritmo de Reinforcement Learning adecuado al dron.
- Desarrollo de simulador para aprendizaje de un dron virtual.
- Adquisición de datos por parte del dron, para la mejora de su base de conocimientos en tiempo real.

1.6 Hipótesis

Es posible lograr que un dron aprenda a moverse de un punto de origen (A) a un punto destino (B), sin el control humano, por medio de la técnica de Reinforcement Learning.

2. ESTADO DEL ARTE

2.1 Antecedentes

Las Redes Neuronales Convolucionales (CNN por sus siglas en inglés) tienen un largo impacto en el campo del aprendizaje de máquina (machine learning), específicamente en la aplicación de aprendizaje profundo (deep learning) para visión computacional. En el presente proyecto se utilizó la red neuronal “AlexNet”[4], una red convolucional muy conocida por haber ganado la competencia de reconocimiento de imágenes “ImageNet LSVRC-2012”, mostrando una diferencia de desempeño entre el primer y segundo lugar del 10.9%.

Hoy los MAV (vehículos aéreos móviles) comerciales son en su mayoría tele operados, usan GPS para navegación y tienen habilidad limitada para automáticamente evitar obstáculos. Sin embargo, la comunidad investigadora ha hecho pasos significativos hacia la introducción de más procesamiento basado en visión para facilitar el sorteo de obstáculos y estimación del entorno y de las variables del vehículo. [2]

El uso de AI e imágenes para manejar vehículos en el mundo real continúa creciendo. Por ejemplo, Monfort y un grupo de investigadores de NVIDIA ha usado una CNN para mapear pixeles de un video capturado por una sola cámara frontal. Con un mínimo uso de información de entrenamiento de los humanos, la red neuronal aprendió a conducir en el tráfico sobre los caminos locales y las carreteras con o sin marcas de carril o barandillas usando las matemáticas de los ángulos de dirección como señales de entrenamiento.

Se ha logrado el desarrollo de un sistema navegación basado enteramente en visión artificial, donde se usa una Resnet modificada para detectar colisiones, eventualmente el dron fue capaz de viajar en ambientes diferentes totalmente basándose en las colisiones entrantes, ejecutando vueltas y otro tipo de acciones. [3]

Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness

Una investigación que realizó la corporación NVIDIA sobre el uso de drones en la identificación de caminos para búsqueda de personas, esta investigación requirió un dataset de más de 20,000 imágenes, fue concebida con la intención de ayudar a agentes de policía en búsquedas de personas perdidas en las montañas o bosques.

La captura de las imágenes fue realizada usando un tubo con 3 cámaras Go Pro, mientras el investigador recorría distintos caminos de tierra en diferentes locaciones, para poder capturar suficientes imágenes, después de eso se elaboró una red neuronal profunda Resnet-18 que permitía detectar la orientación del dron y si estaba dentro del camino o fuera de él. El dron recibió la red entrenada y usando una cámara recibía suficiente información del ambiente para poder mantenerse dentro de la línea debido a que recibía un puntaje negativo si se salía lo cual lo hizo mejorar y empezar a usar el ángulo de giro y la información para seguir la línea.

Más adelante en la investigación se incluyó un láser que hacia al dron mantenerse pegado al suelo sin rebasar cierta altura, y de igual manera se incluyó la librería llamada YOLO que usa redes neuronales para identificar personas en el trayecto y tomar esa información para sobrevolar a la persona en el camino y seguir su curso. Para la realización de ese proyecto se usó un pixhawk para el piloto automático, y una NVIDIA jeston tx 1 para el procesamiento de las imágenes, la cual regresaba la información y daba órdenes al pixhawk sobre los movimientos a realizar. [11]

En la investigación cuyo nombre es Dronet: Learning to Fly by Driving [3], se usó un dron Parrot Beboop 2 que permite la conexión por wifi. Con la conexión de wifi es posible pasar el video al programa que contiene el algoritmo de visión artificial. La investigación primero se realizó tomando muestras usando una bicicleta. Uno de los investigadores se montó en una bicicleta tomando video, de esta manera se pudo hacer un dataset que pudiera ser usado para poder entrenar una red neuronal tipo Resnet que los investigadores habían estado trabajando, y con la cual fueron capaces de modificarla de tal manera que la red neuronal regresaba el ángulo de dirección y detección de colisiones, logrando que el dron fuera capaz de moverse solo. Se hicieron pruebas en las calles y con autos del mundo real, donde el dron demostró habilidades de movimiento autónomo.

La investigación Playing Atari with Deep Reinforcement Learning es bastante interesante y ayudo mucho en la forma la que se visualiza reinforcement learning, donde a través de 4 frames la red neuronal de convolución fue capaz de extraer varias características, como lo son movimientos, velocidad, aceleración, entre otras estas características fueron dadas al algoritmo de reinforcement learning que a través del uso DQN era capaz de aprender más rápido el uso de cómo aplicar movimientos. [10]

Como se puede observer por el articulo titulado Gaming Machine Learning: Game simulations are driving improvements in machine learning for autonomous vehicles and others devices. Los investigadores de Inteligencia artificial hablan de como el uso de entornos simulados en este caso videojuegos ayudo mucho a mejorar la forma en la que los algoritmos de inteligencia artificial eran capaces de aprender por si solos de esta manera podían hacer pruebas en ambientes controlados para poder probar sus algoritmos pre entrenados en la vida real, uno de los juegos que ellos usaron fue el de GTA V el cual permite el control de tráfico, control del clima, entre otros aspectos que permitieron a los investigadores poner a prueba los algoritmos y ver como estos podían aprender en condiciones que no se hayan en el mundo real con frecuencia. [1]

3. MARCO TEÓRICO

Para la primera parte de este proyecto fue necesario apoyarse en diferentes herramientas y plataformas buscando su integración y participación conjunta. Enseguida se mencionan brevemente dichas aplicaciones.

3.2 Anaconda Distribution [12] [13]

Anaconda es una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto.

Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, las librerías de Ciencia de datos y Conda. Todas éstas se instalan de manera automática y en un procedimiento muy sencillo.

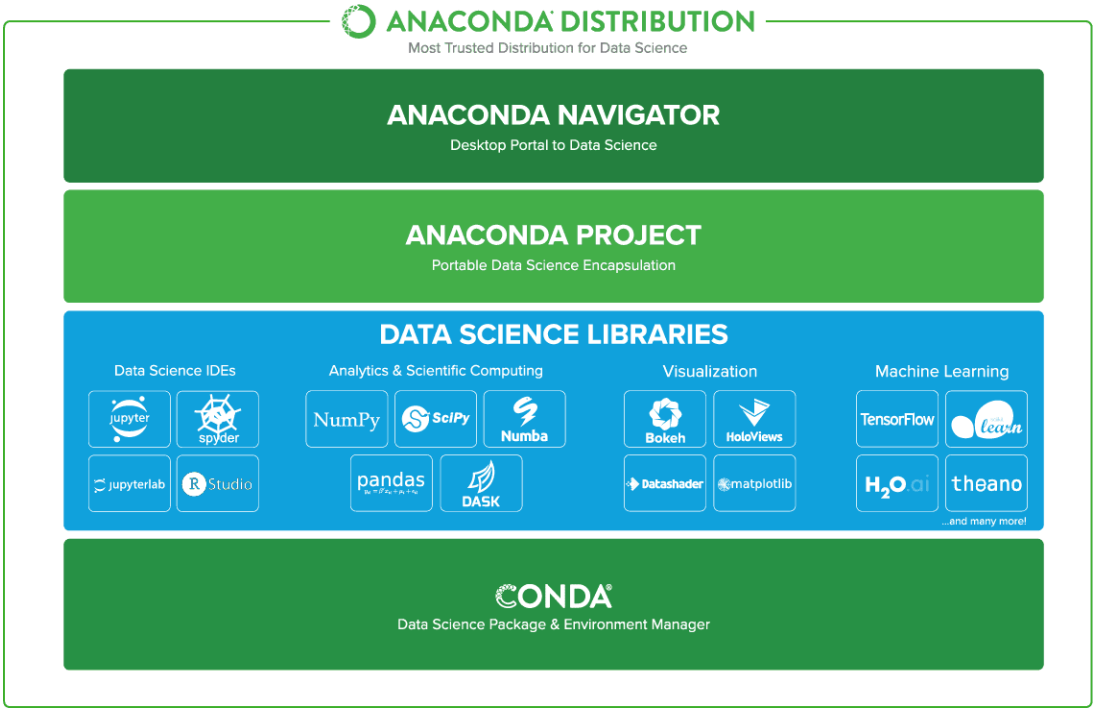

Ilustración 1: anaconda distibution.

Cuando instalamos Anaconda tendremos disponibles todas estas herramientas ya configuradas, podemos gestionarla mediante la interfaz gráfica de usuario Navigator o podemos utilizar Conda para la administración mediante la consola. Puede instalar, eliminar o actualizar cualquier paquete Anaconda con unos pocos clics en Navigator o con un solo comando de Conda.

Características de Anaconda Distribution

Esta Suite para la Ciencia de datos con Python cuenta con una gran cantidad de características entre las que podemos resaltar las siguientes:

- Libre, de código abierto, con una documentación bastante detallada y una gran comunidad.
- Multiplataforma (Linux, macOS y Windows).
- Permite instalar y administrar paquetes, dependencias y entornos para la ciencia de datos con Python de una manera muy sencilla.
- Ayuda a desarrollar proyectos de ciencia de datos utilizando diversos IDE como Jupyter, JupyterLab, Spyder y RStudio.
- Cuenta con herramientas como Dask, numpy, pandas y Numba para analizar Datos.
- Permite visualizar datos con Bokeh, Datashader, Holoviews o Matplotlib.
- Una gran variedad de aplicaciones relacionadas con el aprendizaje de máquina y los modelos de aprendizaje.
- Anaconda Navigator es una interfaz gráfica de usuario GUI bastante sencilla, pero con un potencial enorme.
- Puede gestionar de manera avanzada paquetes relacionados a la Ciencia de datos con Python desde la terminal.
- Brinda la posibilidad de acceder a recursos de aprendizaje más avanzados.
- Elimina problemas de dependencia de paquetes y control de versiones.
- Está equipado de herramientas que permiten crear y compartir documentos que contienen código con compilación en vivo, ecuaciones, descripciones y anotaciones.
- Permite compilar Python en código de máquina para una ejecución rápida.
- Facilita la escritura de algoritmos paralelos complejos para la ejecución de tareas.
- Cuenta con soporte para computación de alto rendimiento.
- Los proyectos son portables, lo que permite compartir proyectos con otros y ejecutar proyectos en diferentes plataformas.
- Simplifica de manera acelerada la implementación de proyectos de ciencia de datos.

Conda enviroments

Un conda enviroment es un directorio que contiene una colección específica de paquetes de Conda que ya se encuentran listados. Por ejemplo, se podría tener un enviroment con NumPy 1.7 y sus dependencias, y otro enviroment con NumPy 1.6 para pruebas de tipo legacy. En caso de algún cambio en algún enviroment, los otros enviroments no se verán afectados. Se puede activar o desactivar fácilmente los enviroments, lo cual permite hacer el cambio de los mismos. También se pueden compartir los enviroments con alguien más al otorgarles una copia del archivo ***enviroment.yaml.***

¿Por qué usar Conda virtual enviroments?

- Control sobre las elecciones de compatibilidad binaria.
- Utilización de estándares de lenguajes nuevos, como C++ 17.
- La necesidad de librerías más allá de lo que el sistema Python ofrece.

- Manejo de paquetes de lenguajes diferentes a Python en el mismo espacio.

3.1 Unity (Motor de Videojuegos Multiplataforma) [14]

Unity es una herramienta de desarrollo de videojuegos creada por la empresa Unity Technologies. Que no engloba únicamente motores para el renderizado de imágenes, de físicas de 2D/3D, de audio, de animaciones y otros motores, sino que engloba además herramientas de networking para multijugador, herramientas de navegación NavMesh para Inteligencia Artificial o soporte de Realidad Virtual.

Esta herramienta tiene múltiples opciones para crear ambientes virtuales que perite controlar agentes y así mismo creación de escenarios personalizados para probar agentes inteligentes, existen varios ejemplos de ello, en algunas de las investigaciones comentadas adelante se usa el juego de GTAV que fue usado para enseñar a un algoritmo de machine learning como detectar señales de tráfico, otros vehículos y personas.

También unity cuenta con varias librerías como lo es unity-tello que permite la conexión de los drones DJI Tello para su uso controlado con la herramienta de Unity, donde se puede controlar un dron real usando comandos en unity, también se puede hacer uso de ML-agents que fue una librería que usa Tensorflow para la creación de redes neuronales entre otros algoritmos de inteligencia artificial que permiten controlar agentes en Unity y realizar distintas pruebas.

3.2 TFlearn [15]

TFlearn es una biblioteca modular y transparente de aprendizaje profundo construida sobre Tensorflow. Fue diseñado para proporcionar una API de nivel superior a TensorFlow para facilitar y acelerar los experimentos, sin dejar de ser totalmente transparente y compatible con él.

La librería contiene las funciones necesarias para la implementación de redes neuronales y que en este caso permitió la implementación del esquema de la AlexNet. Esta librería proveyó las funciones necesarias para guardar la red entrenada y a partir de una nueva imagen de entrada, obtener la acción que el agente debe realizar.

3.3 Lenguaje de programación Python [16]

Python presenta una serie de ventajas que lo hacen muy atractivo, tanto para su uso profesional como para el aprendizaje de la programación. Entre las más interesantes desde el punto de vista didáctico tenemos:

- Python es un lenguaje muy expresivo, es decir, los programas Python son muy compactos: un programa Python suele ser bastante más corto que su equivalente en lenguajes como C. (Python llega a ser considerado por muchos un lenguaje de programación de muy alto nivel).
- Python es muy legible. La sintaxis de Python es muy elegante y permite la escritura de programas cuya lectura resulta más fácil que si utilizáramos otros lenguajes de programación.
- Python ofrece un entorno interactivo que facilita la realización de pruebas y ayuda a despejar dudas acerca de ciertas características del lenguaje.

- El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información muy rica para detectarlos y corregirlos.
- Python puede usarse como lenguaje imperativo, procedimental o como lenguaje orientado a objetos.
- Posee un rico juego de estructuras de datos que se pueden manipular de modo sencillo. Estas características hacen que sea relativamente fácil traducir métodos de cálculo a programas Python.

Los lenguajes de programación no permanecen inmutables a lo largo del tiempo: evolucionan. Python no es una excepción. A partir de la experiencia con una versión del lenguaje y de la influencia que ejercen otros lenguajes sobre los programadores, hay una presión constante por hacer que el lenguaje ofrezca nuevas capacidades o simplifique el modo en el que se expresan ciertos cálculos. Python fue diseñado inicialmente por Guido van Rossum a partir de su experiencia colaborando con otros en el desarrollo de un lenguaje experimental: ABC. La World Wide Web aparecía al poco de crearse la primera versión de Python y ayudaba a poner en contacto a miles de programadores en todo el mundo. La elegancia de Python, unida a la aparición de un nuevo medio de comunicación entre especialistas, hizo que un lenguaje que no provenía de la academia o la industria tuviera un éxito inusitado. Hablamos de los años 90 del pasado siglo, década en la que fue tomando fuerza el concepto de «software libre».

3.4 Importar OpenCV y numpy [17]

OpenCV fue iniciada en Intel en el año 1999 por Gary Bradsky, y la primera versión vino en el año 2000. En el 2005, Opencv fue usado por Stanley, el vehículo que gano el gran reto de DARPA en el 2005. OpeCV hoy en día da soporte a varios proyectos de algoritmos relacionas a la inteligencia artificial.

OpenCV es una biblioteca de código abierto que se ha utilizado en varias plataformas para la detección de patrones, y aplicación de diferentes filtros que son utilizados para el procesamiento de imágenes. En algunos casos se han hecho adaptaciones de la librería para poderse usar en otros lados, como es el caso de Unity donde se puede comprar la librería para realizar operaciones usando el motor de juegos. Pero de igual manera hay una versión gratuita para Python que usa Numpy para hacer distintas operaciones equivalentes a C++ que fue donde originalmente se creó.

OpenCV fue de gran ayuda en captura de las imágenes, en este caso cada *frame* capturado se regresaba como un objeto de OpenCV.

Para instalar OpenCV y Numpy se deben seguir los siguientes pasos:

- Una vez instalado Anaconda hay que instalar OpenCV. Para instalar una biblioteca desde Anaconda hay que abrir un terminal (en el caso de Windows una consola de Anaconda) y teclear:

```
conda search OpenCV
```

- Este comando mostrará las versiones de OpenCV disponibles en los repositorios oficiales de Anaconda.

- Esta vez, en el listado se encuentra OpenCV 3.1 y 3.2. Entonces, hay que elegir un repositorio de OpenCV 3.2 válido para nuestra plataforma. Por ejemplo, si nuestra plataforma es Linux, Windows o Mac de 64 bits, para instalar OpenCV 3.2 podemos utilizar el repositorio "conda-forge/OpenCV3". Para instalar utilizando este repositorio escribiremos:

```
|anaconda install -c conda-forge OpenCV3
```

- Una vez instalado OpenCV podemos comprobar que funciona correctamente abriendo una consola de python e importando la biblioteca. Para ello, desde la consola de Anaconda tecleamos: python. En ese momento se presentará un mensaje de bienvenida a Python. Enseguida podremos teclear los siguientes comandos. Si no se observan mensajes de error es que todo ha ido bien, pero si aparecen mensajes de error deberán leerse y entenderse para buscar una solución.

```
>>> import cv2
>>> import numpy as np
```

3.5 Red Neuronal de Convolución (CNN) [18]

Las redes neuronales convolucionales son algoritmos de Deep Learning que están diseñados para trabajar con imágenes, tomando éstas como *entrada*, asignándole importancias (pesos) a ciertos elementos en la imagen, para así poder diferenciar una imagen de otra. Este tipo de redes, son las causantes del gran auge reciente de la Inteligencia Artificial ya que entre otras cosas han contribuido en el desarrollo y perfeccionamiento del campo de Visión por computadora.
Las redes convolucionales contienen varias *hidden layers* (capas ocultas), donde las primeras capas, puedan detectar líneas, curvas, geometrías sencillas que luego se van integrado para diseños más complejos conforme avanza a capas más internas y así se van especializando hasta poder reconocer formas complejas como un rostro, siluetas, etc. Las tareas comunes de este tipo de redes son:

Detección o categorización de objetos, clasificación de escenas y clasificación de imágenes en general.

La red toma como entrada los pixeles de una imagen. Si tenemos una imagen con apenas 28×28 pixeles de alto y ancho, eso equivale a 784 neuronas. Y eso es si sólo tenemos 1 color (escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales (Red, Green, Blue) y entonces usamos 28x28x3 = 2352 datos en un arreglo tridimensional, el cual se suministra al primer bloque de *kernel* o filtros.

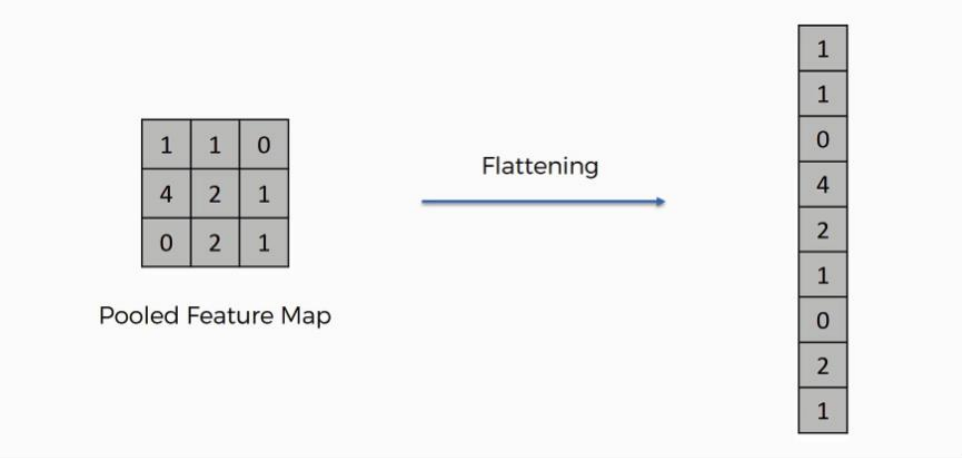

Ilustración 2: Flattening de una red neuronal de convolución.

Pero antes es necesario normalizar la “data”, es decir que nuestros pixeles que ahora tienen valores entre 0 y 255, tengan valores entre 0 y 1, podemos lograrlo dividiendo cada uno de los pixeles entre el valor más alto que estos tienen es decir 255.

Kernel

El *kernel* en las redes convolucionales se considera como el filtro que se aplica a una imagen para extraer ciertas características importantes o patrones de ésta.

Por ejemplo, si tenemos una imagen como la siguiente:

Aplicando el filtro o *kernel* se mostrará de la siguiente forma.

Ilustración 3: Imagen sin filtro.

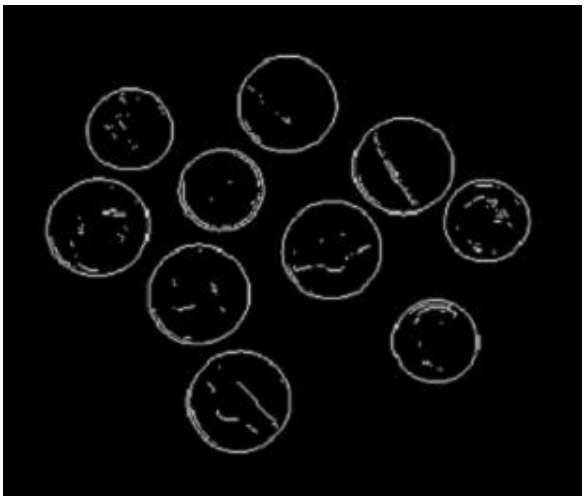

Ilustración 4: Imagen con filtro.

Entre las características importantes para lo que sirve el *kernel* son detectar bordes, enfoque, desenfoque, entre otros. Esto se logra al realizar la convolución entre la imagen y el *kernel*.

Hay que decir que esos *kernels*, en las redes de convolución, NO están definidos y es lo que la red termina por ajustar conforme se entrena.

Convolución

Uno de los procesos más distintivos de estas redes son las convoluciones. El cual consiste en tomar un grupo de píxeles de la imagen de entrada e ir realizando un producto escalar con un kernel. El kernel recorrerá todas las neuronas de entrada y obtendremos una nueva matriz, la cual será una de las *hidden layers*. En el caso de que la imagen sea de color se tendrá 1 *kernel* con profundidad de 3, cuyas operaciones se harán con la sección correspondiente de la matriz tridimensional de datos.

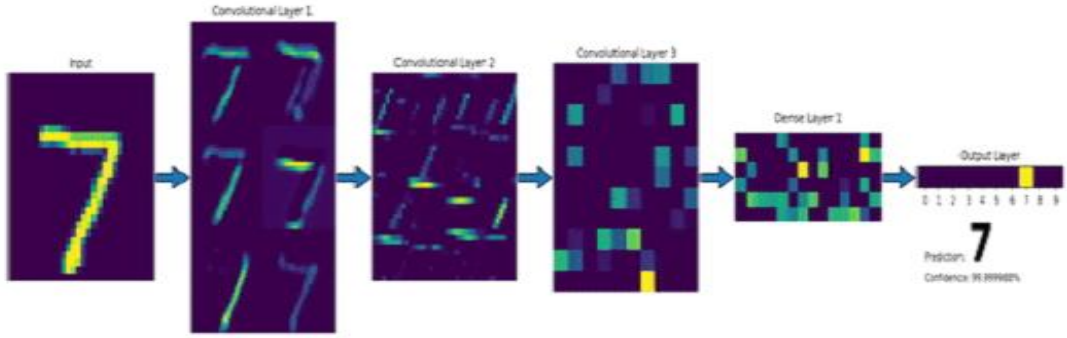

Ilustración 3: Proceso que siguió la red neuronal de convolución.

Para este proceso tenemos una imagen y el *kernel*, con la finalidad que el filtro o *kernel* recorra toda la imagen (pixel). Por lo general, el *kernel* es de menor tamaño que la imagen. La convolución permite multiplicar el *kernel* con la porción de imagen escogida, se realiza la multiplicación tal como indica la imagen y luego el *kernel* se va desplazando, por esta razón es un proceso iterativo.

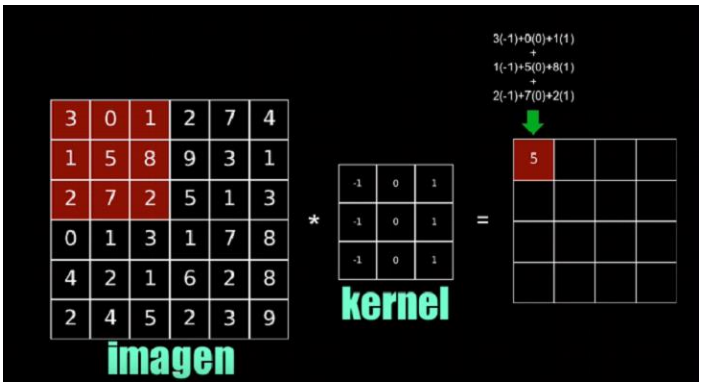

Ilustración 4: multiplicación de imagen con el kernel.

Siguiente desplazamiento y se vuelve a multiplicar.

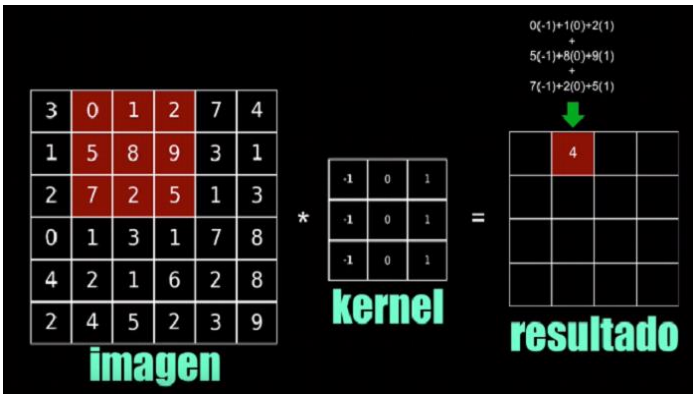

Ilustración 5: multiplicación de imagen con el kernel 2 parte

Y así sucesivamente hasta procesar todos los datos.

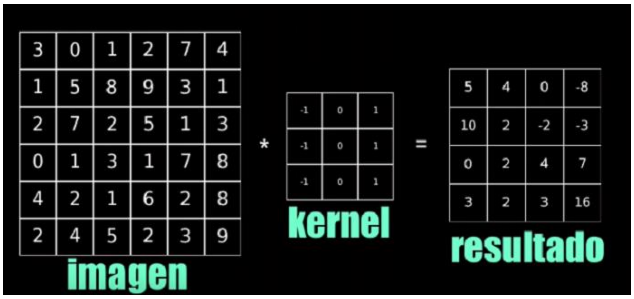

Ilustración 6: Resultado final de la multiplicación de la imagen con el kernel.

Las imágenes de color se representan con 3 planos R (red), G (green), B (blue) y al combinarse se ven todas las mezclas de colores.

Por lo que ahora el filtro también será de 3 planos.

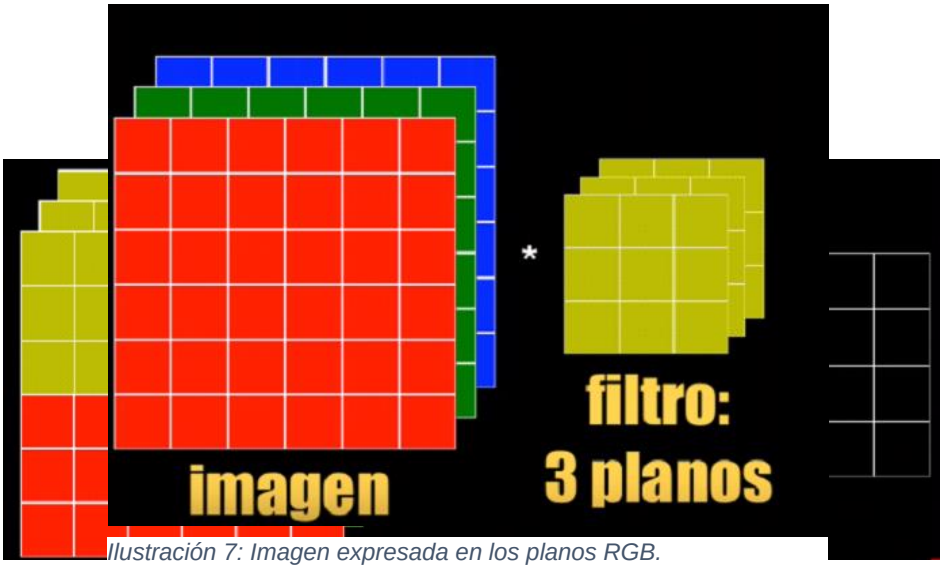

Ilustración 7: Imagen expresada en los planos RGB.

Ilustración 8: Imagen RGB multiplicada por el kernel.

La convolución suele ser similar al de escala de grises, pero ahora el filtro de 3 capas le corresponde a un pixel (capa) correspondiente. Aunque la imagen de entrada es de 3 planos, la imagen resultante será de 1 solo plano

Se hace el proceso iterativo mencionado anteriormente, hasta completar todo

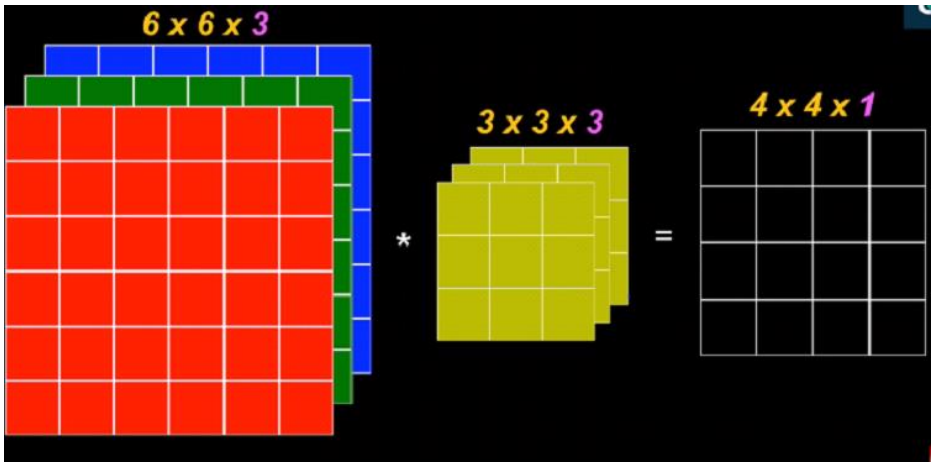

Ilustración 9: Proceso de la imagen multiplicada por el kernel.

Padding

Es una operación que se usa en las redes convolucionales. El *padding* se aplica agregando píxeles de valor cero alrededor de la imagen original.

Tiene dos usos:

El primero es para que al realizar la convolución la imagen resultante sea de igual tamaño que la imagen original.

El segundo es cuando se tiene información relevante en las esquinas de la imagen, al realizar la convolución sin el *padding*, el filtro hace que los valores centrales sean tomados en cuenta

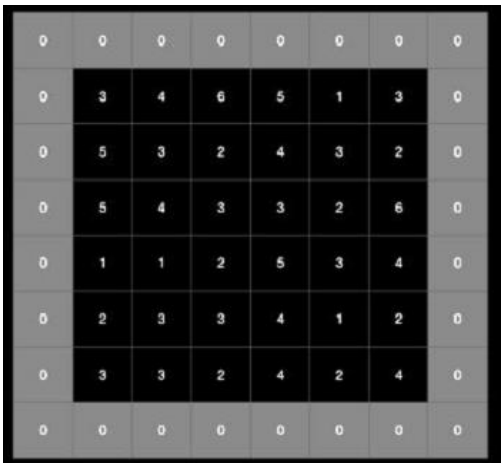

Ilustración 10: Aplicación del padding.

más veces que los de las esquinas y para hacer la corrección de esto es que se añaden esos ceros circundando a los datos.

3.6 Machine learning [19]

“En definitiva, el ‘machine learning’ es un maestro del reconocimiento de patrones, y es capaz de extraer inferencias de nuevos conjuntos de datos para los que no ha sido entrenado previamente”, explica José Luis Espinoza, científico de datos de BBVA México.

El término *machine learning* se usó por primera vez en 1959, en ese tiempo apenas se empezaban las bases de algunos algoritmos con los estudiantes del MIT, Marvin Minsky y Dean Edmons, los cuales crearon el primer programa informático capaz de aprender de la experiencia para salir de un laberinto. Esto permitió que más científicos buscaran el crear nuevos programas que fueran capaces del reconocimiento de patrones y agarrar experiencia de eso y mejorar.

La teoría del *machine learning* es bastante simple, el programa o algoritmo analiza el estado actual y revisa las opciones que tiene disponible, en base eso realizará cálculos para entender o predecir los posibles futuros de realizar ciertas acciones, tomando en cuenta la situación actual, en base a eso lanzará un porcentaje de éxito con cada acción a tomar, y seleccionará la acción. Cada acción tendrá un valor negativo o positivo, que le permitirá al algoritmo entender qué acciones conllevan a mejores recompensas y cuáles no. Esto permite a esta clase de programas informáticos actualizar su base de conocimientos y mejorar en sus predicciones.

DISTINTOS ALGORITMOS DE 'MACHINE LEARNING'

Los algoritmos de *Machine Learning* se dividen en tres categorías, siendo las dos primeras las más comunes:

Aprendizaje supervisado: estos algoritmos son entrenados en ambientes controlados, esto quiere decir que se preparan pruebas con sus resultados en el cual el algoritmo es capaz de recibir un resultado para comparar su decisión e ir mejorando. Una vez entrenado el sistema con algunas muestras, se procede a ir dando datos que validen su aprendizaje, pero dado que ya el algoritmo es capaz de detectar el patrón no hallará dificultad en reconocer casos similares. Esto puede ser como la identificación de ciertas plantas, el algoritmo ya toma como referencia, el color, número de pétalos, etc., información que le permitirá reconocer plantas que cumplan con estas características porque es un patrón repetitivo.

Aprendizaje no supervisado: estos algoritmos son distintos, dado que no reciben información con sus resultados para comparar si están bien o mal, solo reciben puntos negativos en caso de equivocarse o positivos en caso de acertar, esto se refiere a casos como los que se presentan en juegos, donde uno avanza hacia adelante y tiene que elegir entre 2 puertas, una puerta tiene una recompensa y la otra un castigo, si se repite el experimento ya se tomó en cuenta que puerta tiene cada resultado, de esta manera se va actualizando el algoritmo para detectar patrones y mejorar sus resultados. Este algoritmo trata de extraer características y hacer clasificaciones de los datos.

Aprendizaje por refuerzo: estos algoritmos son diferentes dado que son capaces de extraer características y hacer clasificaciones, pero son capaces de mejorar de manera gradual, dado que es como los humanos que aprendemos bajo experiencia, este tipo de aprendizaje es más acelerado. El algoritmo tendrá una base de conocimientos con los mejores resultados obtenidos en una corrida, y empezará a explotar esas opciones y también probará nuevas alternativas, y cada opción remplazará en su base datos si fue mejor que su muestra, de esta manera el algoritmo usa el refuerzo que es visto como experiencia para seguir mejorando sus habilidades en un recorrido.

3.7 Win32 [20]

Cuenta con una serie de funciones programadas en el lenguaje C que permiten el control o llamado de funciones del sistema Windows, esto quiere decir que para nuestro caso particular que requerimos el uso de la librería para usar ciertas funciones, nos ahorra muchos problemas. Se usarán funciones como *GetAsyncKeyState ()* que nos permitirá obtener el estado del teclado, y poder conocer la tecla que fue pulsada, también tenemos funciones para la obtención de información de la pantalla del sistema operativo. Esto se logra con las herramientas contenidas *win32gui, win32ui, win32con.* En estas librerías podemos obtener varias funciones que nos permitirán hacer obtención de las métricas de la pantalla, obtención de imágenes, entre otras funcionalidades nativas del sistema Windows.

La librería fue muy útil en el proyecto para diferentes acciones que ayudaron en el sistema, y su posterior uso y elaboración.

4. ANALISIS Y DISEÑO DEL SOFTWARE

4.1 Técnicas de recopilación de información

La técnicas para la recopilación de información fueron la entrevista, la observación y recopilación documental, dado que aparte de las consultas con el director de tesis asignado (Dr. Hernán de la Garza Gutiérrez), también se recurrió a visualización de información relevante, estudios de trabajos similares que atribuyeron usos importantes, consulta de un diseñador de videojuegos, entre otros medios bibliográficos, mucha de la información fue proporcionada desde el diseño del entorno gráfico para probar el software, hasta la programación involucrada y la elaboración de los datasets. Al final se usa la observación de comportamientos para ajustes necesarios acorde a las necesidades que iban surgiendo con los diferentes elementos usados, donde el diseñador de juegos proporcionó asistencia en la modificación del entorno virtual.

4.2 Captura, definición y validación de requisitos

Captura: Al entrevistar al director de tesis se determinó la necesidad de construir un simulador, definiéndose los requisitos que debería contar el programa.

Los requisitos son:

- Deberá poder ser viable para poder probar una red neuronal de convolución
- La habilidad de procesar video en vivo
- La habilidad de ser autónomo
- Ser capaz de tomar decisiones en base a la información otorgada por el video
- La codificación deberá ser realizada en Python para poder ser usado por otros alumnos.
- Escribir la tesis que hable del proceso que se llevó a cabo para desarrollarlo.

Definición: se probarían varios métodos, recolectando información de los proyectos existentes, pruebas de pilotos automáticos y los problemas que se llevarían a cabo para poder dar información que sería utilizada para que los alumnos pueden tomar esos procesos y mejorarlos.

Se realizará el simulador donde se podrá, después adaptar para las necesidades que tengan los alumnos, dado que el sistema es de código abierto es viable para poder examinarse y usar la tesis como guía para el entendimiento de cómo funciona y así los alumnos puedan adaptarlo a sus necesidades y hacer observaciones de comportamiento, y realizar adaptaciones, y de hecho el software permite usar video en vivo dado que convertirá el video en imágenes y así creará los datasets.

De esta manera el tecnológico tendrá herramientas que servirán a futuras generaciones para aprendizajes e implementación en nuevos proyectos.

Validación de Requisitos: Finalmente se validó la información y se hicieron las preguntas necesarias para determinar si había comprendido bien en qué consistía el proyecto, así como las posibles adaptaciones que ser verían necesarias. La información se validó con el director de tesis (Dr. Hernán de la Garza Gutiérrez).

4.3 Requisitos funcionales y no funcionales.

Funcionales

- Procesamiento de video, dado que será utilizado para creación de datasets.
- Uso de redes neuronales de convolución para procesamiento de video.
- Video en tiempo real.
- Dataset de imágenes para entrenamiento de las redes neuronales.
- Control del teclado para ejecutar comandos de movimiento.

No Funcionales

- El lenguaje de programación que se utilice, dado que existen varios lenguajes donde se puede implementar lo anterior.
- El tipo de simulador, puede ser un juego.

4.4 Prototipo

En la siguiente imagen se muestran los pasos necesarios que se necesitan para realizar el proceso que permitirá probar la red neuronal de convolución

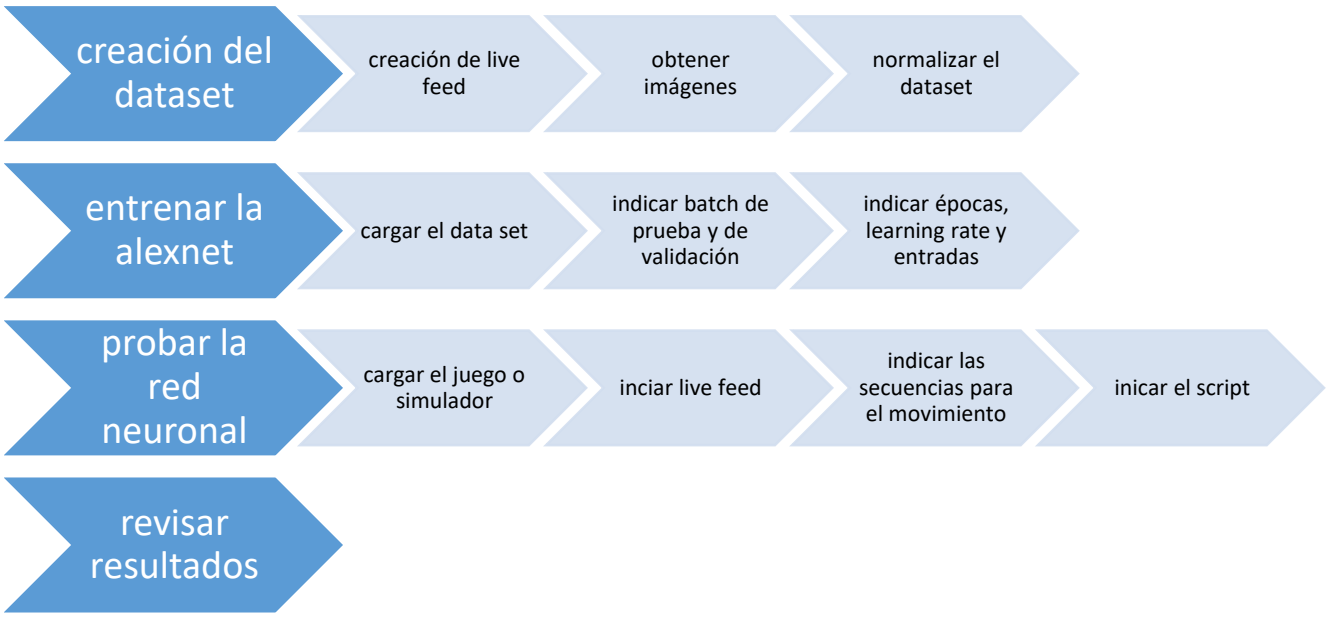

Ilustración 11: Proceso del simulador autónomo.

4.5 Problemas y soluciones al hacer el análisis

Al hacer el análisis se tuvo que reflexionar mucho, ya que no se había pensado a profundidad el problema. Por lo cual se consideró todos los temas de las unidades vistas en clase y la información de la clase de seminario.

Uno de los problemas más importantes es que no se tenía un prototipo para la creación de un software de un dron autónomo. Por lo cual se tuvo que pensar en uno y crear su diseño.

4.6 Diseño

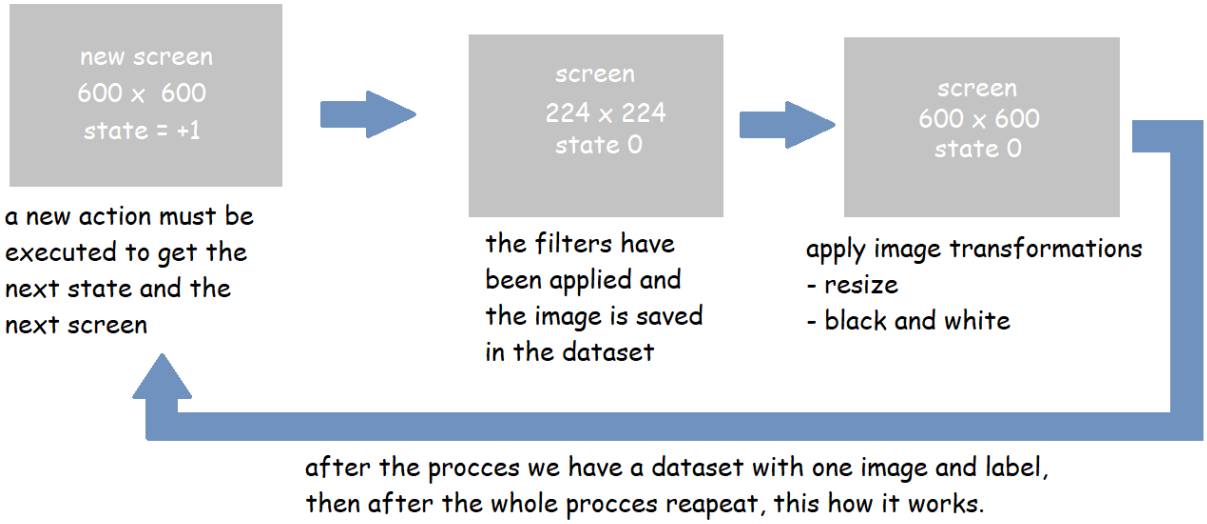

Ilustración 12: Diagrama de funcionamiento del entrenamiento.

5. DESARROLLO DEL SOFTWARE

El desarrollo en general del proyecto fue realizado en una plataforma de videojuegos llamada Unity, esto permitió crear un escenario personalizado que facilita diseñar pruebas para probar la eficacia del esquema de la Red Neuronal de Convolución tipo AlexNet. Se usaron distintas librerías para la captura de imágenes en tiempo real de la pantalla de la computadora a partir del video generado por la cámara del vehículo, las imágenes pasaron por un proceso de pre procesamiento que fue válido como entrada de la AlexNet y así mismo, ésta nos pudo regresar una salida. Esta salida fue la predicción del tipo de dirección que debería ejecutar el dron o vehículo virtual. Una buena parte del reto del proyecto fue la interconexión entre las diferentes librerías usadas para así lograr el recorrido del circuito sin errores.

5.1 Prototipo 1

CC3D

Se realizó la compra de un dron, y se procedió a armarlo, se tuvieron dificultades por falta de conocimiento técnico, y de material necesario para la construcción. Durante las primeras fases del proyecto el dron no respondía bien, y aun no se lograba establecer la conexión a través de Python para tomar control de las acciones. El primer autopiloto, no trabajó porque no había forma de dar instrucciones de forma remota.

Ilustración 13: Autopiloto CC3D

Se hicieron distintas pruebas, el piloto automático no respondía, y la configuración fue muy difícil, dado que muchas páginas de consulta o estaban caídas o requerían conocimientos previos en este tipo de dispositivos. Al final no se pudo obtener el control adecuado sobre el dron.

5.2 Pixhawk 4.3.1

Había varios ejemplos de drones inteligentes usando el pixhawk para hacer pruebas de vuelo autónomas, y habían sido exitosas, se cargaba Python a una raspberry pi y ésta se conectaba al pixhawk.

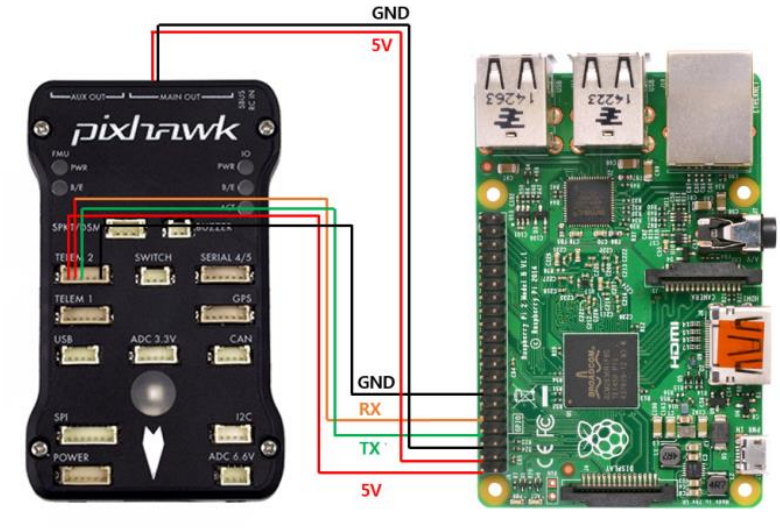

Ilustración 14: Diagrama de conexión de pixhawk y raspberry PI

Lamentablemente eso tampoco funcionó, se hicieron varios intentos de configuración del pixhawk y se acudió con unos técnicos dedicados a la fabricación de drones, los cuales ayudaron para hacer las configuraciones necesarias. Aun con ese apoyo, surgieron varios problemas de control y de estabilidad, parece que el pixhawk no respondía de manera adecuada. Los técnicos comentaron que ellos tuvieron muchos problemas de configuración con el piloto automático, y aún después de lograr hacer las configuraciones, el dron tenía comportamientos erráticos, donde los técnicos explicaron que requería mucha práctica para poder controlar el dron de manera adecuada. Lo anterior se constituyó como otro intento fallido, dado que si el dron tenía esos comportamientos no sería útil para la experimentación. Y aún

Ilustración 15: Primer prototipo del dron.

quedaban los problemas que se generarían para adaptar la cámara que se tenía considerada de usar.

5.3 Tello

El uso de Tello fue bastante cercano a lo que se buscaba, usando la librería Tello-Python, se pudieron realizar pruebas. El dron respondió muy bien a las pruebas que se hicieron con él, excelente manejo a través de Python conectado por el puerto de wifi, el dron tenía varias acciones que se le podía programar en secuencia. Se realizaron varias pruebas de manejo, incluso se accedió a su cámara.

Ilustración 16: Drone Tello.

Se hizo un script que permitía reconocer caras, usando la librería face_recognition se obtenían resultados similares a la ilustracion de abajo, donde se marcaba el rostro y el dron tomaba selfies de la persona, el dron era capaz de seguir el rostro del usuario y tomar selfies de manera constante.

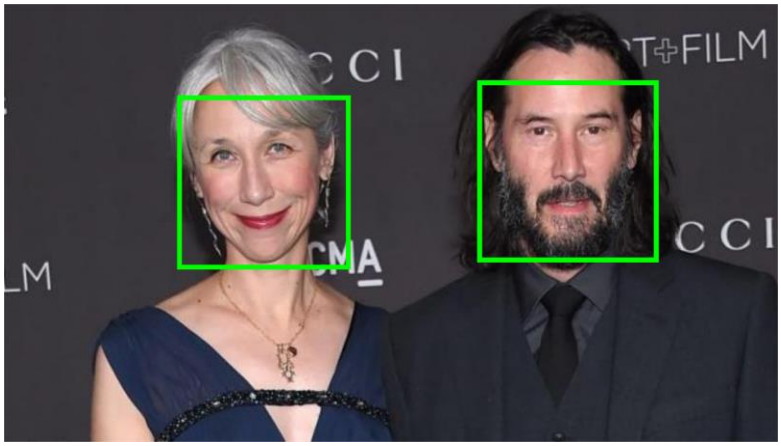

Ilustración 17: Reconocimiento de caras en una Imagen.

Las siguientes pruebas fueron acceder al vídeo en vivo que ofrecía y hacer pruebas sobre las imágenes, donde podíamos dar el *feed* de video directamente a una red Alexnet y probar resultados.

5.4 Conexión Python y unity a través de wifi

Se hizo un simulador básico para probar cómo hacer el control, pero dado que para lograr hacer el machine learning o reinforcement learning, se requería de hacer uso de sensores o de alguna case de feeedback para poder aprender, el simulador lamentablemente no tenía forma de hacerlo. Con **Unity3D-Python-Communication,** esta librería otorgó la posibilidad

de conectar Python para poder hacer el aprendizaje, pero durante momentos la comunicación falló, por lo cual el simulador y el sistema de aprendizaje, no funcionaron adecuadamente.

5.5 Detección de video en tiempo real y procesamiento de video

En esta parte fue donde se tomó la decisión de hacer el dataset de manera manual, esto significa que un usuario sería capaz de manejar el Drone en el ambiente simulado y ese video que se está usando en Segundo plano pasaría a formar parte de las imágenes del dataset. Una vez obtenido el dataset se aplican varios filtros de imagen para poder ser usados en el entrenamiento de la red neuronal, que posteriormente procesaría las imágenes y lograría realizar predicciones de una prueba real, que serían útiles cuando se corra el simulador de nuevo y las predicciones se usen para efectuar movimientos de manera autónoma.

5.5.1 Manejo de pulsaciones de teclado

Creo importante profundizar en esto, dado que principalmente la creación del simulador fue un paso esencial, lo cual llevó a la pregunta de cómo crear un dataset para poder entrenar la red neuronal. Entonces para eso se obtienen las pulsaciones del teclado y se pueden crear etiquetas. La forma de hacer eso fue con una librería llamada “win32api”. Primero se crea un arreglo de las posibles teclas que se pueden usar, usando el alfabeto en inglés y números.

```
# getkeys.py
# Citation: Box Of Hats (https://github.com/Box-Of-Hats )

import win32api as wapi
import time

keyList = ["\b"]
for char in "ABCDEFGHIJKLMNOPQRSTUVWXYZ 123456789,.'APS$/\\":
    keyList.append(char)
```

Ilustración 18: Código usado para pulsación de teclas.

Aquí ya tenemos el arreglo de caracteres donde se va a comparar si la tecla presionada fue alguna de ellas. Para realizar la comparación de cuál de las teclas fue tecleada, se creará un método que usaremos en el *script* principal, el cual permitirá buscar en nuestra lista y que nos regresará un arreglo o listas de teclas. Esto puede ser por las combinaciones de teclas que pueden ser usadas. Lo que hace es que recorre la lista que creamos anteriormente y usa el método “GetAsyncKeyState” para comparar si hay un match con el elemento actual en el ciclo si hay un match lo agrega al arreglo “keys”, si no hay match, no realiza ninguna acción.

5.5.2 Grabsecreen

Esta es una librería perteneciente a otro programador, obtenida a través de la página: https://github.com/Sentdex/pygta5/blob/master/grabscreen.py

La librería contiene una función que usa OpenCV para poder tomar captura del monitor, tiene la opción de asignar una región. La región que se asigne será la que contenga la imagen del simulador, lo demás tendrá color negro o blanco, así que solo será visible la región seleccionada. Si una región no es asignada tomará la pantalla completa. La región fue puesta en la librería para reducir la cantidad de ruido en las imágenes y ayudar al usuario que la use a conseguir mejores imágenes para sus proyectos.

5.5.3 Creación de etiquetas

La creación de etiquetas se realiza usando la librería que mencioné antes, donde cada pulsación del teclado es registrada. Cada pulsación es asociada a un *frame*. Se utiliza el método que nos regresa las teclas usadas en ese *frame*, y se mandan a una función que hace la clasificación de acuerdo a las teclas, claro que si una de las teclas que se recibe en la función de creación de etiquetas no existe, se toma la tecla definida de *default*. Esto quiere decir que si se presionó la tecla Q se creará la etiqueta [1,0,0] y si se tomó la Tecla E se creará la etiqueta [0,0,1], de esta manera cualquier tecla diferente será el default considerado com o [0,1,0].

```python
def key_check():
    keys = []
    for key in keyList:
        if wapi.GetAsyncKeyState(ord(key)):
            keys.append(key)
    return keys
```

Ilustración 20: Comparación de la pulsación de una tecla con las teclas de acción

```python
def keys_to_output(keys):
    #[Q,W,E]
    #[]
    output = [0,0,0]

    if 'Q' in keys:
        output[0] = 1
    elif 'E' in keys:
        output[2] = 1
    else:
        output[1] = 1

    return output
```

Ilustración 19: Código de la creación de etiquetas.

Todas estas etiquetas son usadas en conjunto con el *frame*, y se guardan en un archivo '*.npy'* que se usa como dataset, el cual puede ser abierto y reusado varias veces.

5.5.4 OpenCV para captura de video y pre procesamiento

La imagen que el agente ve dentro del circuito se tiene en el ambiente de Unity, que es donde se está corriendo el programa, es el video a partir de la cámara del agente dentro de Unity. Debido a que hay que entrenar la CNN, con el script desarrollado en OpenCV, se realiza la captura de la pantalla cada cierto tiempo, para así extraer y luego procesar *frames* del video que serán la entrada a la Alexnet.

Se habilitaron dos tipos de recorridos, modo usuario y modo automático. El recorrido en modo usuario es cuando el usuario va indicando por medio del teclado qué acción realizar cada instante, hasta efectuar el recorrido completo del circuito y en modo automático, que es cuando el agente virtual es quien va decidiendo qué hacer según la imagen del circuito que tenga enfrente. El primero es necesario para construir el dataset y entrenar la CNN y el segundo para ver si el entrenamiento de la red fue adecuado para poder efectuar un recorrido completo.

Usando el esquema de usuario, se puede ir haciendo capturas de pantallas (posición del agente) y asociarlas con la acción que el usuario realizó. Al guardar este par de datos en cada acción tomada, permite crear el dataset de entrenamiento de la CNN.

```
while True:
    screen =  grab_screen(region=(0,40,800,640))
    cv2.imshow('window2',cv2.cvtColor(screen, cv2.COLOR_BGR2RGB))
    screen = cv2.cvtColor(screen, cv2.COLOR_BGR2GRAY)
    screen = cv2.resize(screen,(80,60))
    keys = key_check()
    output = keys_to_output(keys)
    training_data.append([screen,output])
```

Ilustración 21: Código utilizado para pre procesamiento de la imagen.

En el código se ven las configuraciones realizadas para la captura del video usando la función grabscreen () que nos regresa imágenes de la pantalla en tiempo real cada ciertos milisegundos. Las líneas siguientes ajustan el tamaño de la imagen a 224 x 224 pixeles que requiere la Alexnet como entrada. Además, se detecta y guarda la tecla pulsada que equivale a la acción tomada. De esta manera se tiene la imagen y su acción. Este par de datos nos permite el aprendizaje supervisado de la CNN.

5.5.5 Creación de archivo del dataset

El archivo se puede crear usando la librería "numpy", la cual creará un archivo al terminar el proceso usando la letra "p", la letra es opcional. Una vez termine el proceso usará el comando *np.save ("nombre del archivo", "arreglo")* con esto se logra crear nuestro archivo con el nombre que elegimos, y así tendremos un arreglo de imágenes etiquetadas para su uso posterior.

También se puede cargar el archivo creado antes, lo cual se hace al inicio del script para añadir nueva información, de esta manera se puede aumentar la información en el dataset. En caso de que el archivo no exista se crea un arreglo vacío que será usado en su lugar. Esto se puede observar en la imagen de abajo.

```
file_name = 'training_data.npy'

if os.path.isfile(file_name):
    print('File exists, loading previous data!')
    training_data = list(np.load(file_name))
else:
    print('File does not exist, starting fresh!')
    training_data = []
```

Ilustración 22: Código de creación de archivos en python.

5.5.6 Balanceo de la información

Para empezar el balanceo de la información lo cual permitirá que funcione mejor el entrenamiento, se debe cargar el archivo que se creó anteriormente, esto permite que la información sea más uniforme. Si usamos una secuencia de imágenes del mismo tipo podría

```
train_data = np.load('training_data.npy')
print(len(train_data))
df = pd.DataFrame(train_data)
```

Ilustración 23: Código de carga de dataset de entrenamiento.

afectar la eficiencia del entrenamiento. Se usa la librería de *pandas* que sirve para trabajar este tipo de datos que están en forma de arreglos. Los *data frame* son capaces de crear el equivalente a tablas ya que divide la información en columnas y renglones.

Una vez realizado lo anterior, el script se encargará de acomodar la información, la cual será organizada por el ciclo que recorrerá cada renglón buscando la imagen y su etiqueta. Una vez

```
lefts = []
rights = []
forwards = []

shuffle(train_data)

for data in train_data:
    img = data[0]
    choice = data[1]

    if choice == [1,0,0]:
        lefts.append([img, choice])
    elif choice == [0,1,0]:
        forwards.append([img, choice])
    elif choice == [0,0,1]:
        rights.append([img, choice])
    else:
        print('no matches!!!!')
```

Ilustración 24: Clasificación de las acciones.

hecho eso se irán asignando cada pareja de datos en sus respectivos arreglos, esto quiere decir que tendremos un arreglo con todas las parejas de un tipo, un arreglo con todos los datos de esa acción, al ser tres acciones, por lo tanto 3 arreglos. Las imágenes que estén sin su etiqueta no serán asignadas. Antes de la asignación se mezcla la información, esto principalmente por lo que se comentó, de evitar al máximo posible los casos datos iguales seguidos, porque si durante el ciclo de acciones son iguales a la hora de guardarse seguirán manteniendo esa continuidad.

Una vez realizado lo anterior, se prosigue a creación de las listas, no se espera poner toda la información dado que sería un entrenamiento irregular, lo mejor es buscar una equivalencia entre los datos, por lo cual se hace lo siguiente: la acción *forward* es una que genera enorme

```
forwards = forwards[:len(lefts)][:len(rights)]
lefts = lefts[:len(forwards)]
rights = rights[:len(forwards)]

shuffle(forwards)

final_data = forwards[:2000] + lefts + rights

shuffle(final_data)
```

Ilustración 25: Proceso de mezclado del dataset.

cantidad de datos de trabajo ya que es la que se repite muchas veces, por lo cual se redimensiona a un tamaño similar al de las otras dos acciones, es decir, acorde a la vuelta izquierda y derecha. De esta manera si la cantidad de datos de la acción forward es 3000 podemos bajarlo a 1000, esto ayuda a que haya un cierto equilibrio en el dataset, así si las otras acciones también se limitan al tamaño de *forward*, esto nos ayudará a que la red no quede sobrentrenada en una acción en específico.

Al final aún se busca limitar *forward* de nuevo, esto puede variar acorde a los análisis que se observen en los resultados de las predicciones. Pero eso es a criterio del investigador que haga uso del recurso. Por último, se guarda el archivo nuevo, que es la información que se usará para alimentar a la red neuronal y hacer la evaluación de sus predicciones.

```
np.save('training_data_v2.npy', final_data)
```

Ilustración 26: guardar archivo de dataset

5.5.7 Alexnet

Aquí se pueden apreciar las diferentes capas que componen a la red neuronal AlexNet, este pedazo de código es básicamente la declaración de la red neuronal, donde se usara librería Tflearn como apoyo para poder hacer los diferentes procesos que son requerido por la red neuronal.

```
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
from tflearn.layers.normalization import local_response_normalization

def alexnet(width, height, lr):
    network = input_data(shape=[None, width, height, 3], name='input')
    network = conv_2d(network, 96, 11, strides=4, activation='relu')
    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)
    network = conv_2d(network, 256, 5, activation='relu')
    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)
    network = conv_2d(network, 384, 3, activation='relu')
    network = conv_2d(network, 384, 3, activation='relu')
    network = conv_2d(network, 256, 3, activation='relu')
    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)
    network = fully_connected(network, 4096, activation='tanh')
    network = dropout(network, 0.5)
    network = fully_connected(network, 4096, activation='tanh')
    network = dropout(network, 0.5)
    network = fully_connected(network, 3, activation='softmax')
    network = regression(network, optimizer='momentum',
                         loss='categorical_crossentropy',
                         learning_rate=lr, name='targets')

    model = tflearn.DNN(network, checkpoint_path='model_alexnet',
                        max_checkpoints=1, tensorboard_verbose=2, tensorboard_dir='log')

    return model
```

Ilustración 27: Alexnet usada en el código.

5.5.8 Entrenando la red

Aquí se inicia con la exportación de red neuronal y Numpy que se usaran para este script, declaramos el tamaño de la imagen con WIDTH y HEIGHT. De ahí se aplica Learning Rate como LR, las épocas que se usaran y el nombre del modelo; el cual será asignado como drone-1e-3-alexnetv2-80.

Posterior a cargar el dataset llamado training_data_v2.npy se crean 2 arreglos uno para que la red use para entrenamiento y otro para que evalué sus resultados, estos tienen que ser asignados a las variables y reacomodados como es requerido. Por lo cual se transforman sus

dimensiones en el caso de las imágenes y se usan las etiquetas para la Y que son los resultados.

Se introduce el arreglo X y los objetivos como el arreglo Y, el número de épocas los arreglos de validación o de test, y otros parámetros que requiere la función para ejecutarse. Se salva el modelo y el código que inicia con tensorboard es un código que se ejecuta en la terminal, para poder obtener las métricas en una página de navegador y hacer examinación del comportamiento que tuvo durante el aprendizaje.

```
import numpy as np
from alexnet import alexnet

WIDTH = 224
HEIGHT = 224
LR = 1e-3
EPOCHS = 80
MODEL_NAME = 'drone-{}-{}-{}-epochs.model'.format(LR, 'alexnetv2', EPOCHS)

model = alexnet(WIDTH,HEIGHT,LR)

train_data = np.load('training_data_v2.npy', allow_pickle=True)

train = train_data[:500]
test = train_data[500:]

X = np.array([i[0] for i in train]).reshape(-1, WIDTH, HEIGHT, 3)
Y = [i[1] for i in train]

test_x = np.array([i[0] for i in test]).reshape(-1, WIDTH, HEIGHT, 3)
test_y = [i[1] for i in test]


model.fit({'input': X}, {'targets': Y}, n_epoch=EPOCHS,
          validation_set=({'input': test_x}, {'targets': test_y}),
          snapshot_step=500, show_metric=True, run_id=MODEL_NAME)

model.save(MODEL_NAME)
```

Ilustración 28: Código que representa el proceso de entrenamiento.

```
tensorboard --logdir=foo:C:/Users/H/Desktop/ai-gaming/log
```

Ilustración 29: arrancar página de estadísticas de la AlexNet

5.5.9 Circuito usado para el entrenamiento

El circuito usado fue creado con Unity. Consiste de una serie de vueltas izquierdas y derechas, que llevan al punto de inicio, cerrando el circuito. Contiene iluminación en diferentes puntos del mismo, para que el agente cuente con suficiente claridad durante el recorrido, así como también se cambiaron las texturas

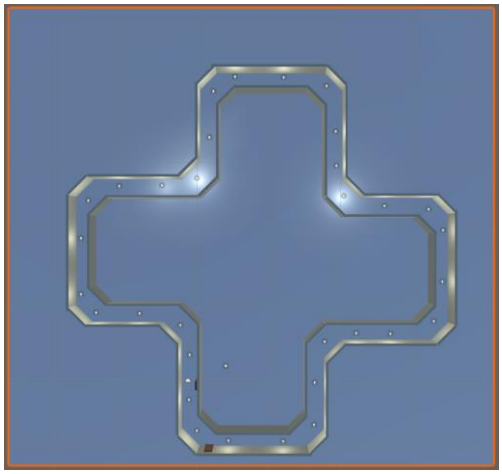

Ilustración 30: Circuito creado en Unity utilizado en las pruebas de la Alexnet.

5.5.10 Imágenes del circuito con texturas en paredes

Ilustración 32: Escenario 1.

Ilustración 31: Escenario 2.

Ilustración 34: Escenario 3.

Ilustración 33: Escenario 4.

5.5.11 Creación de la CNN Alexnet

Se utilizó la estructura de la red ya creada en la “ImageNet Classification with Deep Convolutional Neural Networks”, la cual contenía la estructura de la red neuronal convolucional, y se adecuó a las necesidades de la investigación.

Famosa por ganar la competencia 2012 ImageNet LSVRC-2012 por un amplio margen (15.3% vs 26.2% (segundo lugar) tasas de error). La red tenía una arquitectura muy similar a la de LeNet de Yann LeCun et al, pero era más profunda, con más filtros por capa y con capas convolucionales apiladas.

La Alexnet contiene 8 capas que tienen pesos de entrenamiento, las primeras 5 capas de éstas son de *convolución* y las otras 3 son las capas finales *fully connected*. La primera capa toma la imagen a color de 224 x 224 x 3 y le aplica *convolución* con 96 *kernels* (filtros) de tamaño 11 x 11 x 3 con un *stride* de 4 pixeles, enseguida se aplica *normalización* y *MaxPooling*. La segunda capa *convolucional* toma la salida de la capa anterior y le aplica 256 *kernels* de 5 x 5 x 48, también le sigue una *normalización* y *MaxPooling*. Las siguientes tres capas son de *convolución* y no se aplica *normalización* ni *pooling*. La capa tres aplica 384 *kernels* de 3 x 3 x 256. La capa cuatro tiene 384 *kernels* de 3 x 3 x 192 y la capa quinta con 256 *kernels* de tamaño 3 x 3 x 192. Esto hace que en ese momento se tengan 4096 pixeles totales. Por eso es que la primera capa de la zona de *fully connected* tiene 4096 neuronas de entrada. En la capa final del *fully connected* se deja una salida con el número de neuronas igual al número de acciones que se le habiliten al agente. En este caso son 3 acciones y por tanto 3 neuronas en la capa de salida. En esta capa se utiliza función Softmax para generar probabilidades para cada acción y se toma la mayor de ellas. Adicional a lo anterior, en las últimas capas se aplica Dropout con probabilidad de 0.5. Se utiliza la función de activación ReLU (rectificado lineal) en lugar de Tanh o Función Sigmoide (Logistic Regression) y para el ajuste de los pesos se utilizó Stochastic Gradient Descent (SGD) con momentum. Esta red tiene 62.3 millones de parámetros que entrenar.

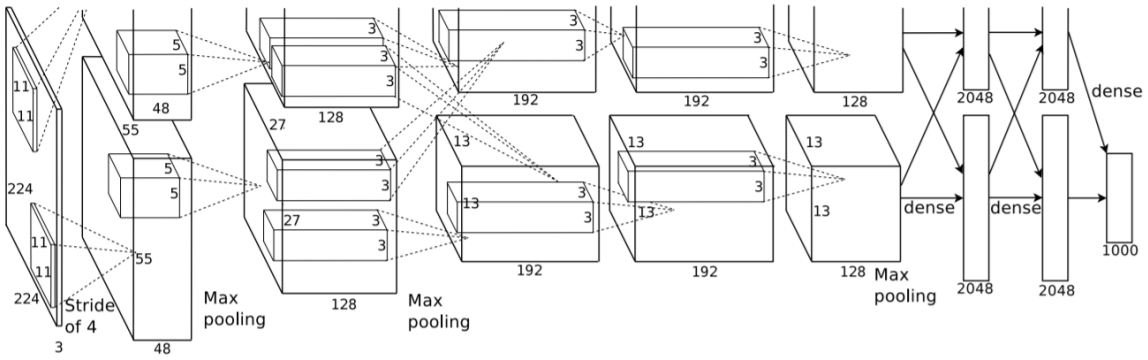

Ilustración 35: Modelo de la red neuronal convolucional Alexnet.

5.5.12 Entrenamiento de la Red Neuronal Alexnet

El entrenamiento del modelo requiere usar el dataset generado, es decir, el arreglo de imágenes capturadas con sus respectivas etiquetas (acciones). El conjunto de imágenes se separó en dos grupos. Uno fue utilizado para el entrenamiento de la red neuronal CNN, mientras que el segundo se usó para la validación y darse cuenta de la eficiencia de la red neuronal para determinar la acción correspondiente a la imagen. El entrenamiento se repite hasta que se logra que se decida la acción correcta de acuerdo a la imagen presentada.

Cuando se realiza el entrenamiento se empieza con la primera época. Cada época puede ser considerada como un ciclo con cierto número de pasos. Los datos de la tabla inferior muestran la pérdida o error de clasificación en correlación con el resultado real, la exactitud, también llamada acurracy, donde se muestra el porcentaje de éxito en la clasificación, el número de imágenes que se probaron, la pérdida total y el tiempo que tomó realizarlo.

El entrenamiento fue realizado usando 80 épocas, y un valor de learning rate de 0.001, usando 1000 muestras de entrenamiento y 237 para la validación del entrenamiento.

5.5.13 Creación del simulador

El simulador fue creado a través de Unity y cumple con la función de una ventana con dimensiones de “800 x 600”. Esta característica es usada para que la librería que captura el video de la pantalla de monitor pueda trabajar y que la librería "grabscreen ()", tome el video en tiempo real, mientras que "OpenCV" realiza el pre procesamiento de la imagen.

Como se expone en la siguiente imagen, el laberinto fue diseñado como un camino infinito, esto con la intención de facilitar la obtención de datos para el dataset:

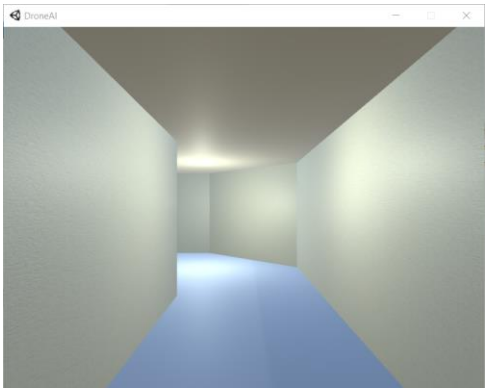

Ilustración 36: Imagen obtenida del simulador en donde se observa una curva a la izquierda.

La perspectiva de la imagen está diseñada para simular la trayectoria de un dron real, con lo que se facilita la transmisión del código y así obtener una mayor correlación.

5.5.14 Generación del dataset

La generación del dataset se realizó tomando capturas relacionadas al entorno virtual, es decir, capturas de pantalla de cada frame; esto se refiere a que durante el momento de estar moviendo el agente virtual se detectaban las pulsaciones de teclado. Así mismo, con cada pulsación de las teclas de movimiento del agente, se obtenía una imagen y una etiqueta, los cuales se guardaban dentro de una lista. Esto permitió obtener un dataset mucho más preciso y confiable.

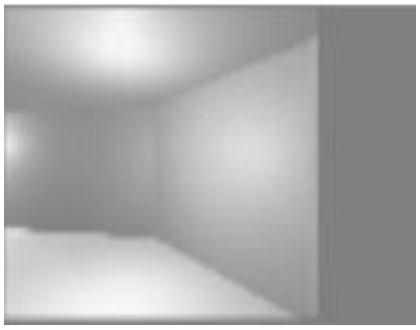

Ilustración 37: La imagen corresponde a una acción de ir en dirección hacia adelante, por eso la etiqueta de la imagen es [0, 1,0].

De la misma manera, se obtiene un conjunto de datos cuya nomenclatura se especifica a continuación:

Etiqueta	Movimiento
[1,0,0]	Izquierda
[0,1,0]	Derecho
[0,0,1]	Derecha

Tabla 1: Nomenclatura seguida para el tipo de etiqueta y movimiento.

Cuando se pulsa la tecla “Q” se guarda la imagen y el label como se muestra en la tabla, mientras que el usuario continúa desplazándose a través del laberinto infinito, para generar suficiente información que sirva para el

entrenamiento de nuestra AlexNet. Cabe destacar que las imágenes se trabajan con las dimensiones “224x224”, dimensiones empleadas en la red neuronal.

5.5.15 Balanceo del dataset

El problema más común de los dataset, al menos a la hora de la creación de uno de esta manera, es la información que viene de manera secuencial. Esto se refiere a que muchas imágenes del mismo tipo vienen una tras otra, lo cual ocasionaría que la red no se entrene de manera adecuada, por lo tanto, se hace un mezclado de la información para crear un dataset desordenado, por así decirlo balanceado, en el cual las imágenes queden fuera de secuencia, es decir mezcladas.

Para lograr el objetivo de balanceo se utiliza la función shuffle de la librería random, la cual hace un conteo de todas las clases y se da un reporte de cuantas clases de cada tipo hay, al finalizar el script. Este genera un archivo npy con el dataset balanceado, para ser usado para el entrenamiento de la red neuronal.

5.5.16 Experimentación y resultados

Los experimentos y resultados realizados se pueden observar en las imágenes de abajo, donde hubo varias versiones del simulador y una fue donde las curvas tenían ángulos de 90 grados y posteriormente en las nuevas versiones se cambiaron vueltas más abiertas, esto debido a que la ejecución se hace cada ciertos frames con la lectura de la imagen por lo cual el dron tuvo esas observaciones que se mencionan abajo.

5.5.16.1 Laberinto con vueltas de 90 grados

El primer laberinto que se utilizó contenía vueltas pronunciadas de 90 grados y se probaron algunas técnicas para tomar dichos caminos, esto de acuerdo a la acción programada. No obstante, se observó que siempre existía una colisión en las curvas, lo cual provocó la modificación de la probabilidad tomada de la CNN (que dependiendo del porcentaje realiza la acción), pero sin mucho aval en la situación.

Ante esto, se descartó que el dron, en condiciones idóneas, fuera capaz de dar una vuelta. Se requería un algoritmo de aprendizaje reforzado para ayudar a mejorar la vuelta que se obtiene.

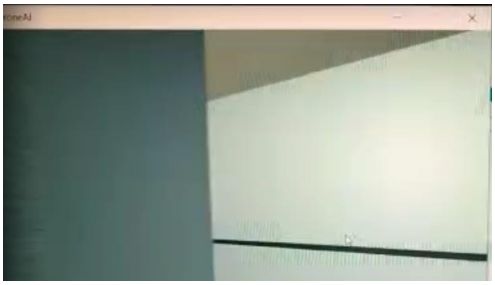

Ilustración 38: Imagen que muestra uno de los choques y por qué no fue útil el uso de vueltas de 90.

5.5.16.2 Laberinto de vuelta abierta

El segundo laberinto mostró mejores resultados para la toma de decisiones, ya que éste era capaz de pasarlo sin colisión alguna. Aunque se realizaron varias pruebas en su funcionamiento, al igual que el anterior, demostró la utilidad de una CNN y el cómo ésta podía ser entrenada bajo simulaciones con especificaciones determinadas.

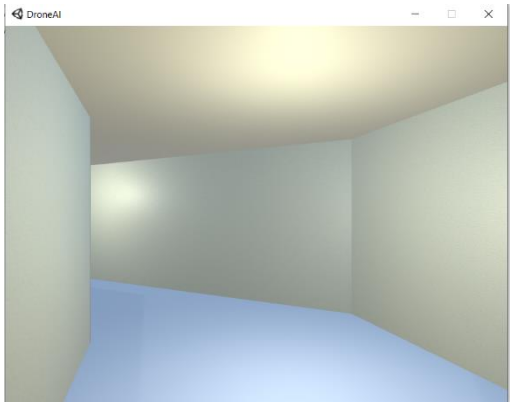

Ilustración 39: La imagen corresponde a la simulación final del laberinto.

6. DESARROLLO DEL SOFTWARE

6.1 Creación del dataset

Para la utilización del software es necesario, primero crear el dataset. Para esto tenemos la opción más importante y que considero puede ser muy útil para muchos proyectos de otras índoles en el tecnológico, que es tener un video en tiempo real de la pantalla del monitor. Una de las principales indicaciones es que el simulador debe tener las dimensiones de *800 px x 600 px,* y debe estar ubicada en la posición superior izquierda como se muestra en la figura. De esta manera el video será transferido para ser procesado, también tiene la configuración para poder establecer límites de lo que se busca visualizar, ya sea que se limite el área con un polígono irregular de 4 vértices o uno regular, pero usualmente son 4 vértices que dan como parámetros.

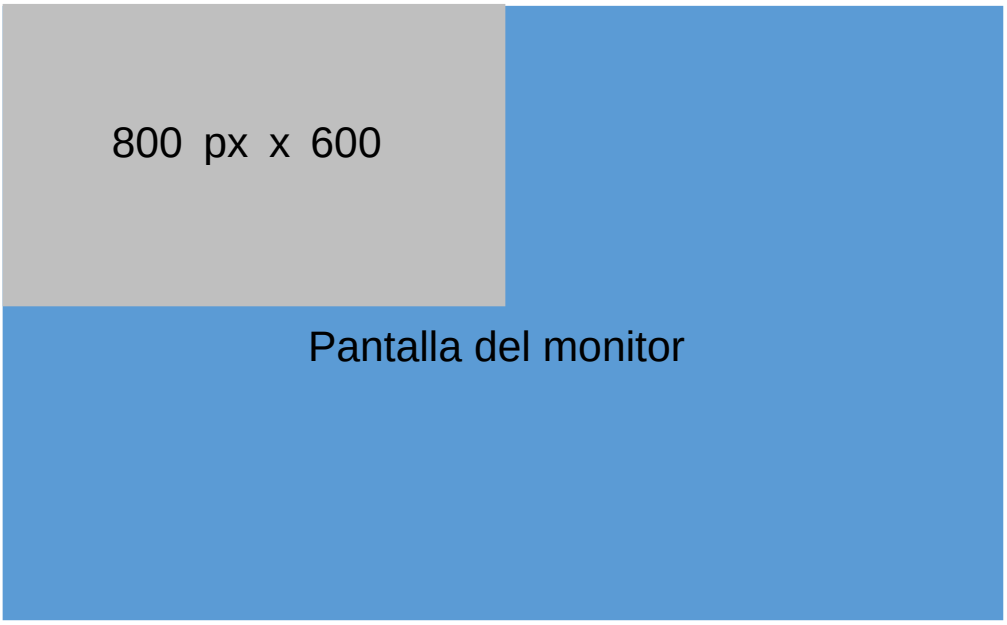

Ilustración 40: Pantalla con simulador abierto.

Una vez iniciada la pantalla del simulador que se usara, o de un live de una cámara, se podrán ejecutar los comandos. Debemos recordar que para poder crear un dataset en este simulador requiere que se conduzca el dron virtual a través del escenario lo cual provocara que se guarden las imagen con la combinación de teclas como etiqueta de la imagen, esto quiere decir que si el dron virtual va hacia enfrente se guarda una imagen de 224 x 224 con una etiquete [0,1,0], para poder iniciar el programa se podrá hacer en “símbolo de sistema” o usando plataformas de Python como anaconda, para eso primero iniciamos el entorno en este caso py35 contiene todas las librerías necesarias.

```
D:\drone-simulator-master>conda activate py35

(py35) D:\drone-simulator-master>
```

Ilustración 41: activación del environment del proyecto.

En esta caso usamos el comando Python que permite ejecutar los archivos .py, y a continuación se iniciara un conteo de 5 para abajo y empezara a lanzar la información sobre la latencia de la captura de video de la pantalla, con esto una vez empecemos a controlar el dron virtual el script de Python detectara las combinaciones de teclas que hayan sido previamente configuradas y cuando sean activadas esas pulsaciones se guardara la imagen y su etiqueta, de esta manera, una vez se acabe de hacer el proceso se puede teclear la tecla P y se guardara el dataset q se creó, dando también información de la cantidad de imágenes recolectadas de cada configuración, o mejor dicho acción registrada.

```
(py35) D:\drone-simulator-master>python training.py
```

Ilustración 42: Comando de inicio del proyecto.

Es importante conocer las configuraciones posibles, y como ejecutarlas. Primero contamos con las acciones disponibles, en nuestro caso 3 acciones, [izquierda, derecho, derecha] y se pueden observar en la imagen como [Q, W, E].

```
if 'Q' in keys:
    output[0] = 1
elif 'E' in keys:
    output[2] = 1
else:
    output[1] = 1
```

Ilustración 43: Código de las acciones.

A continuación, podemos cambiar el nombre del archivo del dataset, tomar en cuenta que será importante preservar se nombre más adelante.

```
file_name = 'new_color_training_data.npy'
```

Ilustración 44: Nombre de archivo.

En este código se pueden modificar la región como anteriormente mencione, y se puede armar un polígono regular o irregular dependiendo de las indicaciones que se pongan ahí. A si mismo

se puede trabajar que la imagen que se guarde sea a color o blanco y negro, y por último, pero no menos importante las dimensiones de la imagen.

```
screen =  grab_screen(region=(0,40,800,640))
cv2.imshow('window2',cv2.cvtColor(screen, cv2.COLOR_BGR2RGB))
#·screen·=·cv2.cvtColor(screen,·cv2.COLOR_BGR2GRAY)
screen = cv2.resize(screen,(224,224))
```

Ilustración 45: Código de configuraciones.

6.2 Balanceo del dataset

El balanceo del dataset es importante, porque de esta manera el dataset quedara revuelto, para no tener múltiples imágenes seguidas de un mismo tipo de acción, así que el balanceo va revolverlas y reducir el dataset, se nos mandara una información de las acciones registradas después del balanceo y se creara un nuevo dataset, como se presencia en la imagen es necesario poner el nombre que se usó en el paso anterior.

```
train_data = np.load('new_color_training_data.npy', allow_pickle=True)
print(len(train_data))
df = pd.DataFrame(train_data)
print(df.head())
print(Counter(df[1].apply(str)))
```

Ilustración 46: Código de carga de archivo.

Aquí se puede poner el nuevo nombre, que por razones simples se le agrego v2 al final, para indicar que el dataset balanceado. Una vez terminado el balanceo se pude usar para el siguiente proceso.

```
np.save('new_color_training_data_v2.npy', final_data)
```

Ilustración 47: Código de guardado de nuevo archivo.

6.3 Entrenamiento del modelo

Para el entrenamiento debemos poner las dimensiones que se usaran como parámetro en la Alexnet, es importante que el dataset contenga las imágenes en esas mismas dimensiones. Después tenemos “LR” learning rate que puede ser ajustado para hacer pruebas con diferentes para las experimentaciones, y por ultimo tenemos las épocas, aquí también el número de épocas dependerá de las pruebas del investigador a cargo. Usando estos datos se guardará el modelo y ese modelo entrenado se podrá usar en la prueba real, donde se le dará un video y el dron virtual será controlado por la máquina.

```
WIDTH = 224
HEIGHT = 224
LR = 1e-3
EPOCHS = 80
MODEL_NAME = 'drone-{}-{}-{}-epochs.model'.format(LR, 'alexnetv2', EPOCHS)
```

Ilustración 48: Configuración de datos de la Alexnet.

Aquí se cargará el dataset balanceado, y tenemos el “training” que podemos asignar parte de nuestro dataset para entrenamiento y la otra parte para pruebas, en este caso los primeros 1000 datos son usados para entrenar y de 1000 en adelante para probar el desempeño de la Alexnet.

```
train_data = np.load('new_color_training_data_v2.npy', allow_pickle=True)

train = train_data[:1000]
test = train_data[1000:]
```

Ilustración 49: Cargar archivo.

Aquí es importante ver que estos son los modelos creados anteriormente, y así es como se irán guardando los modelos. Después se usará el modelo para cargarlo y probar el sistema autónomo.

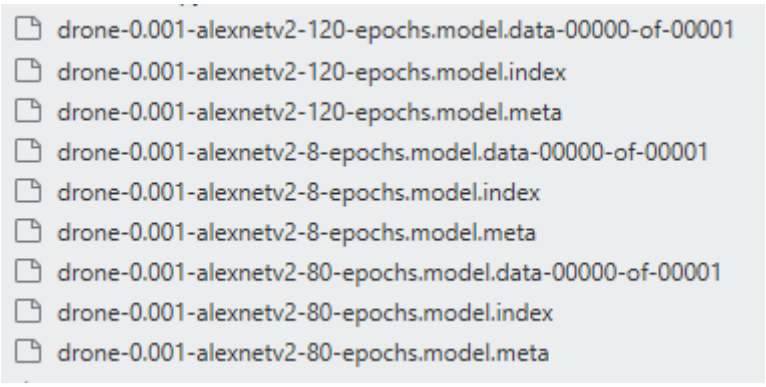

Ilustración 50: Modelos.

6.4 Probar la red neuronal con el simulador

Para probar el modelo es necesario especificar la información tal cual se usó a la hora de entrenarlo, esto permitirá cargar el modelo entrenado en los pasos anteriores. Una vez realizado esto hay que tener abierto el simulador como se explicó anteriormente.

```
WIDTH = 224
HEIGHT = 224
LR = 1e-3
EPOCHS = 120
MODEL_NAME = 'drone-{}-{}-{}-epochs.model'.format(LR, 'alexnetv2', EPOCHS)
```

Ilustración 51: Configuración de la Alexnet para su uso.

A continuación, se puede observar la autonomía, tenemos una predicción que nos arroja el método *predict*, se le da la imagen con la configuración adecuada, esto nos arrojará el movimiento que la red neuronal considera es la acción adecuada. Para eso tenemos algo así [0.2334, 0.83435, 0.3323] uno de los movimientos en este caso sería ir recto. Around () lo que hace es que redondea el valor a 1 o 0, para poder recibir el movimiento como se observa en la parte de abajo, se ejecuta el movimiento se espera un segundo y efectúa el resto del código.

```
prediction = model.predict([screen.reshape(WIDTH,HEIGHT,3)])[0]
moves = list(np.around(prediction))
print(moves, prediction)

if moves == [1,0,0]:
    time.sleep(1)
    left()
elif moves == [0,1,0]:
    time.sleep(1)
    straight()
elif moves == [0,0,1]:
    time.sleep(1)
    right()
```

Ilustración 52: Manejo de las predicciones.

Se obtienen distintos métodos, esos métodos se observan en la imagen, se realiza la conjunción de teclas de tal manera que se puede representar mejor la acción en el simulador, estas acciones fueron evaluadas en varias ocasiones para adaptarse de mejor manera a los movimientos del dron virtual y conseguir mejor resultados. Y lograr que los movimientos fueran más flexibles, y llevara a cabo de mejor manera las acciones que le mandaba la predicción de la red neuronal.

```
def straight():
    PressKey(W)
    ReleaseKey(Q)
    ReleaseKey(E)
```

Ilustración 55: Acción 1.

```
def left():
    PressKey(Q)
    PressKey(W)
    ReleaseKey(E)
```

Ilustración 54: Acción 2.

```
def right():
    PressKey(E)
    PressKey(W)
    ReleaseKey(Q)
```

Ilustración 53: Acción 3.

7. RESULTADOS Y CONCLUSIONES

Se realizaron varios experimentos usando un laberinto sin texturas, con texturas, la cantidad de muestras de entrenamiento y el número de épocas para determinar el mejor entrenamiento de la CNN.

Los resultados que se muestran a continuación corresponden al uso de un dataset con 1000 imágenes de entrenamiento y 236 de validación o prueba y a un laberinto sin texturas en las paredes.

La ilustración 56 muestra la evolución de la exactitud de predicción durante el entrenamiento. Puede verse como alcanza valores del orden de 0.97 al final del entrenamiento con 80 épocas.

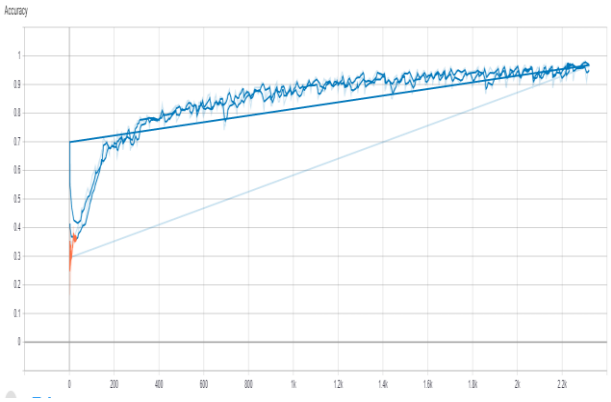

Ilustración 56: Exactitud de predicción durante el entrenamiento.

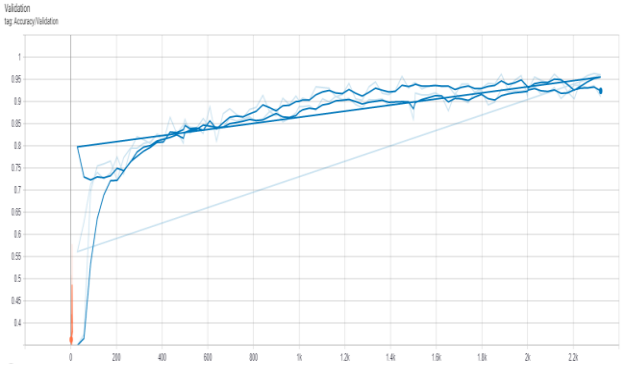

Ilustración 57: Se muestran los resultados de la validación.

En la ilustración 57 se muestran los resultados de la validación con el conjunto de datos para dicho fin. Es decir que son los datos diferentes a los usados en el entrenamiento. El alcance en la validación llega al 90 % y en esta corrida con 80 épocas nos permite ver que se requiere mejorar dicho entrenamiento.

En la ilustración 58 se grafican los resultados generados por la función de pérdida, la cual es la que permite hacer el ajuste en los valores de los pesos de los filtros de la CNN para ir mejorando en su desempeño. La función utilizada por la CNN es la de Cross Entropy. Esta función de pérdida no se requiere que llegue a valores de cero, sino que cuando se estabiliza muestra que la red ya quedó entrenada. En el gráfico se aprecia que lleva tendencia descendente y que podría esperarse que tenga valor menor si se incrementen el número de épocas de entrenamiento.

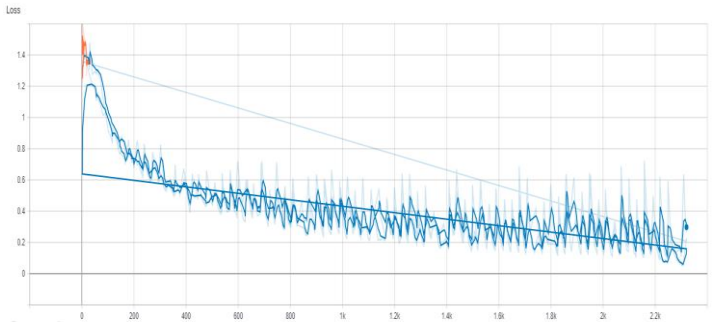

Ilustración 58: Resultados de la función Loss o de pérdida.

En el segundo experimento se utilizó un circuito con 4 texturas diferentes en los segmentos del circuito y eso conlleva a utilizar un dataset propio de este sistema. Se mantiene tanto la cantidad de datos de entrenamiento y de validación, como el número de épocas de entrenamiento.

En la segunda simulación puede verse ilustración 59, que se alcanza un valor un poco más alto en exactitud (aprox 0.98). Igual se aprecia una ligera mejora en la validación, ilustración 60. Los resultados de *Loss* **function** durante la corrida de entrenamiento (Ilustración 61) se ve semejante al experimento anterior.

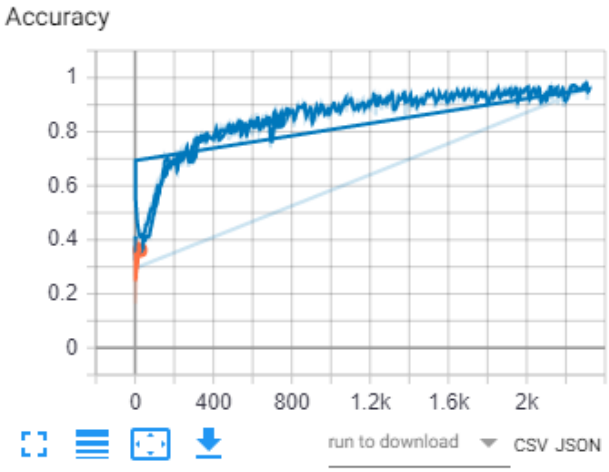

Ilustración 59: Exactitud de predicción durante el entrenamiento.

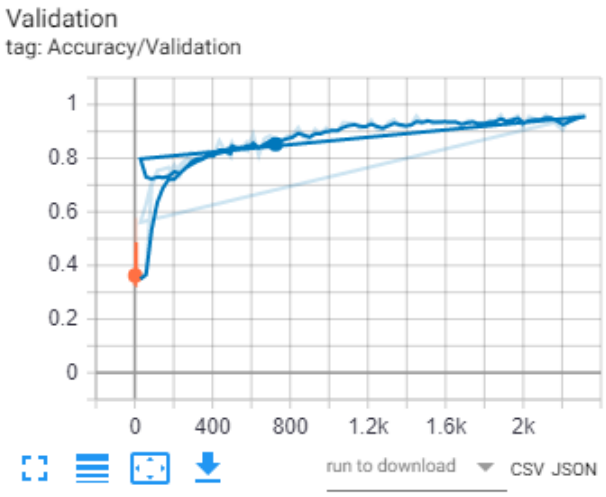

Ilustración 60: Se muestran los resultados de la validación experimento.

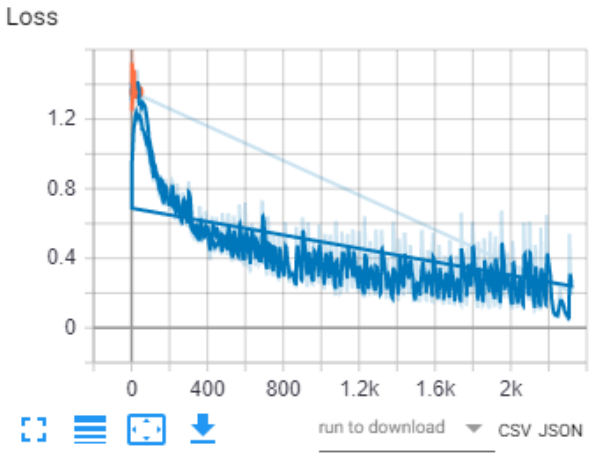

Ilustración 61: Resultados de la función Loss o de pérdida segunda simulación.

Por lo cual se puede apreciar una mejora a la red neuronal cuando se hicieron las segundas pruebas, lo cual puede adjudicar que con más muestras y más variaciones tuve mejor desempeño al final, dado que se usaron imágenes de blanco y negro e imágenes a color por lo que en cierta manera aunque sea ligero, se obtuvo mejores resultados usando imágenes a color.

8. REFERENCIAS BIBLIOGRÁFICAS

1. Greengard, Samuel (2007). Gaming Machine Learning: Game simulations are driving improvements in machine learning for autonomous vehicles and others devices. *Communications of the ACM vol. 60*, 14 – 16, ISSN # 0001-0782
2. H. Alvarez, L. Paz, J. Sturn, and D. Cremers. (2016). Collision avoidance for quadrotors with a monocular camera. *In international Symposium on Experimental Robotics (ISER)*, 195-209, ISSN # 1610-7438
3. A. Loquercio, A. I. Maqueda, C.R.D.Blanco, and D. Scaramuzza. (2018). Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*. 1-1, DOI # 10.1109/LRA.2018.2795643
4. Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.
5. Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, Davide Scaramuzza. (2018) Deep Dron Racing: Learning Agile Flight in Dynamic Environments
6. Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, Stan Birchfield (2017) Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness.
7. Rivero, E. (2015, February 27). DQN: La inteligencia artificial de Google que domina los videojuegos. Unocero. https://www.unocero.com/ciencia/dqn-la-inteligencia-artificial-de-google-que-domina-los-videojuegos/
8. I. (2019, February 14). Qué Es El “Machine Learning.” Iberdrola. https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico.
9. Ai, B. (2019, November 27). Intro a las redes neuronales convolucionales - Bootcamp AI. Medium. https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8.
10. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. (2013, December 19). DeepMind Technologies.
11. Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, Stan Birchfield. Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness. 22 jul 2017
12. Toro, L. (2017, 15 septiembre). Anaconda Distribution: La Suite más completa para la Ciencia de datos con Python. Desde Linux. https://blog.desdelinux.net/ciencia-de-datos-con-python/
13. Conda environments — conda 4.9.2.post24+e37cf84a documentation. (2017). https://docs.conda.io/. https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html.
14. David Erosa García. (2019, June 10) Qué es Unity? https://openwebinars.net/blog/que-es-unity/#:~:text=Unity%20es%20una%20herramienta%20que,o%20soporte%20de%20Realidad%20Virtual.
15. Damien, A. (s. f.). TFLearn | TensorFlow Deep Learning Library. TFLearn: Deep learning library featuring a higher-level API for TensorFlow. http://tflearn.org/
16. Bermudez, D. (2020, 25 febrero). Python. Aprender Libre. https://aprender-libre.com/2020/04/python/
17. OpenCV: Introduction to OpenCV-Python Tutorials. (s. f.). Introduction to OpenCV-Python Tutorials. https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html
18. (2019, 27 noviembre). Intro a las redes neuronales convolucionales - Bootcamp AI. Medium. https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8
19. (2019, 14 febrero). Qué Es El «Machine Learning». Iberdrola. https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico.
20. (2018, 24 octubre). Acceder al API de Windows. Recursos Python. https://recursospython.com/guias-y-manuales/winapi-ctypes-pywin32/