Universidad Autónoma de Querétaro

Facultad de Ingeniería

Maestría en Ciencias en
Inteligencia Artificial

Detección automática de líneas de carril basada en Inteligencia Artificial orientada a la asistencia del conductor

TESIS

Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias en Inteligencia Artificial
Presenta:

Tomás Emmanuel Juárez Vallejo

Dirigido por:

Dr. Juan Manuel Ramos Arreguín

Co-Dirigido por:

Dr. Sebastian Salazar Colores

SINODALES

Dr. Juan Manuel Ramos Arreguín		
Presidente		Firma
Dr. Sebastian Salazar Colores		
Secretario		Firma
Dr. Marco Antonio Aceves Fernández		
Vocal		Firma
Dr. Jesús Carlos Pedraza Ortega		
Suplente		Firma
Dr. Efrén Gorrostieta Hurtado		
Suplente		Firma

Dr. Manuel Toledano Ayala	Dr. Juan Carlos Antonio Jauregui Correa
Director de la Facultad	Director de Investigación y Postgrado

Centro Universitario
Querétaro, QRO
México.
Julio 2022

REPOSITORIO INSTITUCIONAL DGBSDI-UAQ

Dirección General de Bibliotecas y Servicios Digitales de Información

Deteccion automatica de líneas de carril basada en Inteligencia Artificial orientada a la asistencia del conductor

por

Tomás Emmanuel Juárez Vallejo

se distribuye bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional.

Clave RI: IGMAC-300600-0123-123

A mis padres y hermanos, sin su apoyo y ayuda este trabajo no hubiera sido posible.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por proveer los fondos para esta investigación y maestría.

También me gustaría agradecer a mis asesores: el Dr. Juan Manuel Ramos Arreguín y el Dr. Sebastian Salazar Colores, por su guianza y apoyo en este proyecto.

Abstract

Roads can present different weather conditions or alterations such as dirt or weariness, which hinder the visibility of lane lines and can cause accidents, they can be reduced by driver assistance systems that perform detection of lane lines in case of an accidental lane departure an alarm is activated. In recent years, several detection methods have been proposed one of them being detection by artificial vision with implementation of Aritificial Intelligence . In this work two neural networks were proposed which were trained for the Tusimple database, the first trained model was a generative adversarial network called Pix2pix, for this network two preprocessing methods were used: changes in the color space and a clustering algorithm called Superpixel, with this model a Dice index of 0.41 was obtained. The second model consists of a modified U-net, the preprocessing techniques used were: Changes in the color space and a different number of channels in the image. A new loss function called Focal Tversky + L1 was proposed, in the results, a value in the Dice index of 0.83 was obtained and in the official Tusimple metrics an accuracy of 0.92 was achieved while its training time was 15 seconds per epoch. Finally, the implementation was carried out on a Raspberry Pi 3 card using the TensorFlow lite library.

Keywords: *Artificial Intelligence, Artificial Neural Networks, Embedded systems, Lane detection.*

Resumen

Las carreteras pueden presentar distintas condiciones climáticas o alteraciones como suciedad o desgaste lo cual dificulta la visibilidad de las líneas carriles pudiendo casuar accidentes, estos se pueden disminuir mediante sistemas de ayuda al conductor los cuales realizan una detección de las líneas de carril y se emite una alarma en caso de un abandono de carril accidental. En los últimos años se han propuesto diversos métodos de detección siendo una de ellas la detección mediante visión artificial con el uso de redes neuronales. En este trabajo se propusieron dos redes neuronales las cuales fueron entrenadas para la base de datos Tusimple, el primero modelo entrenado fue una red generativa adversarial llamada Pix2pix para esta red se utilizaron dos preprocesamientos: cambios en el espacio de color y un algoritmo de agrupamientos llamado Superpixel, con este modelo se obtuvo un índice Dice de 0.41. El segundo modelo consiste en una U-net modificada, se utilizaron distintas técnicas de preprocesamiento: Cambios en el espacio de color y diferente número de canales en la imagen. Se propuso una nueva función de perdida llamada Focal Tversky + L1, en los resultados se obtuvo un valor en el índice Dice de 0.83 y en las métricas oficiales de Tusimple se logró una precisión de 0.92 mientras que su tiempo de entrenamiento fue de 15 segundos por época. Finalmente se llevo a la implementación en una tarjeta Raspberry Pi 3 utilizando la libreria TensorFlow lite.

Palabras clave: *Inteligencia Artificial, Redes Neuronales Artificiales, Sistemas Embebidos, Detección de carriles.*

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

1.1. Justificación

Los accidentes de tráfico suelen ser una de las principales causas de muerte en el país, siendo esta la primera causa de muerte para personas de edades entre 5 y 29 años, y la quinta en la población en general [1]. En 2019 el 64.85 % de los accidentes viales fueron causados por error humano, debido a fatiga, distracciones, conducción a exceso de velocidad, salud física del conductor, entre otros (CNS, 2015) (INEGI, 2019).

Los sistemas de asistencia al conductor pueden ayudar a disminuir los accidentes viales, un ejemplo de esto es un sistema de alerta de abandono del carril, generando un estímulo al detectar que el coche no está circulando en sus límites designados. Cuando esto ocurra el conductor puede realizar una maniobra correctiva en caso de ser por distracción o fatiga, en consecuencia teniendo una reducción significativa en los accidentes viales [2] [3].

A pesar de que se han desarrollado varios algoritmos para la detección de carriles, no todos se han llevado a la implementación, limitándose a simulaciones o pruebas con bases de datos. Se propone en este trabajo probar y en su caso modificar al menos un algoritmo para la detección de líneas de carril, con el propósito de crear un sistema de ayuda al conductor, que, en caso de un abandono accidental del carril debido a fatiga o distracción emita un estímulo para recuperar la atención del conductor a la carretera. También se propone su implementación en un sistema embebido.

1.2. Antecedentes

Conforme la tecnología ha ido avanzando, los medios de transporte también se modernizan, y esto ha causado el desarrollo de una variedad de mejoras, por ejemplo: mayor eficiencia en los motores, mejor seguridad para los pasajeros, asistencias al conductor, entre otros. Esto ha generado menores accidentes de tránsito, ahorro de combustible y un viaje más cómodo. Recientemente, se han desarrollado vehículos autónomos, en los cuales no hay intervención humana, los cuales a través de sensores y cámaras reconocen su ambiente.

En el desarrollo de vehículos autónomos se requiere que el vehículo tenga la capacidad de reconocer sus entornos y el camino por el cual va a circular. Estas capacidades pueden ser aplicadas hacia la asistencia al conductor, por ejemplo, emitiendo una alerta en caso de abandono accidental del carril. Comúnmente en este tipo de sistemas, se utiliza la detección de carriles, cuya función principal es determinar si el vehículo se encuentra dentro del carril.

En 2004 Wang et al. encontraron que la detección de carriles puede presentar algunos problemas debido a las condiciones del camino, por ejemplo: tipos de iluminación, falta de líneas delimitantes para los carriles, entre otros [4]. Tomando estos factores en cuenta propusieron que las propiedades principales que una técnica de detección de carril debe tener son:

- No debe ser afectada por sombras.
- Debe ser capaz de procesar caminos pintados y sin pintar.
- Debe detectar curvas.
- Debe usar las indicaciones paralelas en ambos lados del camino.
- Debe producir una medición explicita sobre la confiabilidad de sus resultados.

Existe una gran variedad de técnicas para la detección de carriles, por lo que es necesario que estos métodos sean estudiados en diferentes situaciones para evaluar su desempeño en condiciones adversas. Debido a que el estado de los caminos es muy variable, el objetivo de estos estudios es cerciorarse que sean adaptables a cualquier tipo de situación.

En 2004 se usó una técnica basada en B-snake para lograr una detección robusta de curvas, su enfoque fue un algoritmo confiable en caminos con curvas y en cambios de iluminación [4].

En 2008 se desarrolló una técnica para la detección de líneas en tiempo real, utiliza técnicas de filtrado y un algoritmo RSC (Random Sample Consensus), que puede trabajar con líneas desgastadas y cambios de carril en intersecciones [5].

En 2008 se utilizó una cámara montada en un vehículo para proporcionar las imágenes necesarias para su algoritmo, el cual está basado en detección de bordes y en la transformada de Hough, los resultados experimentales mostraron que es adecuado para su uso en tiempo real [6].

En 2016 se dividió la detección en 2 pasos; en el primero se estiman los puntos de fuga mediante un método basado en extracción de texturas; el segundo paso se encarga de detectar las líneas del carril y se usan los puntos de fuga para delimitar el algoritmo; este método no se ve afectado por las variaciones de iluminación y condiciones en el camino [7].

En 2018 se propuso hacer uso de la información geométrica que se puede obtener mediante el área del carril y sus delimitaciones, se implementó un framework que proporciona los datos geométricos y una función propuesta para trabajar con ellos [8] (Zhang, Xu, Ni & Duan , 2018).

Con el desarrollo de nuevas investigaciones se han ido mejorando los métodos de detección y los algoritmos usados se vuelven cada vez más sofisticados. Algunos algoritmos empiezan a incorporar

herramientas de Inteligencia Artificial con el propósito de obtener una medición más confiable en distintas condiciones. Los métodos tradicionales obtienen buena precisión en los ambientes para los que fueron diseñados, pero batallan con cambios en iluminación o clima [9].

La técnica conocida como Deep Learning o Aprendizaje profundo es un conjunto de métodos de Inteligencia artificial que ha sido ampliamente utilizado en la detección de carriles, y uno de sus métodos más usados han sido las RNC (Redes Neurales Convolucionales). Las RNC tienen la capacidad de ignorar ruido en las imágenes, con la desventaja de requerir para su entrenamiento una base de datos amplia y una cantidad de tiempo considerable; para reducir esta desventaja se han utilizado distintas técnicas, por ejemplo, MAE (Máquina de Aprendizaje Extremo), DAP (Destilación de Atención Personal), entre otras.

En 2017 se implementó una RNC con un algoritmo MAE logrando una reducción en el tiempo de aprendizaje y una mejora en el desempeño) [10]. En [11] incorporaron un algoritmo DAP lo cual le permite a la RNC aprender de sí misma mejorando su rendimiento.

En 2019 se utilizó una red basada en LaneNet lo cual reduce el tiempo de cómputo [12]. Liu et al. [13] basaron su algoritmo en Mask R-CNN para lograr un mejor entrenamiento, en sus pruebas se obtuvo una precisión del 97.9 %. Choi et al. usaron Aprendizaje Profundo para un sistema de asistencia al conductor el cual emite una alarma en caso de que se salga del carril, pero no está implementada la alarma con el uso de inteligencia artificial [14].

Cao et al. lidiaron con este problema proponiendo un nuevo método de detección el cual utiliza una visión área de los carriles para obtener las regiones de interés; posteriormente es procesado mediante un algoritmo RSC y finalmente es acoplado a las curvas con B-spline. Las pruebas realizadas con la base de datos "Tusimple" revelan que se logra un 98.49 % de precisión [15].

En 2020 se implementó un algoritmo en una Raspberry pi 3B, con una cámara incorporada al automóvil para la obtención de datos, el algoritmo fue basado en técnicas de filtros Gabor y transformada de Hough. Se logró un 93.67 % de precisión en la detección de carriles y un 95.24 % en el aviso de abandono de carril (Teo, Sutopo, Lim & Wong , 2020).

En [16] se usó una división en los pasos con la diferencia en que implementaron técnicas basadas en detección de bordes y regresión polinomial para la primera parte y transformadas de perspectiva y análisis de histogramas para el segundo paso.

Conforme nuevas técnicas o modelos se van desarrollando, estas han sido aplicadas a este problema, Ghafoorian et al. implementaron un modelo neuronal conocido como Red Generativa Antagónica (GAN) obteniendo un entrenamiento mas estable debido a las características de la red [17]. En 2020 Zhang et al. propusieron una red llamada Ripple-GAN la cual fue probada con distintas bases de datos obteniendo resultados favorecedores. En la Tabla 1.1 se puede ver algunas de las propuestas del estado del arte y los algoritmos utilizados.

Tabla 1.1: Comparativa del estado del arte

Año	Titulo	Algoritmos usados	Pre-Procesamiento	Tiempo real	Resultados
2017	Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection	Convolutional Neural Networks	Extracción de bordes, Random Sample Consensus, Region of interest	sí	Se logro una reducción en tiempo de aprendizaje
2018	Towards End-to-End Lane Detection: an Instance Segmentation Approach	Convolutional Neural Networks, Clustering	Extracción de bordes,Vista de águila	no	Se logra una detección en la base de datos TuSimple
2018	EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection	GAN, Clustering	-	no	Se obtuvo una detección de líneas usando una GAN.
2020	Lane detection technique based on perspective transformation and histogram analysis for self-driving cars	Convolutional Neural Networks	Análisis de histograma, Transformada de Hough	sí	Se logra la implementación en una tarjeta Raspberry pi
2021	Ripple-GAN:Lane Line Detection With Ripple Lane Line Detection Network and Wasserstein GAN	CNN, GAN	RippleLane Line	no	Se logró la detección de líneas en diferentes datasets.

1.3. Descripción del Problema

Los estados de los caminos y carreteras pueden tener mucha variación en cuanto a sus tipos de asfalto o concreto, es deseable que las líneas del camino se encuentren definidas, como se muestra en la Figura 1.1 donde los carriles son completamente visibles. Debido al uso o falta de mantenimiento se presentan alteraciones en las marcas de los carriles, algunas pueden estar desgastadas o pueden aún no estar pintadas debido al mantenimiento correspondiente, como se puede ver en la Figura 1.2, lo cual representa un riesgo para los usuarios de la carretera.

Figura 1.1: Carretera con carriles visibles

Figura 1.2: Carretera sin carriles visibles

Los cambios de iluminación, clima, condiciones de tráfico, entre otros, son situaciones que se deben considerar al momento de realizar la detección de carriles, ya que afecta la calidad de la imagen o a la visibilidad de los carriles. Para lograr una detección correcta se requiere de un método del

estado del arte capaz de desempeñarse en condiciones adversas y de gran variabilidad. Para tratar con estos problemas se han propuesto distintos algoritmos de inteligencia artificial, debido a su capacidad de adaptarse a distintos ambientes. Sin embargo, llevarlo a la implementación presenta diversos retos, entre ellos el tiempo de cómputo.

Por esto, se pretende buscar la manera de implementar al menos un algoritmo, realizando las modificaciones necesarias para su implementación en un sistema embebido, obteniendo como resultado un sistema de alerta para alertar al conductor por un abandono accidental del carril.

1.4. Hipótesis

Los algoritmos basados en inteligencia artificial pueden ser modificados para su implementación en un sistema embebido, para realizar procesamiento de video, y poder detectar las líneas de carril de un camino, así como emitir una alarma por abandono de carril.

1.5. Objetivos

El objetivo general es realizar la implementación de un algoritmo de inteligencia artificial en un sistema de abandono de carril, mediante el uso de un sistema embebido.

1.5.1. Objetivos Específicos

Los objetivos específicos son:

- Seleccionar el método de obtención de imágenes de carreteras, para generar una base de datos usando cámaras para sistema embebido.
- Proponer el pre-procesamiento adecuado para obtener la información requerida de las imágenes realizando diversas pruebas de acuerdo con la literatura.
- Realizar pruebas con los algoritmos de detección para evaluar su desempeño, usando imágenes propias y de bancos de datos públicos.
- Modificar el algoritmo de detección de acuerdo con los resultados obtenidos para ser implementado en un sistema embebido.
- Realizar la elección de la tarjeta de desarrollo a utilizar conforme a los requerimientos de cómputo y periféricos necesarios para la implementación del algoritmo.

CAPÍTULO 2

Fundamentación Teórica

2.1. Visión por computadora

La visión por computadora es un campo de estudio que trata de obtener información de nuestro entorno mediante imágenes. Esto es considerado un problema inverso, donde se trata de recuperar información desconocida en un plano, es decir, se intentan reconstruir las propiedades de una imagen como pueden ser: forma, iluminación, distribución de color, entre otros. Para obtener estas propiedades se utilizan filtros, algoritmos, fórmulas matemáticas, entre otros szeliski2022. Algunas de sus aplicaciones son:

- Reconocimiento de caracteres
- Segmentación de imágenes
- Vigilancia
- Evaluación de riesgos.

Debido a que al momento de captar una imagen se obtiene una gran cantidad de datos, por lo que procesar toda la información es un proceso tardado y aumenta significativamente el tiempo de cómputo. Se usan diferentes métodos para obtener las características deseadas. Esto es llamado pre-procesamiento y ayuda a mejorar la respuesta de los algoritmos al analizar los datos de la imagen. También puede ayudar a reducir ruido o diversos problemas que pueda tener la imagen. Dependiendo de la información que se quiera extraer, se pueden aplicar filtros, modificaciones a la imagen, extracción de bordes, entre otros [18].

2.1.1. Formación de una imagen

Para poder formar una imagen es necesario tener al menos una fuente de luz, y se debe tomar en cuenta intensidad, posición y distribución de la fuente de luz. De la misma manera se tienen que considerar las propiedades reflectivas del objeto o la escena a capturar, esto se ve representado en la Figura 2.1. El hardware óptico juega un papel importante en la formación de la imagen, pero generalmente se puede considerar como ideal [19]

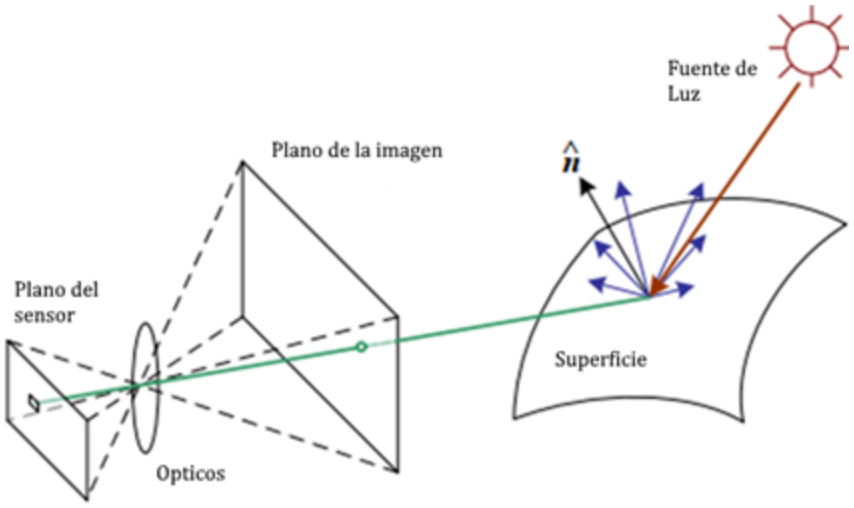


Figura 2.1: Formación de una imagen [19].

Para formar una imagen con una cámara digital, la luz debe ser capturada y dirigida al sensor, donde es transformada a una matriz con los valores de iluminación. Si la imagen es en blanco y negro la matriz contendrá los valores de la opacidad, y si es una imagen a color, se tienen que capturar 3 ondas de luz distintas por cada pixel, generando tres matrices: R (Red), G(Green) y B(Blue). La Figura 2.2 muestra los tres componentes de color RGB de una imagen. El rango de los valores en las matrices RGB puede variar dependiendo del estándar utilizado, pero comúnmente va de 0 a 255.

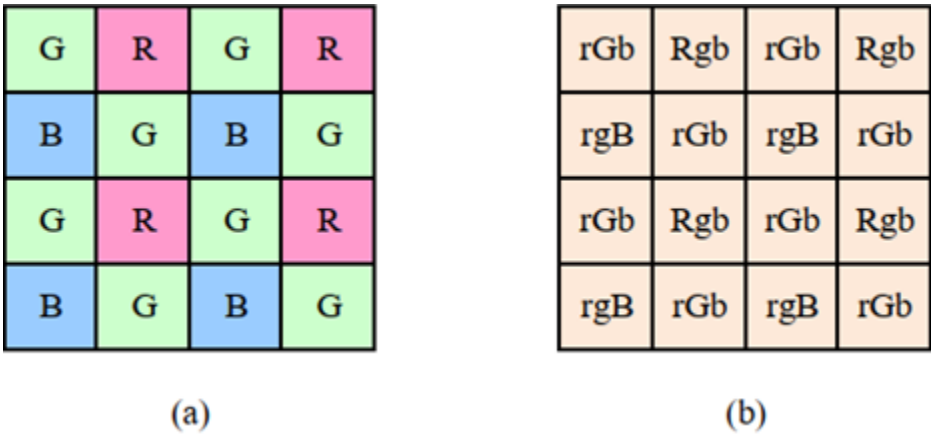


Figura 2.2: Matrices con valores RGB [19].

2.1.2. Espacios de color

Un espacio de color es una representación o interpretación de los colores la cual se utiliza al almacenar una imagen de forma digital, existen diferentes espacios para representar distintas gammas de color o tener la capacidad de aislar características deseadas, algunos de estos espacios son: RGB, CIELAB, HSV, entre otros. Los espacios de color se han utilizado en los modelos tradicionales de detección de carriles ya que se encuentran generalmente pintados en amarillo o blanco, el cambio en espacio de color ayuda a resaltar los colores deseados y mitigar los cambios de iluminación en los carriles [16].

2.1.3. RGB

El espacio de color RGB es un modelo aditivo, el cual se encuentra dividido en *Red, Green, Blue* o Rojo, Verde y Azul. Se dice que es modelo aditivo debido a que esta conformado por los colores primarios y que con la correcta combinación de ellos se puede conseguir un color especifico, en la Figura 2.3 se puede observar un ejemplo de este espacio.

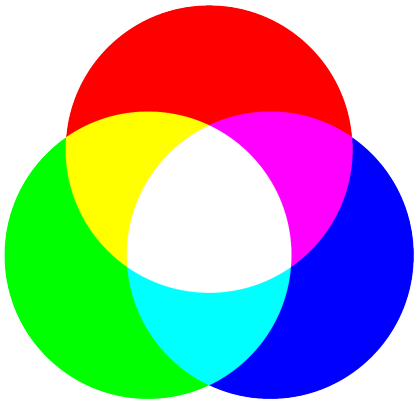


Figura 2.3: Espacio de color RGB [16]

2.1.4. HSV

El espacio de color HSV es una representación alternativa del modelo RGB diseñada en 1970 y se basa en la percepción humana del color, se encuentra divido en *Hue, Saturation, Value* o Matiz, Saturación, Brillo, en la Figura 2.4 podemos ver la representación de este espacio. Se ha probado que este espacio de color es efectivo en distinguir e inhibir sombras que se presentan en el RGB, ha sido utilizado en la visión por computadora debido a que el componente de matiz es insensible a las sombras y cambios de iluminación. [20].

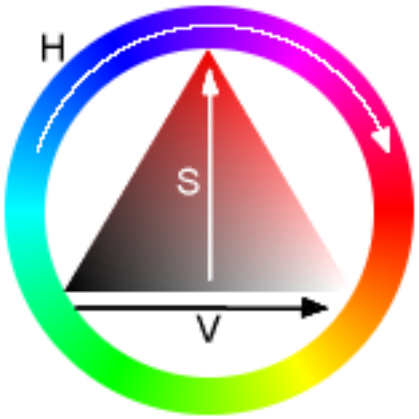


Figura 2.4: Espacio de color HSV 2.4

2.1.5. CIELAB

CIELAB es un espacio de color que describe todos los colores visibles para el ojo humano fue diseñado por la comisión internacional en iluminación, esta dividido en *L, a, b*, donde L es la luminosidad perceptiva,la combinación de los colores esta conformada de la siguiente manera: rojo y verde para a y azul y amarillo para b, al ser separada la luminosidad este espacio de color se considera invariante a los cambios de luz en las imágenes ya que los cromas del color están representados en a y b [21], una representacion de este espacio se puede ver en la Figura 2.5.

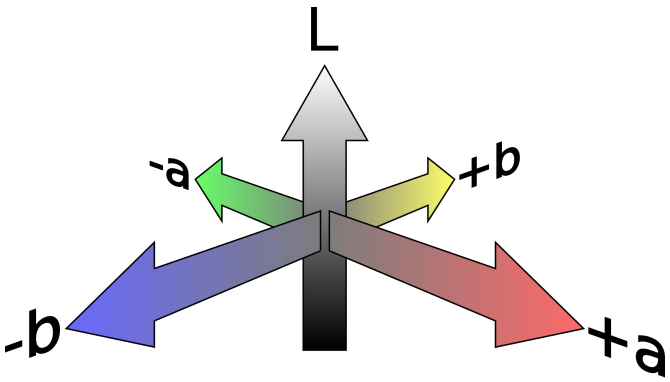


Figura 2.5: Espacio de color CIELAB [21]

2.2. Filtros Morfológicos

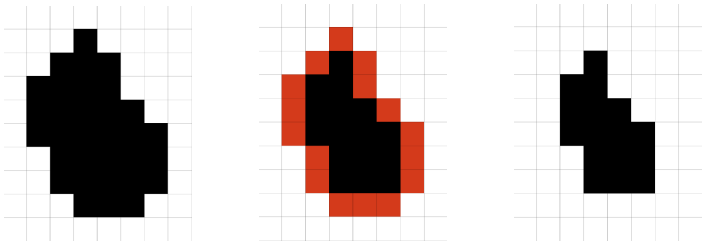
La morfología matemática esta conformada por varias metodologías las cuales buscan describir la estructura geométrica en los objetos de la imagen, esto dio pie a un nuevo campo llamado filtros morfológicos el cual ha sido aplicado en distintos campos llegando al procesamiento digital de imágenes debido a su eficiencia, estas operaciones son aplicadas en imágenes binarias. Algunas de las operaciones básicas del filtrado morfológico son: Erosión, Dilatación, Apertura y Cierre.

2.2.1. Erosión

Consiste en eliminar los pixeles de los bordes de una imagen, esto se ejecuta para todos los pixeles que tengan al menos un pixel adyacente que forme parte del fondo. su objetivo es reducir el tamaño de los objetos o eliminar ruido en la imagen, se encuentra descrito por la ecuación 2.1 [22], en la Figura 2.6 se puede ver un ejemplo de esta operación.

$$A \ominus B \tag{2.1}$$

Donde A es la imagen y B es el kernel a utilizar



(a) Imagen original

(b) Seleccion de los bordes mediante la erosión

(c) Imagen con erosion aplicada

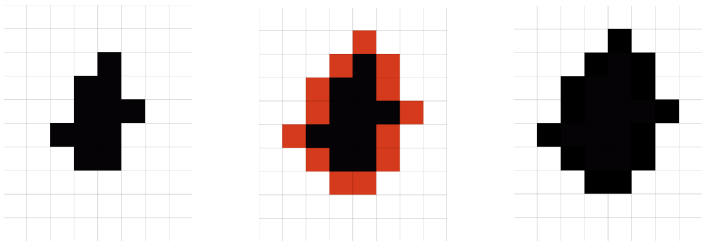
Figura 2.6: Operación de Erosión [22]

2.2.2. Dilatación

La dilatación podría decirse que es la operación contraria a la erosión ya que en este este caso todos los pixeles del fondo que tengan a un pixel adyacente al objeto se volveran parte de el, su objetivo es aumentar el tamaño de los objetos o rellenar partes pequeñas de el [22]. Esta descrito por la ecuación 2.2 y en la Figura 2.7 se puede observar una representación de la operación.

$$A \oplus B \tag{2.2}$$

Donde A es la imagen y B es el kernel a utilizar



(a) Imagen original

(b) Seleccion de los bordes mediante la dilatación

(c) Imagen con dilatación aplicada

Figura 2.7: Operación de Dilatación [22]

2.2.3. Esqueletonización

La esqueletonización (Skeletonize) es una operación morfológica diseñada para encontrar una figura que represente la forma de un objeto, esto se logra mediante la eliminación de pixeles en el objeto hasta que quede un objeto de un solo pixel de grosor, una representacion de esto se puede ver en la Figura 2.8 en donde el borde negro corresponde a un objeto binario, mientras que las lineas rojas dentro de el sería el resultado de aplicar esta operación.

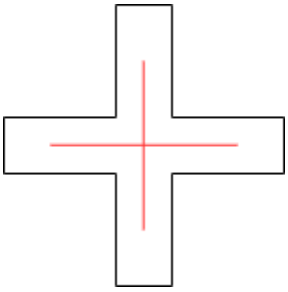


Figura 2.8: Ejemplo de Esqueletonización [22]

Esto se puede lograr mediante varias técnicas algunas de ellas son: realizar un mapa de las distancias de los objetos mediante puntos limite, calcular el diagrama Voronoi generado por los puntos limite del objeto, aplicar repetidamente la operación morfológica erosión hasta lograr la figura deseada.

2.3. Redes Neuronales

Una red neuronal es un modelo que trata de emular el modo en el que el cerebro humano procesa la información, estas redes son compuestas por varias capas de neuronas. La estructura mas básica de la red es una sola neurona la cual recibe el nombre de perceptron, esta compuesta de: entradas, pesos, sumatoria, una función de activación y su salida, una representación gráfica se pude observar en la Figura 2.9.

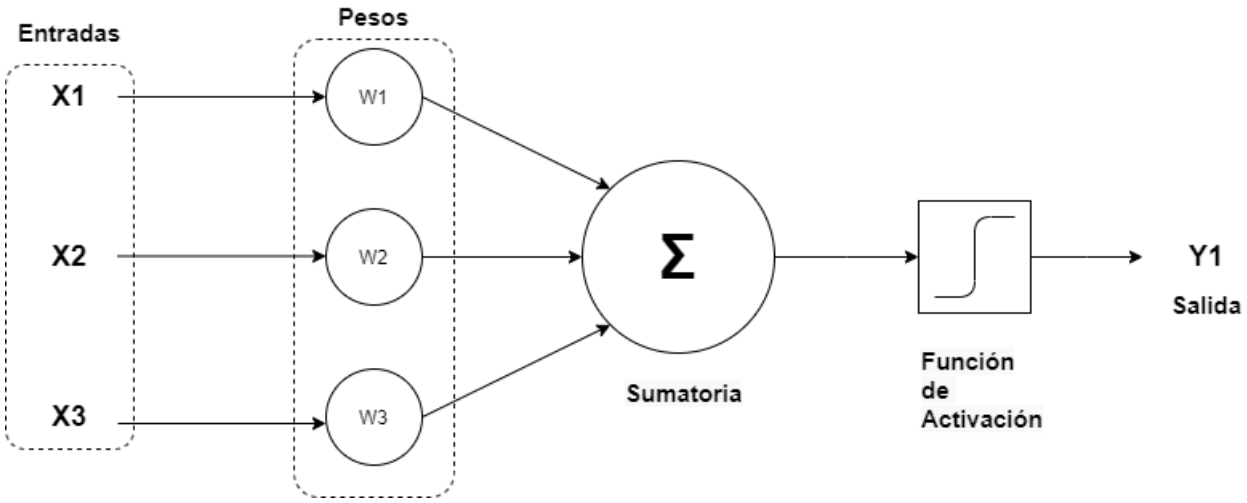


Figura 2.9: Neurona perceptron [23]

Cuando se tiene una red neuronal profunda se aplican varias capas de las neuronas perceptron, la primera capa llamada capa de entrada recibe los valores reales que alimentan a la red neuronal, la ultima capa llamada capa de salida es el resultado del procesamiento que tuvieron los valores durante mediante la red, finalmente las capas que se encuentran entre la capa de entrada y la de salida son llamadas capas ocultas [23], esto se puede visualizar en la Figura 2.10

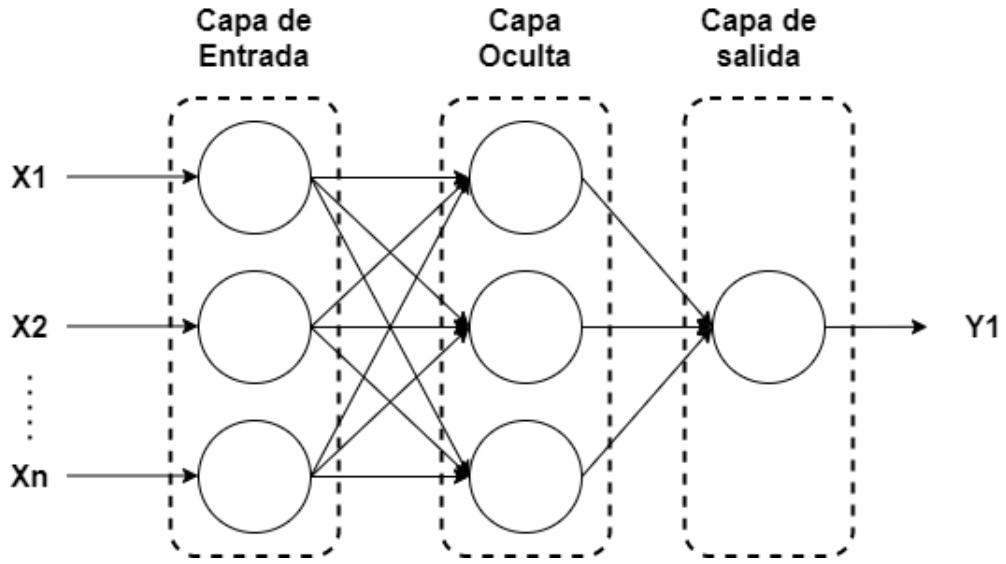


Figura 2.10: Red Neuronal Multicapa [24]

2.3.1. Funciones de pérdida

Las redes neuronales profundas aprenden a cotejar ciertas salidas con un conjunto de entradas con las bases de datos de entrenamiento, con cada época calculada los pesos de las neuronas se van ajustando hasta lograr un resultado satisfactorio, sin embargo, no es posible calcular manualmente

los pesos perfectos ya que se tienen demasiadas variables, para esto es utilizado un algoritmo de optimización.

Las funciones de perdida son usadas para evaluar las salidas que se van generando con la red neuronal, con esto se busca minimizar o maximizar el resultado obtenido dependiendo de la función a evaluar. El problema principal es diseñar una función que capture las propiedades del problema a resolver y den un buen espacio de búsqueda[24].

La segmentación de imágenes se puede considerar como una tarea que se lleva a cabo a nivel píxel, ya que cada imagen esta conformada por un conjunto de píxeles, en estos conjuntos se definen los elementos de las imágenes. Uno de los métodos para clasificar estos conjuntos es llamado Segmentación Semántica. La selección de la función de perdida para este método es muy importante por lo que se ha experimentado con funciones en distintos dominios para mejorar los resultados de las redes [25].

Para este trabajo se propone utilizar las siguientes funciones de prueba para la segmentación de lineas de carril:

- Focal loss
- Dice loss
- Tversky loss
- Focal Tversky loss

Focal loss

Esta función disminuye el impacto de iteraciones sencillas para la red y permite que el modelo se enfoque mas en iteraciones que presentan mas complejidad, presenta buenos resultados para las situaciones que se tienen un imbalance de clases [25]. La ecuación que define esta función se puede ver en 2.3

$$Fl(p_t) = -\alpha_t(1-p_t)^{\gamma} log(p_t) \tag{2.3}$$

donde $\gamma > 0$ y α tiene un rango de [0,1]

Dice loss

El índice Dice [26] es ampliamente usado como métrica para calcular la similitud entre dos imágenes, se puede representar como 2 veces la intersección de 2 conjuntos sobre la suma de ellos como se puede observar en 4.2, el cual ha sido adaptado para funcionar como función de perdida, la cual esta representada en la ecuación 2.5

$$Dice = \frac{2|X \cap Y|}{|X| + |Y|} \tag{2.4}$$

$$DL(y,p) = 1 - \frac{2yp+1}{y+p+1} \tag{2.5}$$

donde 1 es agregado en el numerador y denominador para evitar una división entre cero.

Tversky Loss

El índice Tversky[27] fue sugerido para medir las características de los objetos que tienen en común y en cuales se difieren, dándole un mayor peso a las similitudes que a las diferencias, entonces la similitud es definida por un conjunto de características importantes entre los objetos. el índice es descrito mediante 2.6

$$TI(p,b) = \frac{pb}{pb + \beta(1-p)b + (1-\beta)p(1-b)} \tag{2.6}$$

Focal Tversky Loss

Este índice es similar a Focal Loss ya que se enfoca en ejemplos difíciles y le da menos peso a los ejemplos fáciles, manteniendo la separación de características que nos da el índice Tversky, esta función esta descrita por la ecuación 2.7

$$FTL = \sum (1 - TI_x)^{\frac{1}{\gamma}} \tag{2.7}$$

2.3.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (RNC) son una red neuronal aplicada comúnmente en el análisis de imágenes, este algoritmo asigna pesos a ciertos elementos de la imagen para poder diferenciar entre elementos. Este tipo de red neuronal utiliza la operación convolución, la cual es un operador matemático que transforma la matriz mediante un kernel. Podemos ver un ejemplo de esto en la Figura 2.11 [23][19].

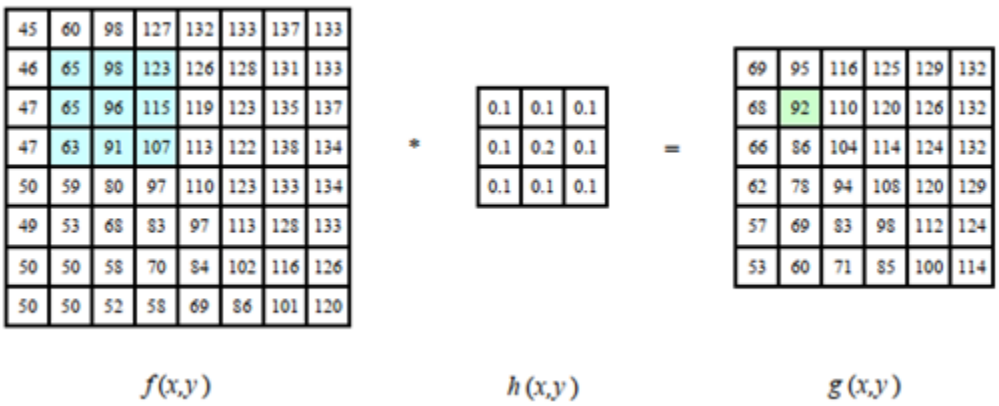


Figura 2.11: Ejemplo de una convolución en una matriz [19]

Una de las características de las RNC es la capacidad de trabajar con entradas de distinto tamaño, mientras que una red neuronal común tiene que relacionar cada entrada con cada salida. Las RNC pueden tener poca conectividad, lo cual nos permite tener una salida más pequeña a la entrada. Esto es útil para poder detectar características específicas que se encuentran en partes pequeñas de la imagen sin tener que analizarla en su totalidad, ahorrando recursos y tiempo de

cómputo [23].

Otra de sus características de las RNC es que puede compartir parámetros, esto significa que en vez de aprender varios conjuntos de parámetros se utiliza solo uno, aunque esto no reduce el tiempo de la propagación si reduce los requerimientos de almacenamiento. Lo cual resulta en un método muy eficiente en términos de requerimientos de memoria [23]. Un ejemplo de una RNC se puede apreciar en la Figura 2.12 propuesta por [28].

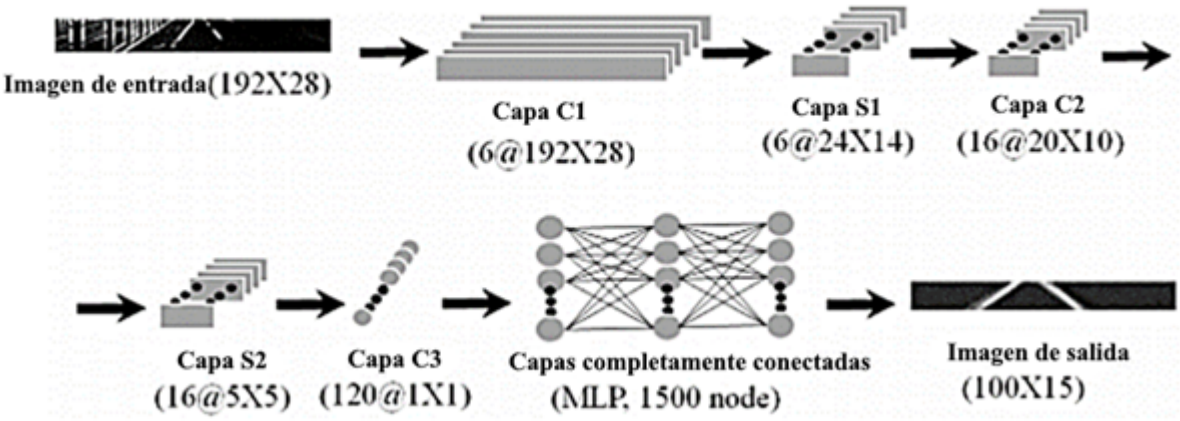


Figura 2.12: Diagrama del procesamiento mediante RNC [28]

2.3.3. U-net

Es una arquitectura basada en redes convolucionales, presentada por la universidad de Freiburg en 2015. Fue diseñada para la segmentación semántica de imágenes, consiste contraer y expandir la imagen. La primera parte la cuales es la contracción de la imagen esto se logra mediante capas de convoluciones seguidos por una función de activación ReLU y operaciones de Max Pooling, mientras que la expansión esta dada por mapas de caracteristicas seguidas por una convolución finalmente se utiliza una función de activación ReLu [29]. La estructura de esta red se puede ver en la Figura 2.13, la cual toma su nombre por la forma que esta tiene.

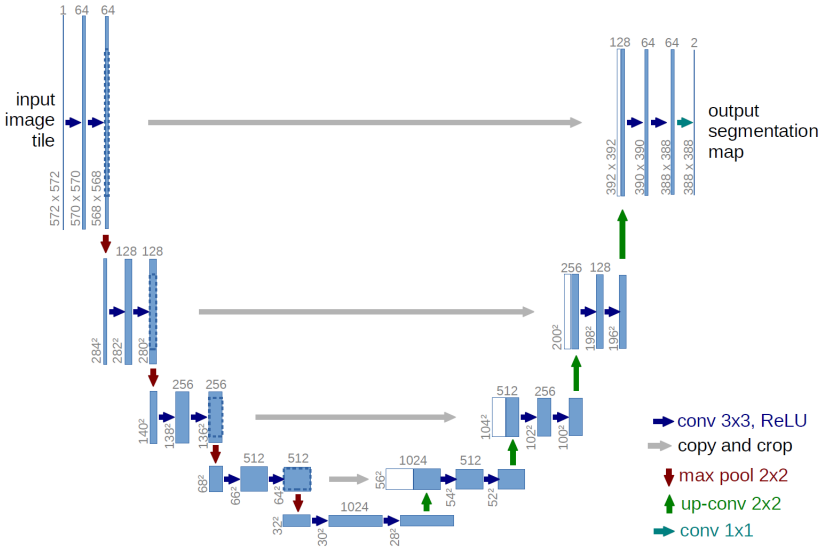


Figura 2.13: Arquitectura U-net [29]

Originalmente fue diseñada para su uso en segmentación de imágenes biomédicas aunque con algunas modificaciones se ha podido ampliar su uso a distintas aplicaciones fuera el ámbito médico, algunos ejemplos de las aplicaciones son[30]:

- Segmentación semántica de imágenes aéreas
- Detección de objetos
- Clasificación de objetos

2.3.4. Redes Generativas Antagónicas

Las Redes Generativas Antagónicas(GAN) utilizan dos redes neuronales profundas las cuales son catalogadas como: Generador (G) el cual captura la distribución de la información y un Discriminador(D) que es usado para detectar si la distribución corresponde a la información original o si es creada por G, esto se puede observar en la Figura 2.14. Al ser redes antagónicas se puede decir que realizan un "juego"de suma cero en donde lo que una red gana la otra lo pierde, esto es medido mediante la ecuación (2.8).

$$\min_G \min_D V(D,G) = \mathbb{E}_{x\ p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\ p_{data}(z)}[\log(1 - D(G(z)))] \tag{2.8}$$

donde $p_{z(z)}$ son las entradas de distribución para D y $p_{data(x)}$ son ejemplos de las distribuciones de probabilidad a aprender por G

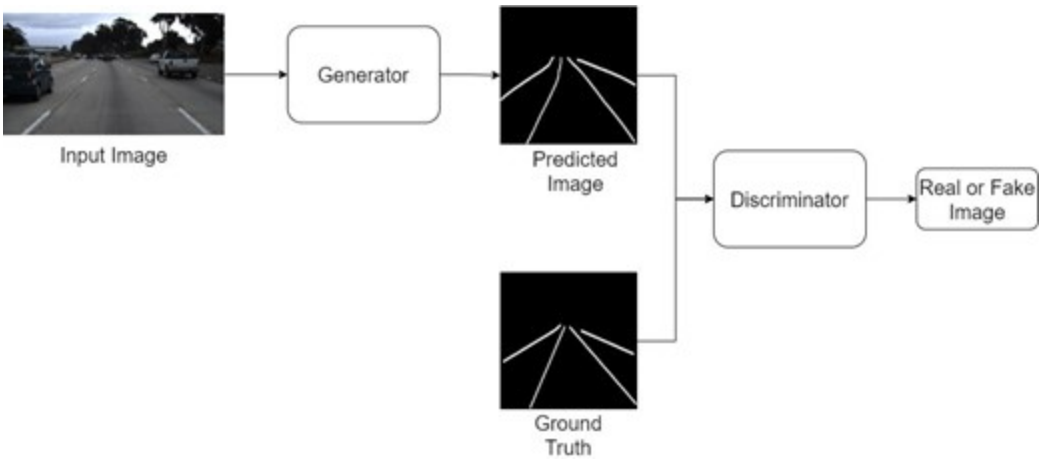


Figura 2.14: Modelo simple de una red GAN

Redes Generativas Antagónicas Condicionales

Una variante en la arquitectura de las GAN donde una condición previa y es establecida, es llamada una Red Generativa Antagónica Condicional($CGAN$). Su función de perdida esta compuesta por las funciones (2.9) y (2.10)

$$\mathcal{L}_{GAN}(G, D) = \log D(x, y) + \log(1 - D(G(x, y), y)) \tag{2.9}$$

$$\mathcal{L}_{L1}(G) = |y - G(x, y)| \tag{2.10}$$

Finalmente uniendo estas dos funciones obtenemos:

$$\mathcal{L}_{CGAN} = \mathcal{L}_{GAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{2.11}$$

donde $\lambda > 0$ es la importancia de L_{L1}

2.4. Modelos Utilizados

2.4.1. Pix2Pix

EL modelo Pix2Pix esta conformado por una red *GAN* el cual utiliza una U-net como generador y una PatchGan como discriminador. Esta red ha sido utilizada en una variedad de escenarios en los cuales se requiere generar una imagen de salida con respecto a una de entrada, algunos de ellos son: Lineas a fachadas, vista aérea a mapas, etc. [31].

El Generador y discriminador están definidos como:

- **Generador** La arquitectura del generador es una U-net modificada la cual consiste en una red codificadora-decodificadora que comparte información entre sus capas. El codificador de la red está dado por un bloque de downsample y el decodificador por un bloque de upsample. El downsample está conformado por las capas: Convolución 2D, Batch Normalization y leaky ReLU como función de activación, las capas del decodificador son una Convolución Transpuesta, Batch Normalization y ReLU como función de activación, se aplica un dropout en las 3 primeras capas. Una representación del Generador puede verse en la Figura 2.15, donde el bloque amarillo representa la capa de entrada, el bloque rojo representa la operación downsample, el verde la operación upsample y el azul es la convolución transpuesta.

- **Discriminador** El discriminador está diseñado para una estructura de alta frecuencia utilizando la pérdida L1 para corregir las frecuencias más bajas. El discriminador necesita restringir la atención del modelo en lotes de la imagen, intenta clasificar cada lote para determinar si la imagen es real o falsa, para obtener el resultado final promedia todos los lotes. En la Figura 2.16 se ilustra la arquitectura del discriminador, donde los primeros bloques amarillos representan las entradas, el siguiente bloque rojo representa una capa de concatenación, despues los colores representan diferentes capas del modelo, el bloque verde representa una capa secuencial, los bloques azules Zero Padding 2d, el negro una capa de convolución y la capa amarilla indica batch normalization.. El discriminador recibe dos entradas, El Ground Truth, y la imagen predicha, utiliza un bloque compuesto por una Convolución, Batch Normalization y ReLU como función de activación, el tamaño del lote es de 30 x 30 píxeles.

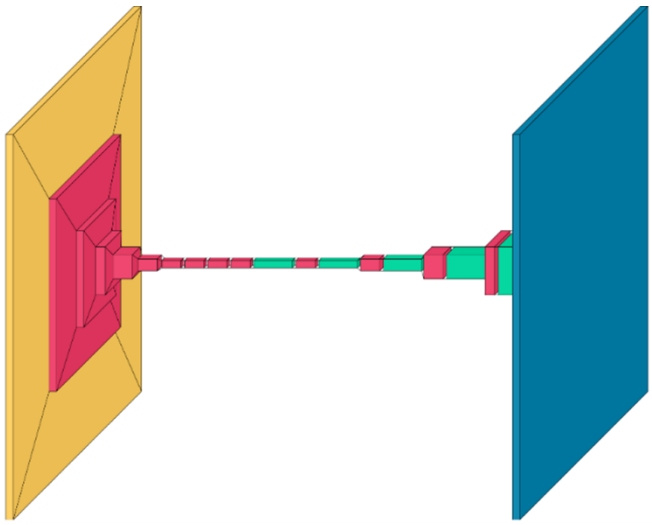


Figura 2.15: Modelo del Generador Pix2Pix [31]

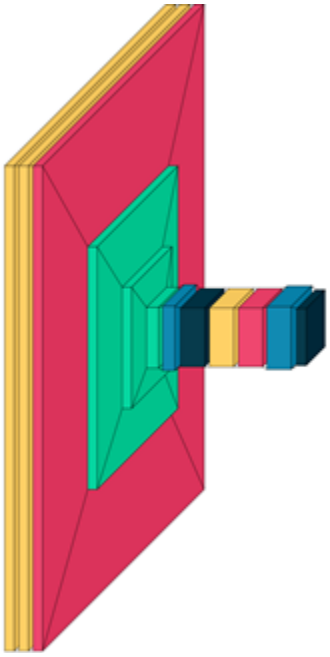


Figura 2.16: Modelo del Discriminador Pix2Pix [31]

2.4.2. CIONet

La estructura de la red neuronal CIONet consiste en una U-net modificada, mantiene las funciones de expandir y contraer la imagen de entrada para aprender sus características, sin embargo la red puede recibir imágenes en blanco y negro como a color, de la misma manera la red es mas profunda, una representación se puede observar en la Figura 2.17, donde el primer bloque amarillo representa la imagen de entrada, posteriormente los bloques rojos representan una convolución, los

verdes representan una capa de batch normalization, los azules un dropout en la red, los bloques negros son una operación de maxpooling, los bloques amarillos muestran una capa de convolución transpuesta.

Las imágenes de entrada para esta red son de tamaño: 256x256 ó 512x512, puede procesar imágenes con distintos canales, pero para este estudio se utilizaron unicamente 1 y 3 canales. Para esta red se utilizaron diversas funciones de perdida creadas para la segmentación semántica de imágenes, algunas de estas funciones son: Dice loss, Focal loss, Tversky Loss.

Se configuraron 2 funciones llamadas callbacks las cuales nos permiten alterar el entrenamiento de la red regulando algunos hiper-parametro. Las funciones son: *learning rate reduction* su objetivo es evitar el estancamiento de el optimizador en algún mínimo local reduciendo el factor de aprendizaje cuando la métrica seleccionada se mantiene en un valor por 10 épocas. *Early Stoping* el objetivo es detener el modelo para evitar un sobre entrenamiento, esta función también restaura los mejores pesos que se encontraron durante el entrenamiento.

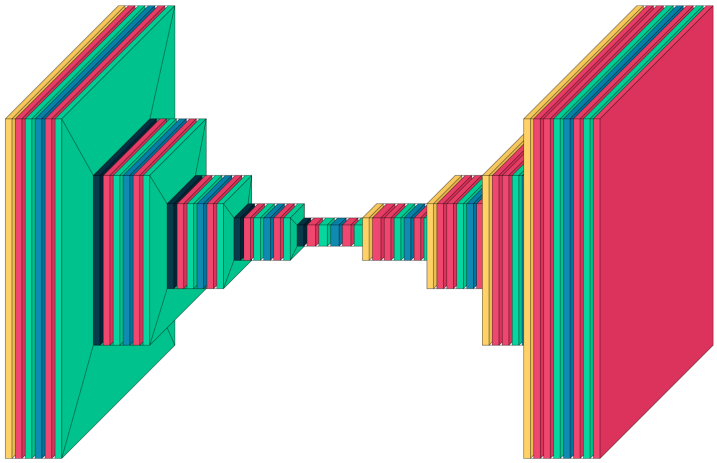


Figura 2.17: Modelo CIONet

2.5. Micro-controladores

Un micro-controlador circuito integrado que puede ejecutar comandos programados por un usuario. Los micro-controladores son una parte central de lo que es conocido como sistemas embebidos, estos sistemas son diseñados para un propósito en especifico y sus aplicaciones son muy amplias siendo algunas de ellas: control de vehículos, robots, impresoras, equipamiento medico, etc. para lograr esto utilizan uno o mas dispositivos como sensores o actuadores los cuales permiten la obtención de información y la interacción respectivamente[32].

Debido a que sus aplicaciones es en diversos campos y en algunos casos en ambientes o requisitos especializados los micro-controladores pueden tener variaciones en sus características, sin embargo, se tienen elementos que son considerados esenciales los cuales estan listados a continuación y una representación de la estructura básica de un micro-controlador se puede observar en la Figura 2.18 [33].

- Procesador (CPU)
- Memoria (RAM, ROM)
- Periféricos I/O

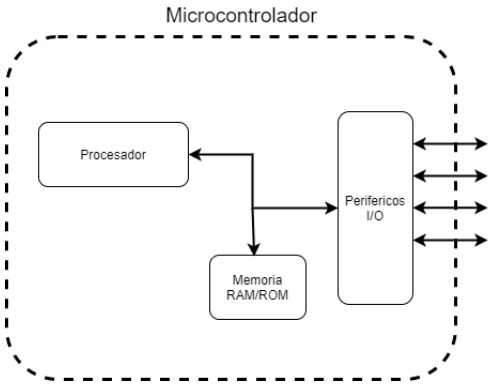


Figura 2.18: Arquitectura básica micro-controlador [33]

Una de las principales variaciones de un micro-controlador es su procesador, el cual puede ir desde 4 bits hasta procesadores mas complejos de 32 o 64 bits dependiendo de la aplicación, también puede variar la cantidad de memoria y la cantidad de pines para los periféricos I/O, así como algunos módulos por ejemplo: Conversor Analógico a Digital(DAC), Esta flexibilidad en sus características ha permitido la implementación de distintos algoritmos en ellos, un ejemplo de esto son las redes neuronales las cuales debido a su alto consumo computacional son difíciles de implementar.

Para lograr la implementación algunas compañías han desarrollado micro-controladores con la capacidad de ejecutar redes neuronales, algunos de estos son: Coral Dev Board, Nvidia Jetson, Jevois. También se desarrollaron aceleradores de hardware para permitir que micro-controladores con menos recursos puedan correr los modelos neuronales, algunos de ellos son: Coral USB, Intel Neural Stick 2 [34].

CAPÍTULO 3

Metodología

La metodología para este trabajo consiste en 6 partes como se muestra en la Figura 3.1, *Definición de imágenes*, en donde se analizaron las diferentes bases de datos existentes para el problema a abordar y se seleccionó una de ellas. *Procesamiento de imágenes*, en este punto se investigaron los distintos pre-procesamientos utilizados en el estado del arte o que podrían ayudar a resaltar las características deseadas. *Implementación de Modelos*, Se implementaron los modelos de redes neuronales a utilizar para la segmentación de los carriles en las imágenes.*Pruebas con los Modelos*, en este paso se realizan los experimentos probando distintos parámetros y funciones de perdida en las redes neuronales así como distintos pre-procesamientos para las imágenes, los resultados de las configuraciones son medidas mediante el indice Dice, cuando se tienen resultados favorables se continua a la siguiente fase. *Post-procesamiento*, debido a que la predicción de la red neuronal es una segmentación semantica nos da unicamente una imagen binarizada pero para obtener la evaluación final se requiere tener los carriles con una clase individual, para esto se aplica el post-procesamiento de la imagen. *Evaluación final*, Una vez que se tiene la imagen con los carriles individuales se aplican las métricas oficiales de la base de datos Tusimple para poder comparar la red con el estado del arte.

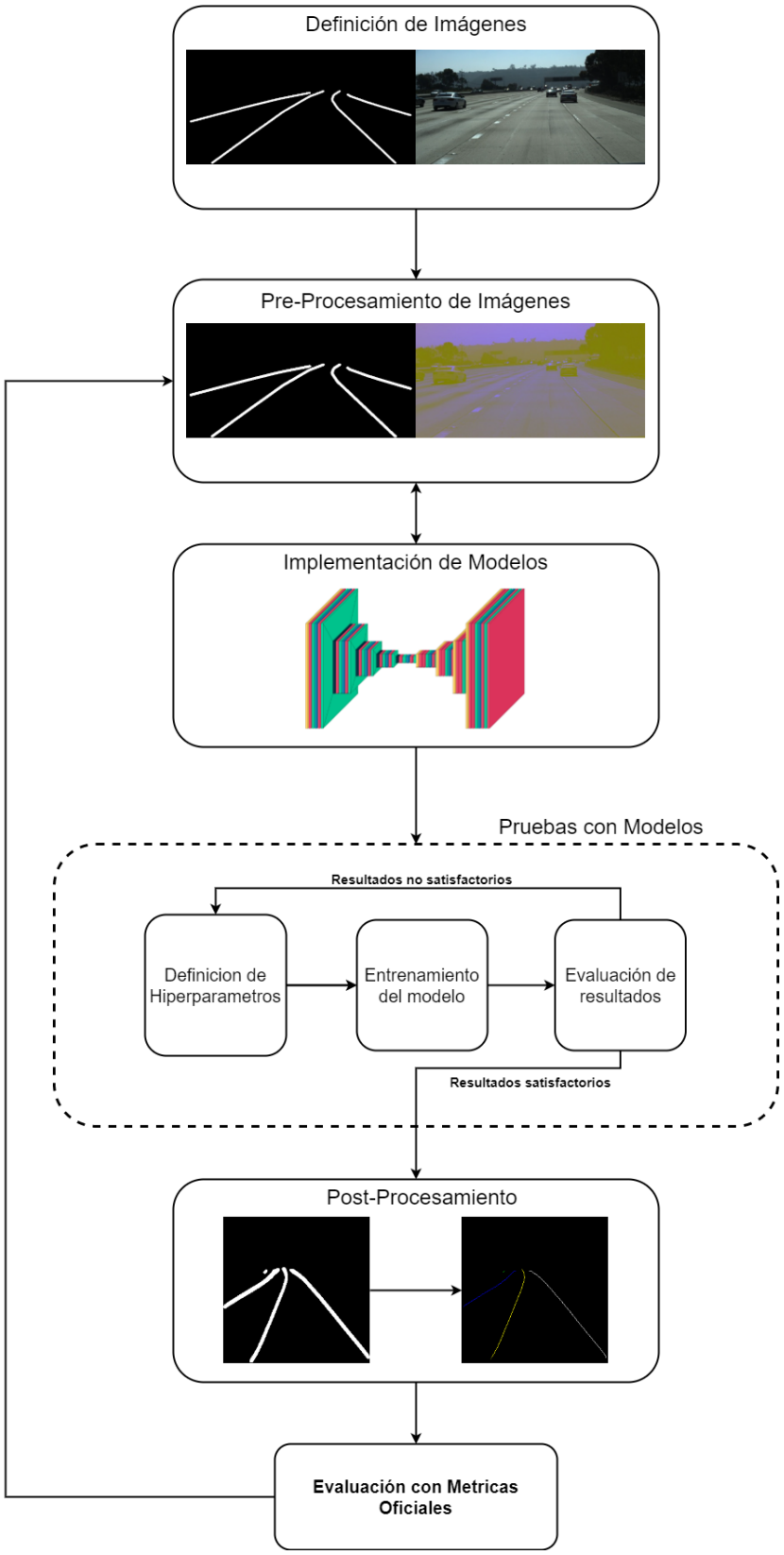


Figura 3.1: Metodología utilizada

3.1. Definición de imágenes

Las bases de datos analizadas fueron: Tusimple, CUlane y BD100k, finalmente se decidió utilizar Tusimple debido a las limitaciones físicas del hardware con el cual se trabajará. TuSimple (Tusimple, 2017) [35] consiste en videos tomados en carreteras de Estados Unidos de América con una resolución de 1280 x 720, algunas de sus características son: variedad de climas, diferentes condiciones de tráfico y diferente número de carriles. Cuenta con 3626 imágenes de entrenamiento y 2782 imágenes de prueba. Las etiquetas de las líneas de los carriles se encuentran en un archivo JSON.

Para entrenar los modelos: Pix2pix y CIONet, se requiere una imagen que consta de dos partes: La etiqueta y la imagen de entrada, para cumplir con este requisito se creó una imagen con los carriles a partir de las etiquetas del conjunto de datos con una resolución de 1280 x 720 píxeles y un tamaño de línea de 12 píxeles, la cual fue concatenada con la imagen original, después de esto las imágenes fueron redimensionadas a una resolución de 1024 x 256, un ejemplo de esto se puede ver en la Figura 4, donde A es la etiqueta de la imagen y B la imagen de entrada.

Figura 3.2: Ejemplo de imagen de entrenamiento [35]

3.1.1. Creación de Base de datos UAQuiLane

Debido a que la base de datos Tusimple fue obtenida en Estados Unidos de América, surgió la necesidad de crear una base de datos con carreteras locales para probar que el modelo entrenado pueda funcionar en el país México, con esto se crea la base de datos UAQuiLane la cual consiste en 42 vídeos tomados en las carreteras del estado de Querétaro, cuenta con tomas en carretera y en ciudad, de estos vídeos se extrajo un frame por cada 30 dando un total de 4,00 imágenes, cuenta con condiciones variadas como sombras, trafico y carreteras sin marcar, sin embargo, aun no se encuentra etiquetado por lo que solo se puede usar para pruebas sin evaluaciones. En la Figura 3.3 se pueden observar algunos ejemplos de esta base de datos.

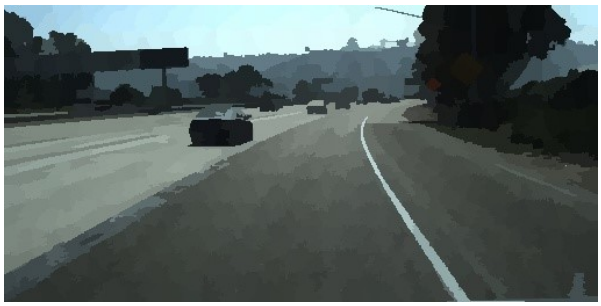
Figura 3.3: Ejemplos de la base de datos UAQuiLane

3.2. Pre-procesamiento de imágenes

Las técnicas de pre-procesamiento seleccionadas para este tipo de problema se basaron en los métodos tradicionales para la detección de lineas en los carriles. Estos métodos buscan resaltar las características de la linea principalmente el color y la forma, de la misma manera se busca mitigar los cambios en la iluminación de la carretera para obtener una detección mas robusta [16]. Las técnicas utilizadas fueron el cambio en espacio de color pasando de RGB a HSV y CIELAB y se uso el algoritmo de agrupamiento superpixel. Estos métodos se aplicaron por separado en las imágenes de entrada formando una base de datos distinta por cada método, los distintos pre-procesamientos se pueden observar en la Figura 3.4.

(a) Imagen Original

j



(b) Algoritmo Superpixel

(c) Espacio de color HSV

(d) Espacio de color Cielab

Figura 3.4: Pre-procesamiento de imágenes

3.3. Implementación de Modelos

La implementación de los modelos fue dividida en tres partes, el modelo Pix2Pix se llevó a cabo en la plataforma de Google: Google Colab, la cual nos permite utilizar una computadora de alta gama mediante computo en la nube, se utilizó esta plataforma debido a que el modelo requiere una amplia capacidad de computo con la cual no se contaba al inicio. Para el modelo CIONet se utilizó una computadora de escritorio con una tarjeta de video Nvidia RTX 3060 , 24 GB de memoria RAM y un procesador Ryzen 5 3600, se utilizó el sistema operativo Windows 10 y Python 3 con el programa de desarrollo Pycharm.

3.4. Pruebas con los Modelos

Las pruebas con los modelos se pueden dividir en 3 partes: Definición de Hiperparametros, Entrenamiento del modelo y la Evaluación de los resultados. En la Definición de los Hiperparametros se modifican diferentes valores para cada red neuronal. Para la arquitectura Pix2Pix es posible modificar el optimizador Adam y la función de perdida la cual esta descrita en 2.11, sin embargo, al modificar el optimizador Adam no se obtuvieron resultados satisfactorios por lo que se decidió dejar fijo en un valor de $2e^{-4}$ y la variable λ de la función de perdida su fijó en 80.

La imagen de entrada se probó sin alteraciones y con cambios en el espacio de color siendo estos: Blanco y negro, HSV, y CIELAB. en la Figura 3.5 se puede observar el flujo de trabajo con este modelo, para cada prueba se fue realizando un registro de los resultados.

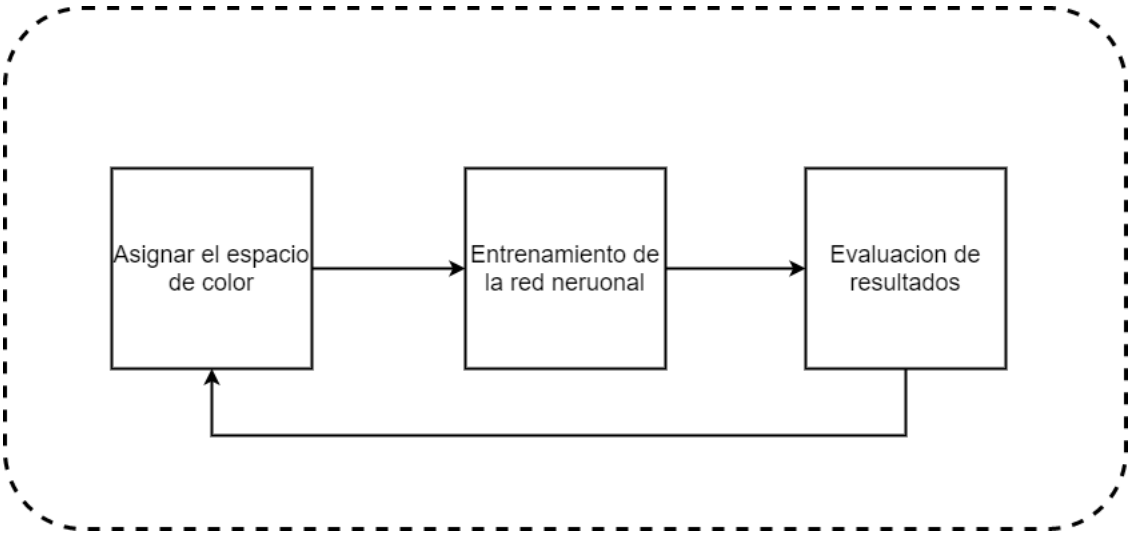


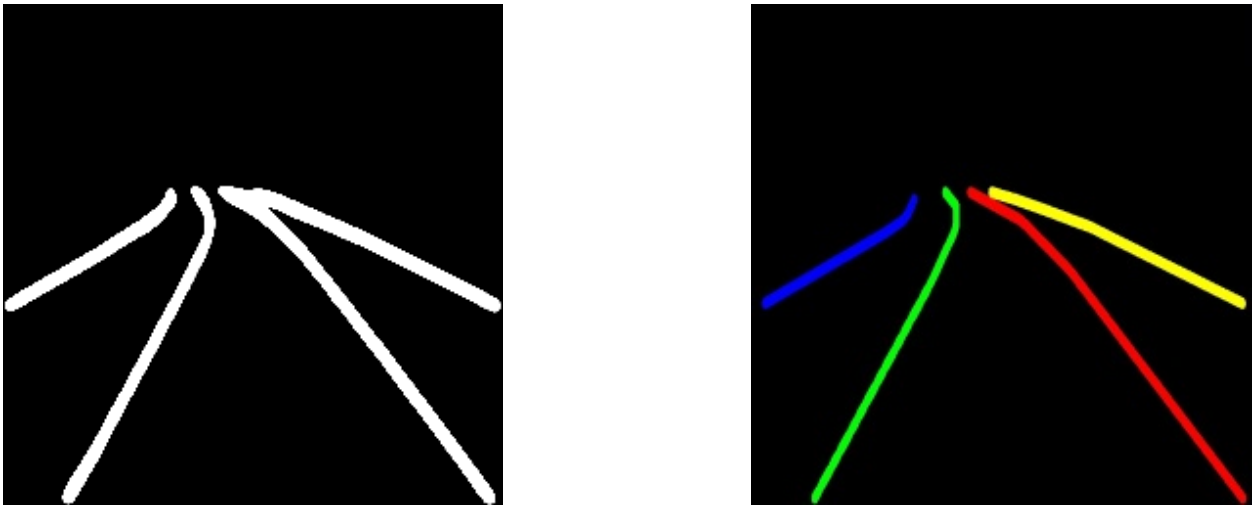
Figura 3.5: Diagrama de Entrenamiento para el modelo Pix2pix

Para la red CIONet es posible modificar los siguientes parámetros: Optimizador Adam, numero de canales, función de perdida, numero de épocas, pasos por época y tamaño del lote. Debido a que variar una gran cantidad de parámetros consumiría mucho tiempo se decidió dejar fijo el opimizador Adam con un valor de $2{,}5-3$, el numero de épocas en 150, los pasos por época en 35 y tamaño del lote en 24, modificando en las pruebas unicamente el numero de canales y los hiperparametros de las funciones de perdida.

Para el numero de canales con excepción del procesamiento en blanco y negro, se utilizó el valor 1 y 3 para cada espacio de color, mientras que el hiperparametro seleccionado en la función de perdida Focal Tversky Loss descrita por la ecuación 2.7, se le dio un valor inicial de 70 el cual se fue modificando en un factor de 10 puntos, inicialmente de manera ascendente y posteriormente de manera descendente.

3.5. Post-procesamiento

Una vez que se tiene la red entrenada se obtienen imágenes binarias de la segmentación de carriles correspondientes a la predicción de la base de datos de pruebas, sin embargo, para la evaluación con las métricas oficiales de la base de datos Tusimple es necesario tener un archivo Json, para obtener esto se requiere una imagen con los carriles divididos por clase, en la Figura 3.6 se puede apreciar la diferencia entre la predicción de la red y la imagen requerida.



(a) Predicción de la red

(b) Carriles separados por clase

Figura 3.6: Segmentación de carriles

Una vez que se tiene la imagen con los carriles separados es necesario pasar los carriles a un archivo en formato Json, el cual debe contener los puntos de los carriles predicción por la red divididos por altura y clase, para lograr esto se utilizó la siguiente metodología: *Separación de carriles por color*, *Esqueletonizacion de la imagen* y *Transformación a Json.* En la Figura 3.7 se puede observar un diagrama de la metodología seguida.

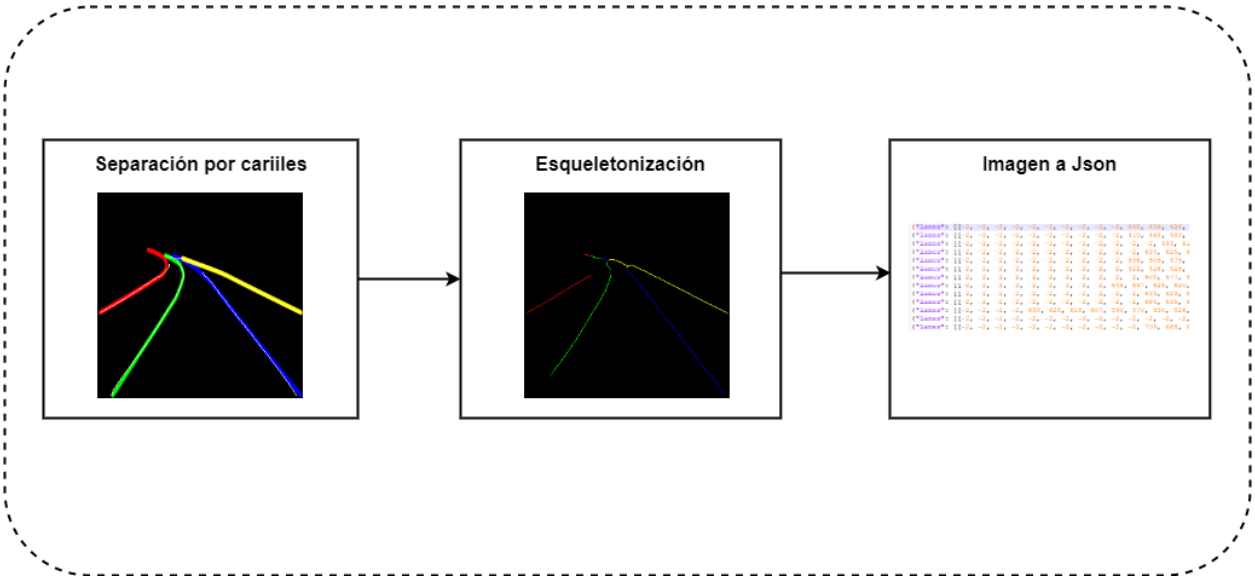


Figura 3.7: Metodología para transformar la imagen a formato Json.

3.5.1. Separación de carriles por color

La separación de carriles se realiza mediante la multiplicación de la imagen predicha por la red neuronal con la etiqueta a color, con el objetivo de obtener unicamente los carriles donde la red neuronal predijo correctamente los carriles utilizando este método se consigue ese objetivo y separar los carriles por color, la formula utilizada para esta operación se ve representada en la ecuación 3.1.

$$\frac{((Img_{Pred} * 255) * Img_{Label})}{255} \tag{3.1}$$

3.5.2. Esqueletonización de las imágenes

En esta fase se aplica la operación morfológica de Esqueletonizacion a la imagen para obtener lineas mas delgadas. la función utilizada es llamada *thin* y se encuentra en la libreria skimage, la diferencia entre esta función y la función *skeletonize* es que la primera permite seleccionar un numero máximo de iteraciones para evitar hacer la linea muy delgada, se definió un máximo de 10 iteraciones debido a que la operación puede borrar predicciones correctas al intentar volver la linea de un solo pixel, un ejemplo de la imagen resultante se puede observar en la Figura 3.8.

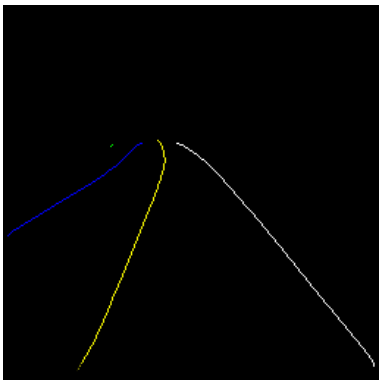


Figura 3.8: Operación de Esqueletonizacion.

3.5.3. Transformación a Json.

Para poder aplicar la evaluación oficial es necesario contar con un archivo Json el cual debe contener los puntos de las lineas del carril, utilizando los siguientes valores: *h_samples* los cuales representan los valores verticales de los carriles van de un rango de $160 - 720$, *lanes* en donde se colocan los valores horizontales de los carriles en un rango de $0 - 1280$ en caso de que no haya un carril se utiliza un valor de -2 , *run_time* es el tiempo que tomó realizar la predicción y *raw_file* es el nombre de la imagen. Se debe de colocar una linea en el archivo por cada imagen, dando un total de 2782 lineas.

Para lograr tener este formato después de aplicar el post-procesamiento se re-dimensiona la imagen a las medidas originales de la base de datos la cual es: 1280 x 720 pixeles debido a que la evaluación oficial está contemplada para imágenes de esa resolución. Para poder obtener los puntos

de los carriles se debe leer la imagen pixel por pixel siguiendo un sentido de la parte superior a la inferior y de izquierda a derecha esto se puede visualizar en la Figura 3.9

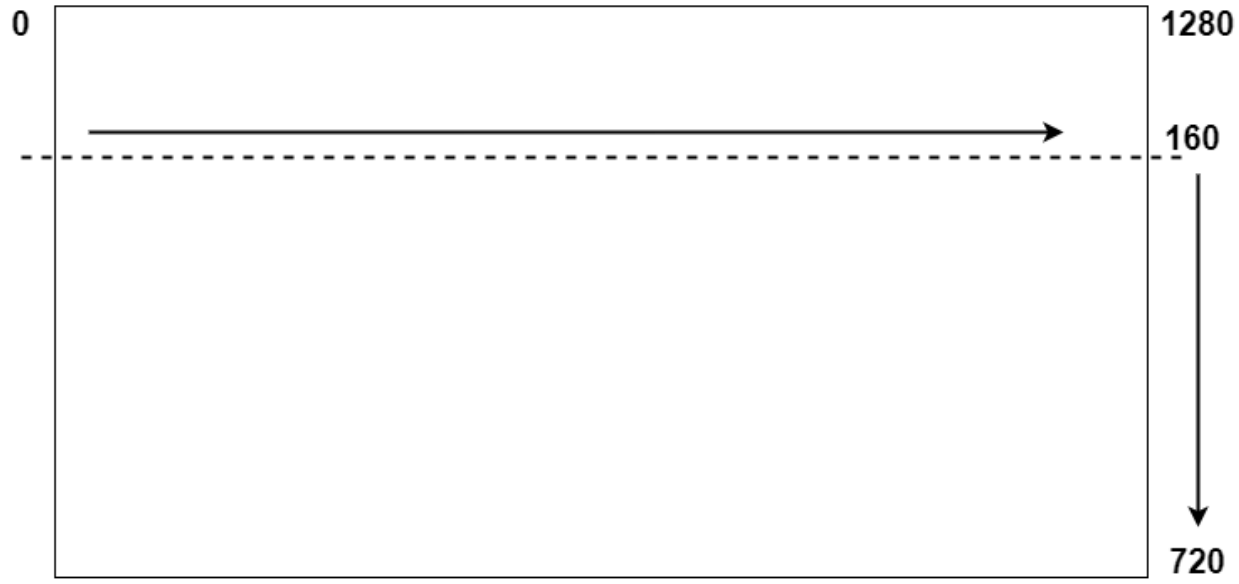


Figura 3.9: Método de análisis para imágenes

Utilizando este método se fue leyendo pixel por pixel de manera horizontal y teniendo un salto de 10 pixeles de forma vertical, si el pixel leído no tenia un valor relevante se guarda un valor -2, en caso de que se encontrará un valor relevante se guarda su valor en un arreglo dependiendo de su color los cuales pueden ser: rojo, azul, verde, amarillo o blanco, finalmente al leer todas las imágenes predichas se guarda el archivo Json.

3.6. Implementación en sistema embebido

3.6.1. Elección de tarjeta

Para realizar la elección de la tarjeta se llevo a cabo una investigación de las tarjetas existentes en el mercado tomando en cuenta su potencia de computo, disponibilidad, precio y compatibilidad con el modelo entrenado. Las tarjetas que fueron consideradas son las siguientes:

- Raspberry pi 3
- Jetson Nano
- Google Coral
- Intel Neural Stick 2
- Asus tinker board

Para evaluar su desempeño se tomo en cuenta la Tabla 3.1, la cual muestra una evaluación realizada en [34] a distintas tarjetas. Se puede observar que el mejor resultado es obtenido por la

tarjeta Jetson Nano seguido por Google Coral y finalmente se encuentran la Intel Neural Stick 2 y la Raspberry pi 3.

Desafortunadamente tanto Google Coral como Jetson Nano, se encontraban en desabasto al momento de realizar este trabajo. La siguiente opción fue la tarjeta Intel Neural Stick 2, sin embargo, utiliza un framework llamado OpenVINO se intentó realizar la conversión del modelo desde Tensorflow pero no fue exitoso, por lo que finalmente se opto por utilizar la Raspberry Pi 3.

Tabla 3.1: Tabla comparativac de desempeño en microcontroladores [34]

Model	Framework	Raspberry Pi 3	Intel Neural Stick 2	Jetson Nano	Google Coral
EfficientNet-B0 (224x224)	TensorFlow	14.6 FPS	95 FPS	216 FPS	200 FPS
ResNet-50 (224x224)	TensorFlow	2.4 FPS	16 FPS	36 FPS	18.8 FPS
MobileNet-v2 (300x300)	TensorFlow	8.5 FPS	30 FPS	64 FPS	130 FPS
Unet (3x257x257)	TensorFlow	2.0 FPS	-	-	-

3.6.2. Metodología

Una vez que se tuvo el modelo entrenado con los mejores parámetros encontrados con la metodología utilizada se propuso una implementación utilizando un sistema embebido. Esto se llevo a cabo mediante una Raspberry Pi 3 y la librería TensorFlow lite, fue necesario convertir el modelo creado con Tensowflow a Tensorflow lite para obtener un mejor desempeño con los recursos limitados del micro-controlador.

La Raspberry 3 modelo B v1.2 cuenta con las siguientes características: Procesador ARM Quad-Core 1.2 Ghz, 1GB RAM , Broadcom Videocore IV GPU y 16 GB de almacenamiento. Se utilizó el sistema operativo Raspberry OS de 32 bits version 5.15, se uso la version de python 3.9 y en cuanto a las librerías se utilizaron: OpenCV 3.4.11.41 y tensorflow lite 2.5.0

Para la implementación se sigue el proceso que se observa en la Figura 3.10. Para obtener la imagen a procesar se utiliza una camara web de marca AVEDISTANTE con conexión USB y una resolución 1920 x 1080 pixeles , posteriormente se aplica un pre-procesamiento el cual consiste en cambiar la resolución de la imagen a 256 x 256 pixeles y transformarla a blanco y negro, después de esto se ingresa al modelo entrenado para crear la inferencia de los carriles finalmente la predicción es mostrada.

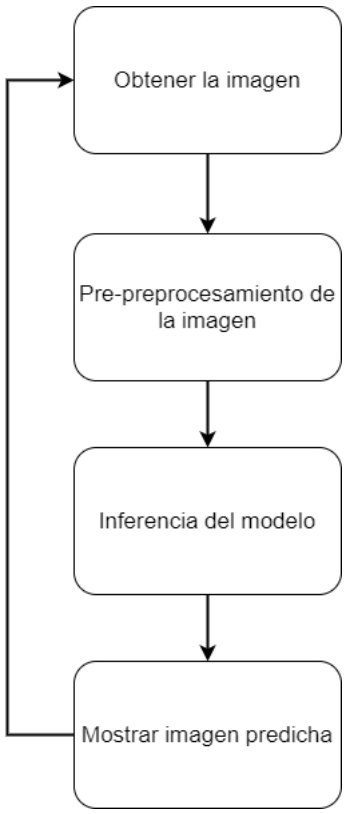


Figura 3.10: Metodología Implementación

CAPÍTULO 4

Resultados

En este capitulo se mostratan los resultados obtenidos de las redes neuronales pix2pix y CIOnet, incluyendo resultados de las métricas utilizadas, las imagenes obtenidas por cada red y tablas comparativas.

4.1. Métricas

Las métricas utilizadas fueron: Indice Jaccard, Coeficiente Dice y la métrica oficial de la base de datos Tusimple. El indice Jaccard y el Coeficiente dice miden la similaridad entre las imágenes en un rango de 0 a 1, siendo 0 la similaridad mas baja y 1 la mas alta.

Estas métricas han sido utilizadas en imágenes medicas y diversos campos de visión por computadora, algunos de sus usos son: evaluación de la segmentación de una imagen, obtener la similitud entre dos conjuntos de datos, entre otros. Tomando en cuenta que tanto la red Pix2pix y CIOnet buscan recrear la etiqueta de la imagen se decidió utilizar este tipo de métricas para evaluar su desempeño.

El indice Jacard esta definido como la intersección de los conjuntos divididos por su unión, esta definido por la siguiente ecuación 4.1.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

El Coeficiente Dice esta definido como 2 veces la unión de los conjuntos divido sobre la sumatoria de los mismos, se encuentra definido por la ecuación 4.2.

$$SDI(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{4.2}$$

La metrica utilizada para la base de datos Tusimple consiste en 56 puntos que representan los valores verdaderos, con esto se calculan: los falsos positivos (FP) y los falsos negativos (FN), una vez que se tienen estos valores son utilizados para calcular el *recall* con la siguiente ecuacion 4.3 y la precisión de cada linea con la ecuación 4.4

$$recall = \frac{TP}{FN + TP} \tag{4.3}$$

$$precision = \frac{TP}{FP + TP} \tag{4.4}$$

4.2. Pi2pix

Como comparativa se decidió utilizar como base la red neuronal LaneNet la cual fue diseñada para la base de datos Tusimple, los resultados base fueron obtenidos al evaluar 400 imágenes de la base de datos de prueba, para este modelo neuronal un simple pre-procesamiento fue aplicado el cual consiste en la binarizacion de las imágenes de salida.

Para la predicción de LaneNet se utilizaron los pesos pre-entrenados, mientras que para el modelo neuronal Pix2Pix se realizo un entrenamiento completamente nuevo, los resultados base los cuales no tienen ningun pre-procesamiento aplicado se pueden ver en la tabla 4.1.

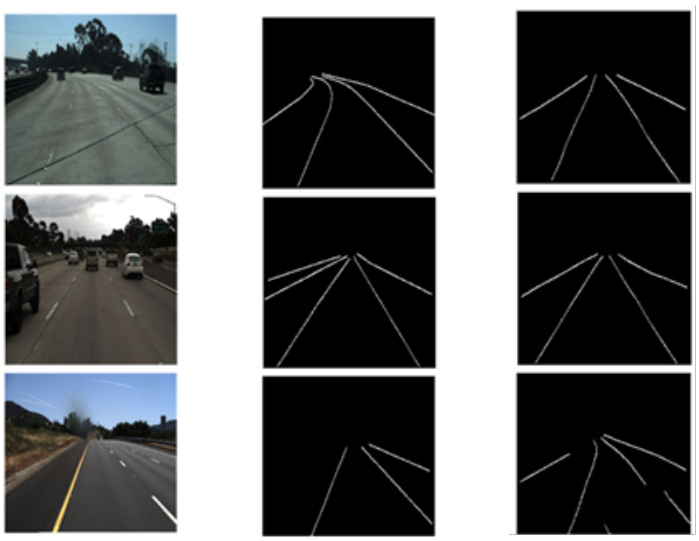
Modelo Base	**Indice Jaccard**	**Coeficiente Dice**
Pix2Pix	0.348	0.499
LaneNet	**0.607**	**0.753**

Tabla 4.1: Modelos base

Para los resultados experimentales se aplico la misma metodogia, sin embargo se fue cambiando el pre-procesamiento que se le da a las imágenes de entrada, con esto se busca encontrar cual es que puede ayudar mas a la red neuronal. Los resultados experimentales están mostrados en la Tabla: 4.2, mientras que los resultados experimentales se pueden ver en la Figura 4.1 para el resultado base, en la Figura 4.2 para el espacio de color CIELAB, en la Figura 4.3 para el algoritmo Superpixel y finalmente en la Figura 4.4 para el espacio de color HSV.

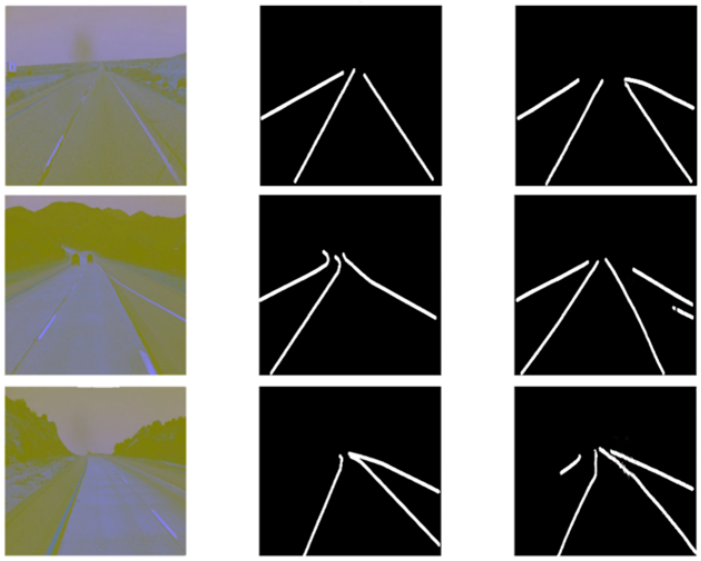
Pre-procesamiento	**Indice Jaccard**	**Coeficiente Dice**
HSV	0.267	0.401
CIELAB	**0.356**	**0.514**
Superpixel	0.322	0.482

Tabla 4.2: Resultados experimentales Pix2Pix



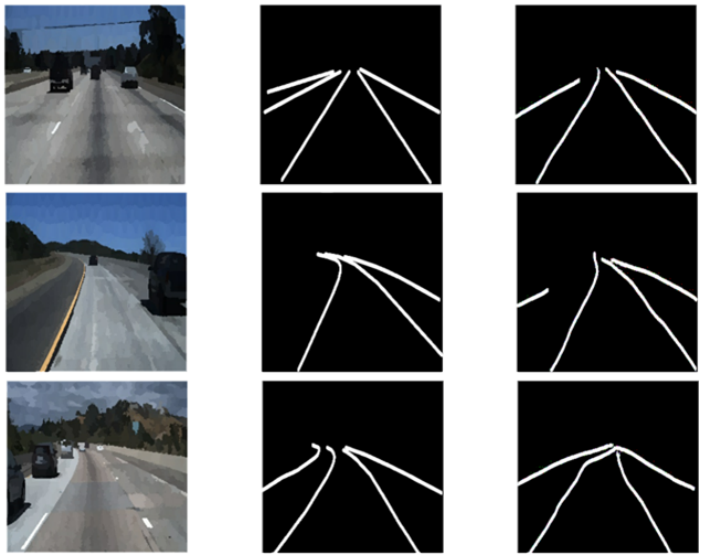
(a) Imagen de entrada (b) Etiqueta de la imagen (c) Imagen Predicha

Figura 4.1: Resultados Base Pix2Pix



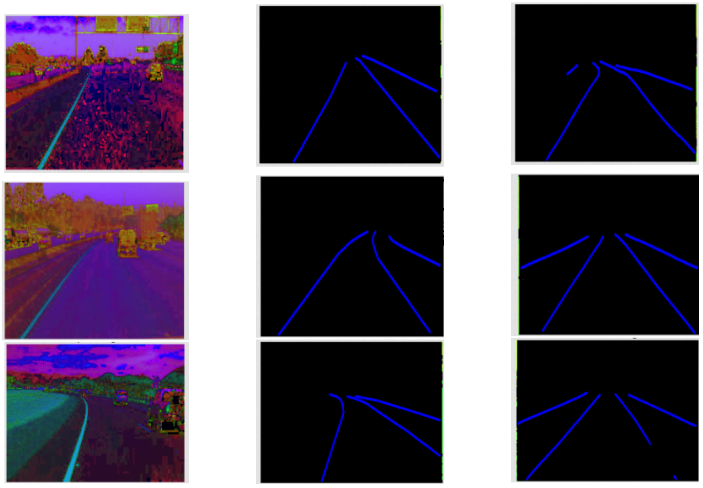
(a) Imagen de entrada (b) Etiqueta de la imagen (c) Imagen Predicha

Figura 4.2: Resultados CIELAB Pix2Pix



(a) Imagen de entrada (b) Etiqueta de la imagen (c) Imagen Predicha

Figura 4.3: Resultados Superpixel Pix2Pix



(a) Imagen de entrada (b) Etiqueta de la imagen (c) Imagen Predicha

Figura 4.4: Resultados HSV Pix2Pix

4.3. CIOnet

Como comparativa para CIONet se decidió utilizar los resultados de la red neuronal Ripple-GAN, la cual fue diseñada para la detección de carriles y ha sido evaluada en la base de datos Tusimple, para el pre-procesamiento se utiliza ruido gaussiano a la imagen de entrada. Para CIO-Net el pre-procesamiento utilizado fue el descrito en la sección 3.2.

Para tener un control sobre las pruebas, se utilizaron resultados base para Ripple-GAN fueron los resultados publicados en su articulo mientras que para CIONet se realizaron pruebas de entrenamiento sin ninguna modificación en los parámetros, tanto de la red neuronal como de la función de pérdida, la configuración de la red neuronal fue la siguiente:

- Numero de épocas: 150
- Pasos por época: 35
- Tamaño de lote: 24
- Numero de canales: 1
- Optimizador Adam: $2{,}5 * 10^{-3}$

Con esta configuración se probaron 3 funciones de pérdida para evaluar su desempeño, estas fueron: Dice Loss, Focal Loss , Tversky Loss y Focal Tversky Loss. Las funciones se evaluaron mediante la métrica del indice Dice, los resultados se pueden ver en la Tabla 4.3.

Tabla 4.3: Resultados Funciones de pérdida

	Dice Loss	Focal Loss	Tversky Loss	Focal Tversky Loss
Indice Dice	0,691	0.659	0.688	**0.712**

Tomando en cuenta los resultados se decidió utilizar la función de perdida: Focal Tversky Loss, para intentar mejorar el resultado de esta funcion de perdida se le agregaron las funciones de perdida L1 y L1 las cuales corresponden al error absoluto medio y el error cuadrático medio, nuevamente se realizaron pruebas con los parámetros iniciales, los resultados se pueden ver en la Tabla 4.4

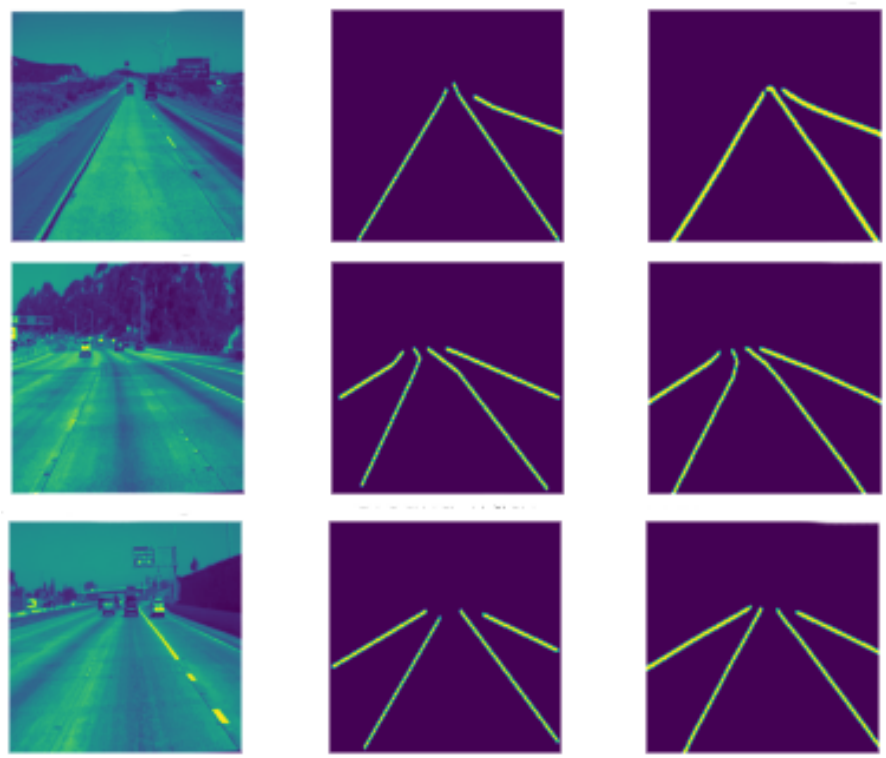
Tabla 4.4: Combinaciones con Focal Tversky

	Focal Tversky	Focal Tversky + L2	Focal Tversky + L1	Focal Tversky + L2 + L1
Indice Dice	0,710	0.799	**0.816**	0.649

Con estas evaluaciones se conoce cual es la mejor función de perdida a utilizar para este problema, por lo que se inician las pruebas con los distintos espacios de color y con el numero de canales en las imágenes, asi como la evaluación con las métricas oficiales Tusimple, los resultados se expondran en dos tablas diferentes, la Tabla 4.5 será para las pruebas con un solo canal, sus resultados se pueden visualizar en la Figura 4.5 y la Tabla 4.6 contienen los resultados para las pruebas con 3 canales.

Tabla 4.5: Resultados imágenes de un canal

	Blanco y Negro	**HSV**	**CIELAB**
Indice Dice	0.8108	**0.8206**	0.8164
Accuracy	0.9313	**0.9330**	0.9323
FP	0.132	**0.126**	0.130
FN	0.145	**0.141**	0.142



(a) Imagen de entrada

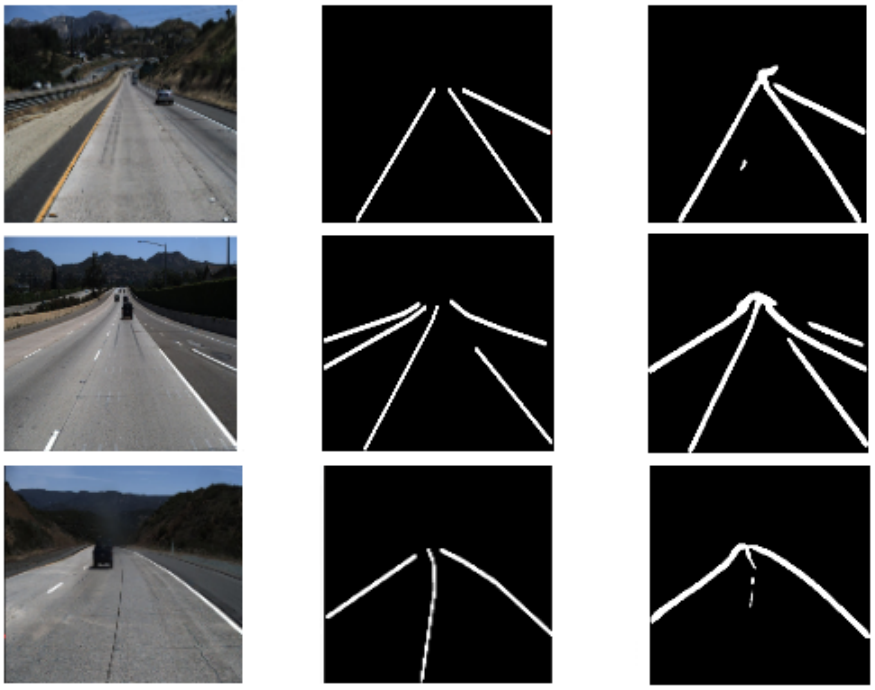
(b) Etiqueta de la imagen

(c) Imagen Predicha

Figura 4.5: Resultados con imágenes de un canal

Tabla 4.6: Resultados imágenes de tres canales

	Color	**HSV**	**CIELAB**
Indice Dice	**0.8108**	0.7786	0.7876
Accuracy	**0.9218**	0.9053	0.9086
FP	**0.137**	0.143	0.139
FN	**0.152**	0.158	0.154



(a) Imagen de entrada

(b) Etiqueta de la imagen

(c) Imagen Predicha

Figura 4.6: Resultados con imágenes de tres canales

Analizando los resultados obtenidos mediante el distinto número de canales se puede observar que la red neuronal tiene un mejor desempeño cuando se utiliza un solo canal con el espacio de color HSV, mientras que utilizando 3 canales en la imagen de entrada baja el rendimiento.

Eligiendo el mejor resultado de estas pruebas se pasa a la siguiente fase de la experimentación el cual consiste en modificar los hiper-parámetros de la función de pérdida, el parametro elegido de la funcion es gamma, la cual originalmente tiene un valor de 75, se decidió aumentar y disminuir su valor en 5 puntos hasta llegar al límite superior de 90 y al limite inferior de 60, los resultados se dividieron fueron divididos en 2 tablas, en la Tabla: 4.7 se puede observar el aumento en el hiper-parámetro Gamma y en la Tabla 4.8 el decremento.

Tabla 4.7: Resultados en la modificación del parámetro gamma, limite superior

	Gamma: 75	**Gamma: 80**	**Gamma:85**	**Gamma: 90**
Indice Dice	0.8210	0.8130	0.8234	0.8147
Accuracy	0.9332	0.9399	0.9391	0.9390
FP	0.126	0.110	0.113	0.112
FN	0.140	0.124	0.127	0.126

Tabla 4.8: Resultados en la modificación del parámetro gamma, limite inferior

	Gamma: 75	**Gamma: 70**	**Gamma:65**	**Gamma: 60**
Indice Dice	0.8210	0.8321	0.8413	0.8367
Accuracy	0.9332	0.9234	0.9420	0.9445
FP	0.127	0.134	0.108	0.100
FN	0.140	0.154	0.121	0.112

Una vez que se tienen los resultados finales obtenidos mediante la metodología propuesta se realizó una sumatoria de las imágenes de salida con de entrada con el objetivo de tener una visualización de la predicción, estas imagenes se pueden observar en la Figura 4.7, los resultados en comparación con Ripple-GAN se pueden ver en la Tabla 4.9

Tabla 4.9: Resultados Finales CIONet

Modelo	**FN**	**FP**	**Precision**
LaneNet	**0.0244**	0.0780	0.964
RippleGan	0.0289	**0.0048**	**0.9728**
CIONet	0.112	0.100	0.9445

Figura 4.7: Predicciones sobre imágenes de entrada

4.4. Resultados en sistemas embebidos

Las pruebas realizadas con el modelo implementado en el sistema embebido se llevaron a cabo con la base de datos UAQuiLane aunque al no estar etiquetada no se pudo llevar a cabo alguna evaluación mediante las métricas establecidas, sin embargo, se puede notar que el modelo puede crear predicciones de lineas de carril con imagenes de carreteras fuera de la base de datos con la que fue entrenada, los resultados se pueden ver en las Figuras 4.8 y 4.9.

Se consideró realizar pruebas en carretera mediante el uso de una cámara web y un automóvil pero se tuvo la limitante del poder de computo en el sistema embebido ya que en la inferencia de la base datos UAQuiLane se obtuvieron aproximadamente 0.3 FPS, esto se mantuvo utilizando la cámara web como la fuente de las imágenes haciendo la detección en tiempo real inviable.

4.5. Trabajo Futuro

Como trabajo a futuro podría ser profundizada la red neuronal o dividirla en 2 secciones la primera que se encargue de la segmentación de carriles y la segunda en la separación de clases, esto seria con el objetivo de tener una salida completa al final sin la necesidad de realizar un post-procesamiento.

Otro punto que se puede considerar sería el uso de distintos pre-procesamientos los cuales puedan incrementar el rendimiento de la red, así como el uso de un sistema embebido con mayor capacidad de procesamiento para poder llevar a cabo la implementación en tiempo real.

CAPÍTULO 5

Conclusión

En esta trabajo se propusieron 2 modelos de redes neuronales, el primer modelo fue basado en la arquitectura GAN utilizando como base la red neuronal Pix2Pix, el segundo modelo se basó en la arquitectura U-net modificada. Para su entrenamiento se buscaron distintas bases de datos que hayan sido utilizadas para modelos del estado del arte, para esto se seleccionó Tusimple debido a la capacidad de procesamiento que se contaba para el proyecto.

A partir de esto se planteó una metodología para probar distintos pre-procesamientos que pudieran ayudar a disminuir los problemas que impactan la detección de carriles, estos fueron: cambios en el espacio de color, disminución de canales en la imagen y un algoritmo de agrupamiento. Al realizar las pruebas nos dimos cuenta que el cambio en los espacios de color no influía tanto en el rendimiento de la red neuronal. Sin embargo, la disminución de los canales impactó de una manera significativa los resultados aumentando la calificación del indice Dice, ese aumento se puede ver en todos los espacios de color si comparamos la tabla 4.5 con la tabla 4.6.

Una vez que se tuvo esta información el siguiente paso es evaluar con las métricas oficiales pero para esto se necesitó separar los carriles lo cual representó un obstáculo ya que no se contaba con la información necesaria directamente de la red neuronal ya que su salida es una imagen binarizada. Para abordar esta problemática se propuso multiplicar la imagen de entrada por la imagen de salida, con esto se puede obtener la predicción realizada a color sin alterar los resultados del red.

Finalmente se llega a la ultima parte de la optimización la cual consiste en ajustar los hiperparámetros de la función de pérdida, para este trabajo se selecciono el parámetro gamma la cual representa la curva de aprendizaje con la que se encontrará la red neuronal , para realizar la evaluación se utilizaron las métricas oficiales de la base de datos. El valor inicial de gamma es: 75, por lo que se propuso incrementar y disminuir su valor por un factor de 5 hasta llegar a un limite inferior de 60 y un valor superior de 90.

Analizando los resultados obtenidos podemos ver que el valor óptimo en este rango fue: 60, obteniendo un valor en las métricas oficiales de: 0.9445 y un valor en el indice Dice de: 0.8367. Un aspecto a notar en la experimentación es que el valor del indice Dice no muestra una correlación aparente con la métrica oficial.

Con la información obtenida de la experimentación podemos definir la combinación optima encontrada mediante la metodología propuesta, primero describimos el modelo neuronal utilizado.

- **Red Neuronal:** CIONet.
- **Función de pérdida:** Focal Tversky + L2.
- **Pre-procesamiento:** Espacio de color HSV de 1 canal.
- **Épocas:** 150
- **Tamaño de lote:** 32
- **Valor de Gamma:** 0.60

Por ultimo se llega a la implementación en un sistema embebido para la cual el modelo tuvo que ser transformado a la librería Tensorflow Lite la cual esta optimizada para estos sistemas, la base de datos utilizada para estas pruebas fue UAQuiLane la cual fue creada tomado videos de las carreteras del estado de Querétaro

Referencias

[1] México, séptimo lugar mundial en siniestros viales.

[2] Penmetsa, P.; Hudnall, M.; Nambisan, S. Potential safety benefits of lane departure prevention technology. *IATSS Research* **2019**, *43*, 21–26.

[3] Bengler, K.; Dietmayer, K.; Farber, B.; Maurer, M.; Stiller, C.; Winner, H. Three decades of driver assistance systems: Review and future perspectives. *IEEE Intelligent transportation systems magazine* **2014**, *6*, 6–22.

[4] Wang, Y.; Teoh, E. K.; Shen, D. Lane detection and tracking using B-Snake. *Image and Vision Computing* **2004**, *22*, 269–280.

[5] Kim, Z. Robust Lane Detection and Tracking in Challenging Scenarios. *IEEE Transactions on Intelligent Transportation Systems* **2008**, *9*, 16–26.

[6] Assidiq, A.; Khalifa, O. O.; Islam, M. R.; Khan, S. In *2008 International Conference on Computer and Communication Engineering*, IEEE: **2008**.

[7] Li, Z.-Q.; Ma, H.-M.; Liu, Z.-Y. In *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, IEEE: **2016**.

[8] Zhang, J.; Xu, Y.; Ni, B.; Duan, Z. In *Proceedings of the European Conference on Computer Vision (ECCV)*, **2018**, 486–502.

[9] Huang, Y.; Chen, S.; Chen, Y.; Jian, Z.; Zheng, N. In *Artificial Intelligence Applications and Innovations*, ed. by Iliadis, L.; Maglogiannis, I.; Plagianakos, V., Springer International Publishing: Cham, **2018**, 143–154.

[10] Kim, J.; Kim, J.; Jang, G.-J.; Lee, M. Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection. **2017**, *87*, 109–121.

[11] Hou, Y.; Ma, Z.; Liu, C.; Loy, C. C. Learning Lightweight Lane Detection CNNs by Self Attention Distillation. **2019**.

[12] Yang, W.-J.; Cheng, Y.-T.; Chung, P.-C. Improved Lane Detection With Multilevel Features in Branch Convolutional Neural Networks. *IEEE Access* **2019**, *7*, 173148–173156.

[13] Liu, B.; Liu, H.; Yuan, J. In *3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)*, **2019**.

[14] Choi, S.; Lee, K.; Kim, K.; Kwak, S. Lane departure warning system using deep learning. *Journal of the Korea Industrial Information Systems Research* **2019**, *24*, 25–31.

[15] Cao; Song; Song; Xiao; Peng Lane Detection Algorithm for Intelligent Vehicles in Complex Road Conditions and Dynamic Environments. *Sensors* **2019**, *19*, 3166.

[16] Muthalagu, R.; Bolimera, A.; Kalaichelvi, V. Lane detection technique based on perspective transformation and histogram analysis for self-driving cars. *Computers & Electrical Engineering* **2020**, *85*, 106653.

[17] Ghafoorian, M.; Nugteren, C.; Baka, N.; Booij, O.; Hofmann, M. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, **2018**.

[18] Narote, S. P.; Bhujbal, P. N.; Narote, A. S.; Dhane, D. M. A review of recent advances in lane detection and departure warning system. *Pattern Recognition* **2018**, *73*, 216–234.

[19] Szeliski, R., Computer vision: Algorithms and applications; Springer: **2022**.

[20] Huang, J.; Kong, B.; Li, B.; Zheng, F. In *2007 International Conference on Information Acquisition*, IEEE: **2007**.

[21] Karavaev, A.; Al-Naim, R. In *Fifth Conference on Software Engineering and Information Management (SEIM-2020)(full papers)*, **2020**, 66.

[22] Gonzalez, R. C.; Woods, R. E., Digital image processing; Prentice Hall: Upper Saddle River, N.J., **2008**.

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

[24] Goodfellow, I.; Bengio, Y.; Courville, A., Deep Learning; Adaptive computation and machine learning; MIT Press: **2016**.

[25] Jadon, S.; Leary, O. P.; Pan, I.; Harder, T. J.; Wright, D. W.; Merck, L. H.; Merck, D. L. In *Medical Imaging 2020: Imaging Informatics for Healthcare, Research, and Applications*, **2020**; *11318*, 113180Q.

[26] Yeung, M.; Rundo, L.; Nan, Y.; Sala, E.; Schönlieb, C.-B.; Yang, G. Calibrating the Dice loss to handle neural network overconfidence for biomedical image segmentation, **2021**.

[27] Rahnama, J.; Hüllermeier, E. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, ed. by Lesot, M.-J.; Vieira, S.; Reformat, M. Z.; Carvalho, J. P.; Wilbik, A.; Bouchon-Meunier, B.; Yager, R. R., Springer International Publishing: Cham, **2020**, 269–280.

[28] Tang, J.; Li, S.; Liu, P. A review of lane detection methods based on deep learning. *Pattern Recognition* **2020**, *111*, 107623.

[29] Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation, **2015**.

[30] Lamba, H. Understanding Semantic Segmentation with UNET, **2019**.

[31] Isola, P.; Zhu, J.-Y.; Zhou, T.; Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. **2016**.

[32] The Embedded Rust Book.

[33] White, E., Making Embedded Systems: Design Patterns for Great Software; Oreilly and Associate Series; O'Reilly Media, Incorporated: **2011**.

[34] Politiek, R. Deep learning with raspberry pi and alternatives in 2022 - Q-engineering, **2022**.

[35] Pizzati, F.; Allodi, M.; Barrera, A.; García, F. Lane Detection and Classification using Cascaded CNNs, **2019**.