INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

“MODELO DE RED PROFUNDA PARA SEGMENTACIÓN SEMÁNTICA EN TIEMPO REAL CON PLATAFORMA GPU EMBEBIDA”

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS EN INGENIERÍA ELECTRÓNICA

PRESENTA:

JESÚS ALONSO REYES PORRAS

DIRECTOR DE LA TESIS:

DR. JUAN ALBERTO RAMÍREZ QUINTANA

CODIRECTORA DE LA TESIS:

M.C. ALMA DELIA CORRAL SÁENZ

CHIHUAHUA, CHIH., Noviembre 2021

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MÉXICO

Chihuahua, Chih. 22 de noviembre de 2021

C. JESÚS ALONSO REYES PORRAS
PRESENTE

En cumplimiento con los requerimientos para la obtención del grado de Maestro en Ciencias en Ingeniería Electrónica y a propuesta de su Comité Tutorial, la División de Estudios de Posgrado e Investigación le concede la autorización para imprimir la tesis titulada ***"Modelo de red profunda para segmentación semántica en tiempo real con plataforma GPU embebida"*** dirigida por el Dr. Juan Alberto Ramírez Quintana con el siguiente contenido de capítulos:

I Antecedentes
II Análisis del funcionamiento de las redes convolucionales en segmentación
III Propuesta
IV Resultados y conclusiones

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"La Técnica por el Engrandecimiento de México"

ROGELIO ENRIQUE BARAY ARANA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Chihuahua, Chih. 9 de noviembre de 2021

ROGELIO ENRIQUE BARAY ARANA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
PRESENTE

En cumplimiento con los requerimientos para la obtención del grado de Maestro en Ciencias en Ingeniería Electrónica, le notificamos que el documento de tesis del alumno C. JESÚS ALONSO REYES PORRAS, titulado ***"Modelo de red profunda para segmentación semántica en tiempo real con plataforma GPU embebida"*** dirigido por el Dr. Juan Alberto Ramírez Quintana ha sido aprobado y aceptado para su impresión.

Por lo anterior, proponemos le sea concedida la autorización de impresión correspondiente.

Agradeciendo la atención a la presente, quedamos de usted:

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"La Técnica por el Engrandecimiento de México"

MTRA. ALMA DELIA CORRAL SÁENZ
MIEMBRO DEL COMITÉ TUTORIAL

DR. MARIO IGNACIO CHACÓN MURGUÍA
MIEMBRO DEL COMITÉ TUTORIAL

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MTRO. ALFREDO CHACÓN ALDAMA
MIEMBRO DEL COMITÉ TUTORIAL

Graciela Ramírez A

DRA. GRACIELA MARÍA DE JESÚS RAMÍREZ ALONSO
MIEMBRO DEL COMITÉ TUTORIAL

CARTA CESIÓN DE DERECHOS

En la ciudad de Chihuahua el día 22 de noviembre de 2021, el que suscribe, C. JESÚS ALONSO REYES PORRAS con número de control G19060582, de la Maestría en Ciencias en Ingeniería Electrónica, adscrita a la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del el Dr. Juan Alberto Ramírez Quintana y cede los derechos del trabajo titulado "Modelo de red profunda para segmentación semántica en tiempo real con plataforma GPU embebida", al Tecnológico Nacional de México y/o Instituto Tecnológico de Chihuahua para su difusión, divulgación, transmisión, reproducción, así como su digitalización con fines académicos y de investigación.

Alonso Reyes P.

C. JESÚS ALONSO REYES PORRAS

DECLARACIÓN DE ORIGINALIDAD

En la ciudad de Chihuahua el día 22 de noviembre de 2021, el que suscribe, C. JESÚS ALONSO REYES PORRAS de la Maestría en Ciencias en Ingeniería Electrónica, con número de control G19060582, adscrito a la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua, manifiesta que es autor intelectual de la tesis titulada "Modelo de red profunda para segmentación semántica en tiempo real con plataforma GPU embebida" bajo la dirección el Dr. Juan Alberto Ramírez Quintana; que el contenido es original y que las fuentes de información consultadas para su fundamentación están debidamente citadas y referenciadas.

Alonso Reyes P.

C. JESÚS ALONSO REYES PORRAS

Chihuahua, Chihuahua, a 23 de noviembre de 2021

Dra. María Elena Álvarez-Buylla Roces
Directora del CONACYT

Por este medio aprovecho la ocasión para saludarla e informarle que a la fecha he obtenido el grado de **Maestría en Ciencias de Ingeniería Electrónica** en la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua. Agradezco todo el apoyo brindado por CONACYT ya que el otorgamiento de la beca para estudios de posgrado permitió dedicarme de tiempo completo al programa y con ello lograr el cumplimiento del objetivo principal del convenio establecido.

Sin otro particular por el momento, me es grato quedar de usted como su seguro servidor, no sin antes reiterar nuevamente todo mi agradecimiento.

Atentamente

Alonso Reyes P.

Jesús Alonso Reyes Porras
Exbecario CONACYT

AGRADECIMIENTOS

A mi familia

En especial a mis padres, hermanos y sobrinos porque con su apoyo, amor y comprensión no hubiera podido culminar esta etapa en mi carrera.

A mis amigos y compañeros

Agradezco a mis amigos y compañeros de generación, Javier e Iván, por su apoyo y el gran equipo que hacíamos a lo largo de las materias. Agradezco a los demás amigos que hice en el laboratorio Oscar, Gerardo, Alejandro, Adrián, David, Raúl, Andrea, Daniel, Eduardo, José, Abimael, Luis, Xavier, Vicente, Miguel, Eliacim y Wendy por compartir su conocimiento conmigo, además de hacer más agradable y llevadera esta etapa.

A mi comité de tesis

Agradezco a mi asesor el Dr. Juan Alberto Ramírez Quintana por sus consejos, paciencia y empeño que puso a lo largo de la realización de este trabajo. Agradezco al Dr. Mario Chacón por sus valiosos aportes en cada reunión de comité tutorial. Agradezco a la M.C. Alma Corral por sus excelentes clases de procesamiento de señales, por sus observaciones en cada comité tutorial y por su excelente trabajo como coordinadora. Agradezco a la Dra. Graciela Ramírez por sus comentarios y aportes a este trabajo.

Al Tecnológico Nacional de México y CONACYT

Agradezco al Consejo Nacional de Ciencia y Tecnología por su apoyo durante mi estancia en la maestría. Al Tecnológico Nacional de México por el apoyo a esta tesis bajo el proyecto “Modelo de red neural profunda de procesamiento inteligente de señales para sistemas de conectividad informática de computación en el borde”, clave 7598.20-P y "Clasificación de parásitos en muestras de copro mediante aprendizaje de máquina” con clave 10071.21-P.

Resumen

MODELO DE RED PROFUNDA PARA SEGMENTACIÓN SEMÁNTICA EN TIEMPO REAL CON PLATAFORMA GPU EMBEBIDA

Ing. Jesús Alonso Reyes Porras

Maestría en Ciencias en Ingeniería Electrónica

División de Estudios de Posgrado e

Investigación del Instituto Tecnológico de

Chihuahua, Chihuahua, 2021

Director de Tesis: Dr. Juan Alberto Ramírez Quintana

Codirector de Tesis: M.C. Alma Delia Corral Sáenz

El aprendizaje profundo es un área de la inteligencia artificial que ha crecido de manera significativa en los últimos años gracias al desarrollo de tecnologías embebidas que permiten ejecutar algoritmos de forma más eficiente y con menor tiempo de procesamiento. Esta área ha sido muy útil, ya que se pueden extraer características abstractas de manera automática, lo que causa que el aprendizaje profundo sea propicio para la segmentación semántica de objetos. Entre estos modelos y formas de aprendizaje, las redes totalmente convolucionales (FCN por sus siglas en inglés) son las que han tenido mayor éxito en segmentación semántica debido a la alta precisión que tienen. Sin embargo, para lograr estos resultados, estas redes suelen ser muy profundas, por lo que el costo computacional aumenta. De esta manera, han surgido recientemente diversos trabajos en la literatura para mejorar la eficiencia computacional de las FCN. Por lo tanto, contribuyendo con estos esfuerzos, en esta tesis se propone un modelo de FCN para segmentación semántica que corre en tiempo real en una plataforma de Unidad Grafica de Procesamiento embebida.

Este modelo se denominó Red de tiempo real Consciente del Contexto para sistemas embebidos, CARTNet por sus siglas en inglés (*Context Aware and Real Time Network for Embedded Systems)* y se enfoca principalmente en la relación que existe entre el contexto global y local en un escenario. Esto lo logra mediante dos enfoques: uno es una red de dos ramas de resolución, en la que una rama se encarga de la alta resolución para obtener el contexto global

y la otra rama con la baja resolución se encarga del contexto local y los detalles finos. El otro enfoque es mediante convoluciones multi-dilatadas, que combinan convoluciones de diferentes grados de dilatación para lograr tener diferentes campos receptivos para analizar diferentes niveles de contexto, similar a cuando una persona se aleja de un objetivo para poder ampliar su campo de visión.

De acuerdo con los resultados, este modelo presenta una velocidad de 26.12 FPS en la tarjeta GPU embebida Nvidia Jetson TX2, una precisión de 52.12% en la base de datos de Cityscapes, 221,827 parámetros y 1.25 GFlops (GIGA operaciones de punto flotante) mientras que otras redes eficientes como ContextNet logra 22.03 FPS, 66.1% de precisión, 876,560 parámetros y 1.78 GFlops.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1
ANTECEDENTES

En este capítulo se presentan los conceptos y trabajos necesarios para fundamentar esta tesis. Por la naturaleza de este trabajo, es necesario abordar antecedentes sobre el procesamiento de imágenes, sistemas embebidos y aprendizaje automático. Por ello, este capítulo se organiza de la siguiente manera: la sección 1.1 presenta los conceptos básicos de procesamiento de imágenes y la sección 1.2 describe los sistemas embebidos. Finalmente, las secciones 1.3 y 1.4 presentan las ideas elementales sobre aprendizaje automático y profundo aplicados a segmentación semántica.

1.1 Procesamiento de imágenes

Una imagen se define como una función de dos dimensiones $f(x,y)$ donde x e y son coordenadas espaciales y f es la intensidad o nivel de gris. Si las coordenadas (x, y) y el nivel de intensidad son finitos y discretas, las imágenes se les conocen como digitales.

Una de las primeras apariciones de las imágenes digitales fue a principios de los años 20, cuando un periódico logró enviar imágenes por cables submarinos entre Londres y Nueva York, por medio del sistema Bartlane, que fue una adaptación del telégrafo para enviar imágenes codificadas. Gracias a la introducción de estos cables, se redujo el tiempo de trasporte de información de más de una semana, a tan solo tres horas. Este sistema era capaz de codificar en cinco distintos niveles de gris [1].

Por otro lado, el procesamiento digital de imágenes se refiere al tratamiento de imágenes por medios electrónicos, y sus aplicaciones atañen diversos campos de estudio como la medicina, la manufactura, sistemas de seguridad, etcétera. Comúnmente, se asocia con visión por computadora ya que muchas de las técnicas para procesar imágenes se basan en cómo funciona el sistema de visión humano [2].

En lo referente al procesamiento de imágenes y la visión por computadora, existen tres niveles:

- Bajo nivel: se refiere al preprocesamiento de imágenes, reducción de ruido, mejora de contraste y nitidez.

- Medio nivel: involucra tareas como la segmentación y la detección, que consisten en dividir la imagen en regiones, o definir límites y superficies que tengan relación con los objetos presentes en la imagen. Generalmente, la salida del medio nivel son características como contornos, bordes e identificadores de cada objeto.
- Alto nivel: hace referencia al procesamiento que toma las decisiones o se encarga de interpretar los atributos de las imágenes [1].

En la Figura 1.1 se ilustran las etapas de un sistema para procesamiento de imágenes y visión:

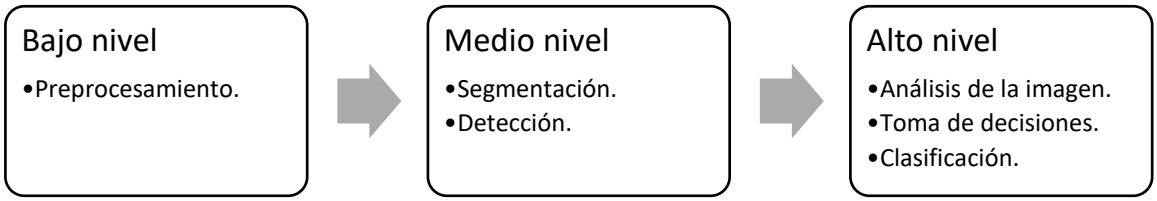

Figura 1.1 Etapas de un sistema de visión.

Debido a que esta tesis se trabaja con segmentación, se hará énfasis en el nivel intermedio, por lo que a continuación se describirán los conceptos relacionados a segmentación.

1.1.1 Segmentación

La segmentación subdivide una imagen en regiones que representan a los objetos del escenario, y el nivel de las subdivisiones depende de la aplicación. Los métodos de segmentación tienen dos vertientes principales:

- Basada en la determinación de bordes de objetos y discontinuidades, para delimitar los objetos de interés.
- Basado en similitud de la información para realizar agrupamientos de pixeles que representen los objetos.

La importancia de la segmentación reside en que sirve como puente entre el medio y el alto nivel de procesamiento de imágenes, ya que permite tener una mejor interpretación de la escena.

1.1.2 Detección de bordes

La detección de bordes se basa en encontrar las discontinuidades derivadas de los cambios en la intensidad de tonos de gris en una imagen. Para que esta forma de segmentación funcione, los objetos deberán tener tonos de grises uniformes en varias zonas de la imagen, pero en otras áreas deben ser diferentes entre sí.

Dos ejemplos de operador que se usa para detectar dicho comportamiento son la primera y segunda derivada, ya que miden la variación de la intensidad en la función $I(x, y)$, que representa a la imagen [2].

1.1.3 Segmentación por umbral

Este método está basado en uniformidad y busca dividir la imagen por medio de uno o varios umbrales con respecto a tonos de gris o color que permiten generar agrupamientos de pixeles que definen los objetos presentes en la imagen. La segmentación por umbral está definida por la ecuación 1.1 [2].

$$\mathrm{I_N(x,y)} = \begin{cases} 0, & 1_\mathrm{N}(\mathrm{x,y}) < \mathrm{T} \\ 1, & 1_\mathrm{N}(\mathrm{x,y}) \geq \mathrm{T} \end{cases} \tag{1.1}$$

donde $I_N(\mathrm{x,y})$ es el píxel en la posición (x, y) y T es el umbral seleccionado.

1.1.3.1 Segmentación semántica

La meta de la segmentación semántica consiste en asignar una etiqueta a cada píxel, para indicar a que objeto conocido pertenece cada píxel. Dichas etiquetas se asignan dependiendo que la clase o significado que tiene cada píxel. Algunos ejemplos de segmentación semántica se pueden apreciar en la Figura 1.2, donde se puede observar que a cada elemento del escenario se le otorga una etiqueta de color. Por ejemplo, donde se encuentra una vaca se le otorga la clase de vaca a cada píxel que conforma esta clase [3].

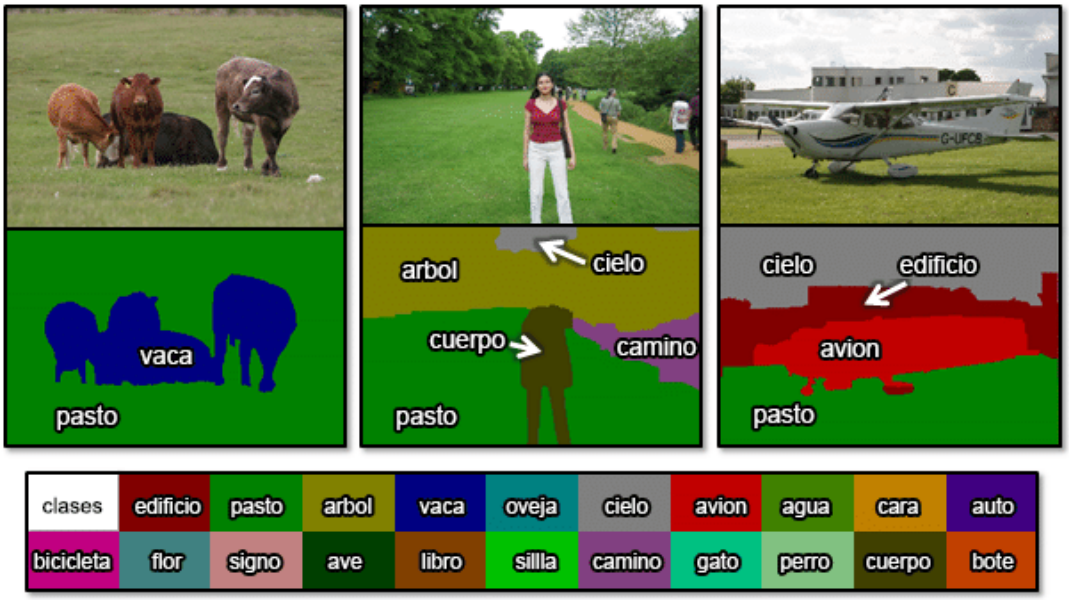

Figura 1.2 Ejemplo de segmentación semántica con las etiquetas de cada clase.

Este tipo de segmentación ha tenido diversas aplicaciones tales como:

- Manejo autónomo, donde un vehículo puede detectar una persona o saber el camino que tiene que llevar.
- En la detección temprana de la retinopatía diabética, glaucoma e hipertensión. En este caso, la detección vasos sanguíneos retinianos presentes en los ojos se puede realizar con la segmentación semántica [4].

1.2 Sistemas embebidos

Un sistema embebido se basa en un microprocesador y se desarrolla para una función específica. Son portables y escalables, ya que se le pueden agregar componentes de hardware sin necesidad de modificar el software [5].

1.2.1 Partes de un sistema embebido

Los sistemas embebidos se componen de lo siguiente:

Procesador: es el núcleo principal del sistema y se encarga de realizar operaciones, ejecutar programas o instrucciones en lenguaje de bajo nivel como aritméticas, lógicas y de acceso a memoria. Debido a que existe una gran cantidad de aplicaciones embebidas, también existe una

gran variedad de procesadores que cumplan con las características necesarias para realizar diferentes tareas y aplicaciones [6].

Memoria: almacena datos durante algún periodo de tiempo y tiene dos funciones:

Memoria de programa: proporciona almacenamiento para el software que se va a ejecutar y puede ser de solo lectura (ROM) o externa (EPROM). Esta memoria almacena programas de ejecución del encender y del sistema.

Memoria de datos: otorga almacenamiento para datos tales como variables de programa, resultados intermedios, información de estado y cualquier otro dato que pudiera ser creado a través de las operaciones. El software necesita memoria para almacenar variables y para manejar *stacks*.

Periféricos: un sistema embebido necesita comunicarse con el mundo exterior y lo hace a través de los periféricos de entrada y salida. Los periféricos de entrada se asocian con sensores que miden variables del ambiente exterior para controlar la manera en que actúan las salidas. Por otra parte, los periféricos de salida se conectan a los actuadores, que son los dispositivos con los que el sistema interactúa con el entorno. Ejemplos de actuadores son motores, luces, pantallas, etcétera. Además de los periféricos, los sistemas embebidos cuentan con terminales digitales de propósito general (GPIO), que son pines que funcionan como entradas o salidas binarias.

Software: son programas computacionales que organizan y administran un sistema [7]. El software en un sistema embebido consta de diferentes componentes:

- Inicialización y configuración de los recursos.
- Sistema operativo y entorno de ejecución.
- Aplicación del software.
- Librerías para el manejo de errores.
- Soporte de mantenimiento y corrección de errores [5].

1.3 Aprendizaje de máquina

El aprendizaje automático es una forma de inteligencia artificial que aprende a partir de los datos en lugar de una programación explícita. El esquema general de un algoritmo de aprendizaje de máquina consta de las siguientes etapas: preprocesamiento de los datos, extracción de características y clasificación. A continuación, se detallan cada una de estas etapas.

1.3.1 Extracción de características

Esta etapa se refiere a la reducción de dimensiones de un conjunto de datos mediante el análisis de las propiedades de la información de cada muestra para poder categorizarla a cada conjunto o clase. Es preciso seleccionar adecuadamente las características debido a que impactan directamente en el desempeño del sistema. Algunas características más usadas en la literatura de segmentación son:

Color del píxel: las imágenes se pueden representar en distintos espacios de colores como Rojo-Verde-Azul (RGB), Matiz-Saturación-Valor (HSV), Matiz-Saturación-Iluminación (HSI) y escalas de grises. Típicamente, las imágenes se almacenan en el espacio de color RGB, pero en ocasiones, se recurre a otro espacio de color. Los espacios de color más utilizados son HSI y RGB, debido a que RGB es el espacio de color en el cual se adquieren las imágenes, mientras que HSI es invariante a la iluminación y representa el color de una manera similar a como lo hace a la visión humana. Comúnmente, se suelen buscar espacios de colores distintos al RGB ya que en algunas aplicaciones es necesario separar iluminación del color. Por ejemplo, Ju et al. [8] presenta un método de segmentación basado el espacio de colores HSI ya que como mencionan en el artículo, es el espacio de colores que representa de manera más fiel la visión humana. Este espacio de color se combina con un algoritmo de variación de media (*Mean-Shift*) como método de agrupación, que es un método no paramétrico y se lleva a cabo en cuatro pasos:

1.- Se escoge una ventana de búsqueda para tener un punto de partida de este algoritmo.

2.- Calcular el centro de masa de esa ventana.

3.- Centrar la ventana en el centro de masa.

4.- Repetir el paso 2 hasta convergencia.

Se realizaron algunos experimentos comparando este algoritmo con k-medias (*k-means*) obteniendo los resultados mostrados en la Figura 1.3 y 1.4. Donde se puede apreciar una segmentación con gran detalle entre las áreas verdes, la tierra y el cielo.

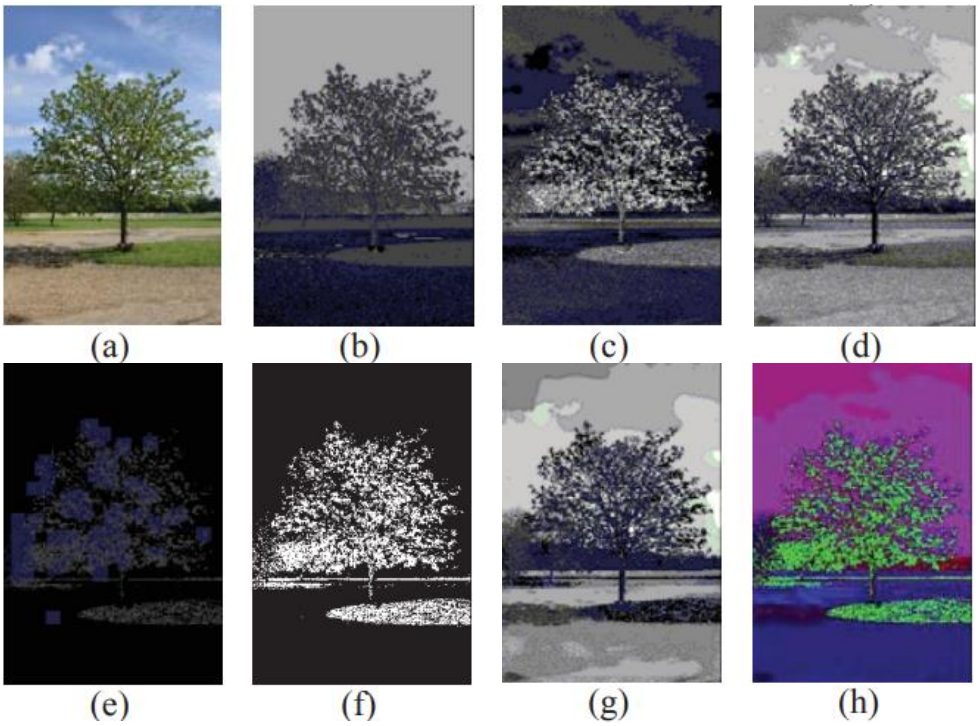

Figura 1.3 Resultado del algoritmo propuesto por ju et al. [8] donde (a) es la imagen de entrada, (b) es el matiz, (c) saturación, (d) intensidad, (e) es el matiz aplicando el algoritmo de variación de mediana, (f) saturación con variación de mediana, (g) intensidad con variación de mediana y (g) es el resultado de la segmentación.

Figura 1.4 Resultado del algoritmo de comparación donde (a) es la imagen de entrada (b) son los objetos en el clúster 1, (c) objetos en el clúster 2, (d) objetos en el clúster 3 y (e) es el resultado de la segmentación.

Histograma de gradientes orientados (HOG): las características de histograma de gradientes orientados interpretan una imagen como una función discreta. Por cada píxel hay dos gradientes: la derivada parcial de *x* e *y*. HOG toma la imagen original para transformarla en dos mapas de características de igual tamaño representados por el gradiente. Este mapa de características se divide en celdas de *NxN* pixeles y se calcula un histograma de direcciones para cada celda de *NxN* pixeles. Por ejemplo, este mapa de características es usado por Wentao et al. [9] en la detección semántica de vehículos en plataformas en movimiento, donde el HOG se usó como método de extracción de características para ser introducido en un clasificador basado en máquina de vectores de soporte. Se entrenó y probó con videos de aproximadamente 15 minutos de duración con 24 cuadros por segundo (de estos cuadros, 2500 se destinaron al entrenamiento), cada cuadro es de 1024x768 píxeles, obteniendo una precisión de 88.7%.

SIFT: la transformada de características invariantes a las escalas (SIFT) es una combinación entre el detector de puntos claves y la representación de histogramas basados en gradientes orientados. Estas características son invariantes a la escala, la rotación, cambios de iluminación, ruido y pequeños cambios de pose o perspectiva. Debido a estas características, Gao et al. [10] utilizaron SIFT en segmentación semántica de imágenes con información de profundidad en el espacio de colores RGBD. Los experimentos fueron realizados en la base de datos de la Universidad de Nueva York Depth Dataset Version 2 (*NYUD2*) y obtuvieron un 57.97% de precisión para cuatro clases. Pero cuando se segmenta para 40 clases las clases, la precisión disminuye drásticamente a 18.87%.

Reducción dimensional: cuando las imágenes son de alta resolución, tienen gran cantidad de píxeles y millones de características que causan un entrenamiento más tardado. Sin embargo, no significa que contenga más información relevante. Consecuentemente, en ocasiones se reducen las dimensiones de la imagen [11] [12] mediante submuestreo.

Por otra parte, existen otros tipos de reducción dimensional que no se basan en submuestreo como el caso del análisis de componentes principales (PCA). Un ejemplo, es el desarrollado por Tai et al. donde utilizaron en [13] el análisis PCA para segmentación semántica en el procesamiento de resonancia magnética funcional. Se generó una base de datos con seis

voluntarios, de la que se determinaron cinco distintas clases (ignorando el fondo): huesos, fluido espinal, disco lumbar, vértebras y fluido cerebroespinal. Se consiguió una precisión de intersecciones sobre la unión promedio (mIOU) de 60.7%.

1.3.2 Segmentación no supervisada

Los algoritmos de segmentación no supervisada suelen aprender de los datos sin tener un conocimiento previo de ellos y se pueden usar como medio para obtener información o para refinar la segmentación. Por ejemplo, Feng et al. [14] utilizaron la factorización matricial ortogonal no negativa para extraer características de diferentes objetivos. Se probó con la base de datos de MNIST que está orientada al reconocimiento de números escritos a mano, logrando separar el fondo de los números escritos a mano. Ju et al. [8] utiliza este mismo método de segmentación no supervisada, al utilizar un algoritmo de variación de media presentado con anterioridad y se obtuvieron como resultados vistos en las figuras 1.3 y 1.4.

Otro algoritmo no supervisado es la segmentación basada en grafos (GBS). Estos algoritmos interpretan los píxeles como vértices y los bordes como pesos, donde los pesos son una medición de disimilitud entre los dos pixeles conectados por ese borde. Por ejemplo, Kim et al. [15] basaron su modelo de GBS para realizar un modelo jerárquico que segmenta regiones para llegar al objeto de interés, logrando así la segmentación de vehículos y peatones. Se entrenó este modelo con la base de datos PETS y se evaluó con la base de datos i-LIDS. Concluyeron que el modelo es capaz de detectar eficientemente objetos de varios tamaños y formas, además de objetos parcialmente ocluidos.

1.3.3 Algoritmos supervisados para segmentación

Los algoritmos de aprendizaje supervisado consisten en aprender del ejemplo, es decir, aprender a mapear una entrada con base en su salida. Para ello, es necesario que el conjunto de datos de entrenamiento tenga muestras con salidas conocidas. Por ello, para el desarrollo de algoritmos supervisados para segmentación semántica se requiere conocer que objeto representa cada región. Un algoritmo que destaca es el *Random Decision Forest* y se trata de un conjunto de árboles de decisión donde cada nodo interno usa una o más características para decidir en cuál rama descender, y cada hoja es una clase. Una desventaja de este clasificador es

que el añadir más árboles no hace que el algoritmo generalice, causando un sobreajuste. Por ejemplo, Liu et al. [16] utilizaron este método en dos capas de etiquetado semántico de imágenes, una primera capa etiqueta los súper-píxeles a cada clase, y la segunda capa genera los límites de cada clase etiquetada. Se evaluó el método con la base de datos CamVid, obteniendo un promedio de exactitud de clase de 36.1% y un promedio global de 76.2%.

1.4 Aprendizaje profundo aplicado a segmentación semántica

El aprendizaje profundo es una derivación del aprendizaje máquina, donde los modelos analizan las propiedades de la entrada y clasifican a través de redes con múltiples capas (el término "profundo" se refiere a muchas capas). Como consecuencia, la etapa de extracción de característica se lleva a cabo en el mismo modelo de manera abstracta. Las redes de aprendizaje profundo que se utilizan comúnmente en segmentación son las redes completamente convolucionales o auto codificadas ya que permiten realizar una clasificación por píxel.

1.4.1 Redes neurales completamente convolucionales

Long et al. [17], propusieron el modelo de Red Completamente Convolucional (FCN por sus siglas en inglés), y constan de dos secciones principales: el codificador y el decodificador. El codificador realiza la etapa de extracción de características, tal como una red convolucional de clasificación común. El decodificador tiene un conjunto de convoluciones traspuestas o dilatadas para escalar los mapas de características de modo que se tenga como salida una imagen con las regiones de segmentación que representan a los objetos. Este modelo se propone en [17] , donde logró resultados aceptables en bases de datos populares en el estado del arte, ya que se obtuvo un 62.2% de precisión en PASCAL VOC 2012, 34% en NYUDv2 y 39.5% en SIFT Flow que, si bien no parecen resultados muy altos, fue muy destacado al principio y no dista mucho de redes más modernas como la de Lin et al. [18] cuyos resultados en PASCAL VOC 2012 son de 78%, en NYUDv2 son de 40.6% y en SIFT Flow presenta resultados de 44.9%.. Por otro lado, Badrinarayanan et al. [19] proponen SegNet, una arquitectura codificador-decodificador, que también es una FCN, donde el decodificador sobre muestrea el mapa de características, pero manteniendo los índices de agrupación de las capas de codificación. Se probó en la base de datos CamVid y se logró un porcentaje de intersecciones sobre la unión de 60.1%.

Con las FCNs se puede tener dos maneras de generar una comprensión de la escena a segmentar, que son: contexto espacial y contexto temporal, lo que depende de la información dentro de la base de datos a utilizar. Cuando se refiere a contexto espacial, se trata únicamente de los objetos presentes en una misma imagen y la relación que tienen entre ellos mismos. El contexto temporal requiere de una secuencia de imágenes y la relación que existe entre los objetos de imágenes anteriores y posteriores.

1.4.2 Modelos conscientes del contexto espacial

El contexto espacial es idóneo para aquellas bases de datos que presentan imágenes estáticas. Para analizar este contexto, el campo receptivo en una capa convolucional es muy importante ya que, de manera similar a la visión humana, a mayor campo receptivo más contexto de un escenario se puede distinguir. El problema de aumentar el campo receptivo es que se pierde definición en objetos pequeños y aumenta la cantidad de parámetros necesarios para la convolución. Para contrarrestar este problema, Yu et al. [20] introdujeron las convoluciones dilatadas, que expanden el campo receptivo sin perder resolución espacial. Se experimentó en la base de datos PASCAL VOC de 2012, y se obtuvo una precisión de 67.6%. Por otro lado, Wu et al. [21], propusieron una red menos profunda usando conexiones residuales y que incluían convoluciones dilatadas. Esta red se implementó en una GPU modelo NVIDIA GTX 980 y se probó para dos tareas: detección de objetos y segmentación semántica. Para segmentación semántica, se probó en las bases de datos de Cityscapes, PASCAL VOC y ADE20K. La precisión obtenida fue de 80.84% para PASCAL VOC, 43.73% para ADE20K y 77.86% para Cityscapes.

Lin et al. [18], propusieron un método que integra campos aleatorios condicionales (CRF) y redes neurales convolucionales (CNN) para segmentación semántica, que consistió en generar conjuntos de pixeles contiguos gracias a los a campos condicionales y verificar la correlación que tienen con los conjuntos de pixeles vecinos mediante CNNs. Los experimentos se llevaron a cabo en las bases de datos de PASCAL VOC 2012, PASCAL-Context, NYUDv2 y SIFT-flow. Se logró como resultado una precisión de 75.3% en PASCAL VOC 2012. También, se experimentó aumentando el conjunto de entrenamiento con imágenes de la base de datos de Microsoft COCO. Los resultados de este aumento del conjunto de imágenes en diferentes bases

de datos se pueden ver en la Tabla 1.1, donde se observa un incremento del 2.7% en la base de datos de PASCAL VOC 2012.

Tabla 1.1 Resultados de la agregación de las imágenes de Microsoft COCO

Base de datos	Precisión
PASCAL VOC 2012	78%
NYUDv2	40.60%
PASCAL Context	43.30%
SIFT-Flow	44.90%

1.4.3 Modelos con información temporal

Recientemente, ha surgido la inquietud por la segmentación semántica para secuencias de video que brindan información de contexto temporal [22], [23]. Shelhamer et al. [22] introdujeron las Redes Neurales basadas en Relojes, que son señales de reloj que controlan el aprendizaje de las diferentes capas con diferentes factores de tiempo. Este modelo se probó con las bases de NYUD, Youtube-Objects y Cityscapes, y se lograron precisiones de 31.1%, 70% y 65.9% respectivamente. Tran et al. [24] propusieron una red convolucional en tres dimensiones entrenada para segmentación semántica en secuencias de video. Se probó el modelo con la base de datos de GATECH y se obtuvo una exactitud de 76%. Un inconveniente con esta convolución en tres dimensiones es que podría no capturar información a largo plazo debido a su corta extensión en el eje temporal. Sin embargo, las redes recurrentes pueden solventar dicho problema. Por ejemplo, Fayyaz et al. [23] incorporaron características espacio-temporales usando modelos de memoria a largo plazo (LSTMs por sus siglas en inglés). Los experimentos fueron llevados a cabo con las bases de datos CamVid, NYUDv2 y PASCAL VOC. De esta última, solo se tomaron los pesos preentrenados, y se obtuvo una precisión de 30.9% en la base de NYUDv2 y de 50.6% para CamVid.

1.4.4 Modelos enfocados a la eficiencia computacional

Existe un esfuerzo en la literatura realizado por varios autores para proponer FCNs con la menor cantidad de recursos posibles. Esto quiere decir que los modelos sean eficientes computacionalmente, lo que significa que la cantidad de recursos que utiliza el algoritmo o modelo a utilizar es la menor posible para lograr su objetivo. A continuación, se describen algunos trabajos donde se proponen modelos eficientes para FCNs.

Romera et al. [25], proponen el uso de conexiones residuales y convoluciones factorizadas para lograr una alta eficiencia computacional. Dichas conexiones residuales evitan el problema de degradación que se presenta en las FCNs con una gran cantidad de capas apiladas. Las convoluciones factorizadas son implementadas en una arquitectura de cuello de botella para reducir el número de parámetros y una mejor regularización, pero sin tener un impacto significativo en su desempeño de aprendizaje. Dicho modelo se entrenó y probó con la base de datos de Cityscapes, con la unidad de procesamiento gráfico (GPU) embebida Jetson TX1 y se obtuvo un resultado de 7 fps y un 68% de precisión (mIOU).

Li et al. [26] presentan un modelo donde se obtienen dos principales contribuciones: la primera es un módulo de propagación que fusiona características de manera adaptativa con el tiempo a través de una convolución variante en el espacio, para reducir el costo computacional por cuadro. La segunda contribución es un planificador adaptivo que asigna dinámicamente la cantidad de operaciones en función de la predicción de precisión. Los experimentos de este algoritmo fueron realizados en las bases de datos de Cityscapes y CamVid, y se obtuvo un 82.9% de exactitud y un 75.26% de precisión (mIOU) de clase en el caso de CamVid. Además, se reporta en este mismo trabajo una mejora en el tiempo de inferencia de 119 milisegundos respecto al modelo en el que se basó (ResNet) que obtuvo un tiempo de inferencia de 360 milisegundos.

Wu et al. [27] siguieron dos principales estrategias, la primera fue disminuir los mapas de características con mayor número de parámetros en las capas de convolución en el primer y último bloque convolucional. La segunda estrategia fue el uso de una estructura de red asimétrica con un codificador profundo y un decodificador relativamente superficial. Los

experimentos se realizaron con las bases de datos CamVid y Cityscapes, y se implementaron con diferentes GPUs para conocer la diferencia en tiempo de procesamiento entre una GPU de equipo portátil a un equipo de escritorio común. Se tuvieron resultados de 40 milisegundos de tiempo de procesamiento para CamVid y 39 milisegundos para Cityscapes. Respecto a la precisión, se obtuvo un 48% en CamVid y 44.5% en Cityscapes.

1.4.5 Bases de datos

Para el desarrollo de aplicaciones de aprendizaje profundo es necesario una gran cantidad de datos. Por ello, gran parte de la investigación en este ámbito ha sido enfocada a la generación de bases de datos que representen información útil para desarrollar dichas aplicaciones. De esta manera, esta subsección presenta aquellos datasets que se utilizan en el rubro de la segmentación semántica.

1.4.5.1 *Cityscapes*

Esta es la base de datos más utilizada en la literatura para segmentación semántica y panóptica. Las imágenes dentro de esta base de datos fueron obtenidas de 50 ciudades en Alemania con diferentes estaciones del año, de día y con condiciones climáticas variadas. El volumen de esta base de datos es de 5,000 imágenes divididas en 30 clases, y tienen 20,000 anotaciones con escenarios urbanos tomados desde la perspectiva delantera de un automóvil [28].

1.4.5.2 *CamVid*

Es una base de datos desarrollada por la Universidad de Cambrigde con información de 32 clases semánticas. Cuenta con videos capturados desde la perspectiva del automóvil, con más de 16 minutos de grabación a alta calidad a 30 Hz. Tiene 701 imágenes etiquetadas y un total de 5 videos de 3030, 6120, 202, 5130 y 3720 cuadros o imágenes en total [29].

1.4.5.3 *PASCAL VOC (Visual objects classes)*

Este dataset fue publicado por PASCAL (del inglés *Pattern Analysis, Statistical Modelling and Computational Learning*), que es una asociación financiada por el programa de tecnologías de la información para la sociedad de la unión europea. Esta asociación publicó la base de datos

como parte de una competición anual de reconocimiento de objetos. Cuenta con 500,000 imágenes, 20 clases y se divide en 50% de las imágenes para entrenamiento y validación y 50% para imágenes para prueba. Las imágenes son obtenidas desde una página web usada para compartir fotografías llamada Flickr [30].

1.4.5.4 *Microsoft COCO (Common objects in context)*

Publicaba por Microsoft, cuenta con 91 diferentes tipos de objetos y un total de 2.5 millones de instancias etiquetadas en 328 mil imágenes. Esta base de datos se enfoca en la segmentación de instancias individuales [31].

CAPÍTULO 2 ANÁLISIS DEL FUNCIONAMIENTO DE LAS REDES CONVOLUCIONES EN SEGMENTACIÓN

La forma más común de realizar la segmentación semántica es mediante FCNs, ya que presentan una salida del tamaño de su entrada y generan una clasificación a nivel de píxel. Estas redes constan de dos etapas: el encoder que extrae las características y el decoder que genera la segmentación de la imagen. El problema con estas redes es que tienen una alta complejidad computacional debido a que suelen tener millones de parámetros y operaciones de punto flotante. Por ello, se han presentado diversos trabajos en la literatura que tratan de proponer FCNs más eficientes que se puedan correr en plataformas embebidas como la Nvidia Jetson Tx2. Estas FCNs son de interés en esta tesis ya que el objetivo es desarrollar un modelo tan eficiente que sea capaz de correr en tiempo real en dicho sistema embebido. Por ello, en esta sección se presenta la metodología de análisis para proponer redes eficientes de bajo costo computacional para la segmentación semántica.

Para presentar esta metodología, la sección 2.1 se presenta las FCN y su arquitectura. En la sección 2.2 se presentan las técnicas más populares en la literatura para desarrollar redes eficientes. La sección 2.3 presenta un instrumento de análisis por métricas que ayudó a encontrar los modelos más adecuado para desarrollar la red propuesta en esta tesis.

2.1 Redes completamente convolucionales.

Las FCNs para segmentación semántica se componen por tres bloques principales como se puede apreciar en la Figura 2.1 [17]. El primer bloque es la imagen de entrada $I(x,y)^{RGB}$, donde (x,y) es la posición del píxel, $x \in R^x$, $y \in R^Y$. El segundo bloque llamado encoder (codificador) genera los mapas de características de los objetos presentes en la imagen. Finalmente, el tercer bloque llamado decoder (decodificador) forma los mapas de segmentación. Estos bloques se detallarán en secciones posteriores.

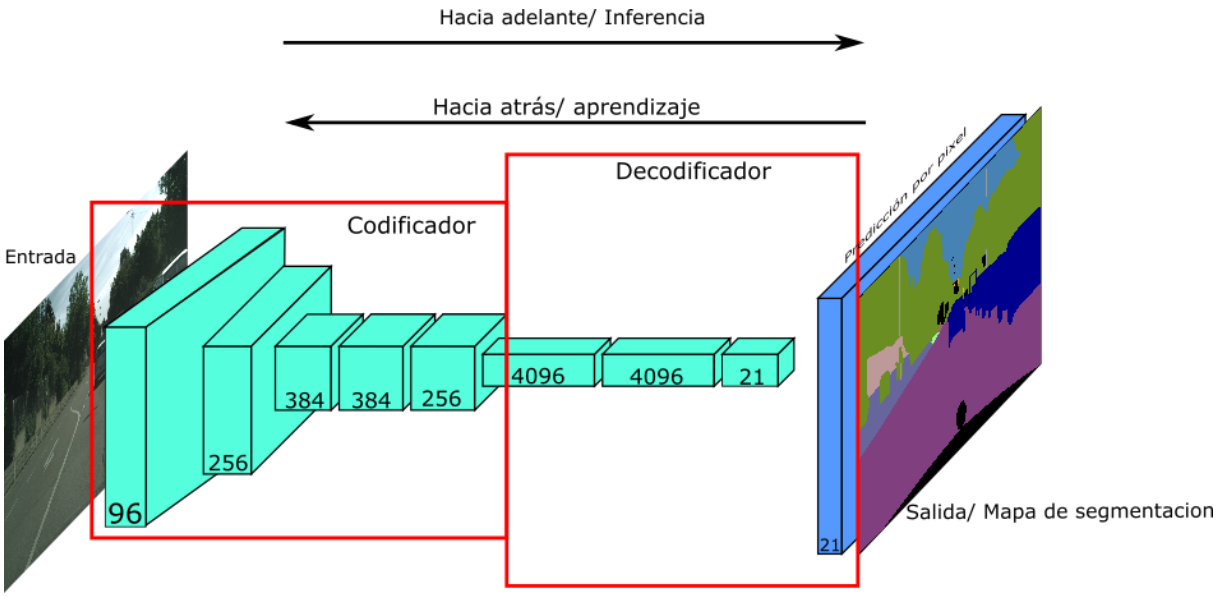

Figura 2.1 Arquitectura de la red completamente convolucional

2.1.1 Encoder

Este bloque se compone de distintos módulos que integran capas convolucionales, unidad de rectificación lineal (ReLU) y agrupamiento por máximos o *max pooling*. A mayor cantidad de módulos, mayor es la profundidad de la red.

Las capas de convolución encuentran características abstractas mediante la convolución entre los *kernels* $K(x,y)$ (también conocidos como filtros) y la salida del módulo anterior $Y(x,y)^{l-1,\,p_{l-1}}$ dado por:

$$W(x,y)^{l,P_l} = K(x,y)^{P_l} * Y(x,y)^{l-1,P_{l-1}} \tag{2.1}$$

Donde l es el módulo y P_l es la profundidad P del módulo l. (en otras palabras, p es la capa del módulo l), *K(x,y)* es el *kernel* compuesto por los coeficientes extraídos del entrenamiento. El encoder tiene L módulos $(l = 0, \dots, L)$. Para $l = 0$, $Y(x,y)^{0,P} = I(x,y,t)^{RGB}$ y $P = \{R,G,B\}$. $W(x,y)^{l,P_l}$ son las características extraídas en la $l-ésima$ capa y el tamaño de $W(x,y)^{l,P_l}$ este dado por $x \in R^{Ml}\, y \in R^{Nl}$ donde $X > M_l, Y > N_l$ y $M_l > M_{l-1} - 1, N_l > N_{l-1}$.

Después de cada capa de convolución, se encuentra una función de activación ReLU que es una función de activación definida por:

$$R(x,y)^{l,P_l} = \max\left[0, W(x,y)^{l,P_l}\right] \tag{2.2}$$

Esta capa determina el máximo entre 0 y el valor de cada elemento de la salida de la capa de convolución, haciendo cero los valores negativos. Esto genera que la salida de esta capa sea no lineal. Además, al ser una función rampa hace que el gradiente no se desvanezca conforme pasan las épocas de entrenamiento, ya que la derivada de una función rampa es una constante [32].

Al final de cada ReLU está la capa de pooling, que reduce la dimensión de las características obtenidas. Esta capa divide $R(x,y)^{l,P_l}$ en pequeñas ventanas $r(i,j,z)$, donde (i,j) es la posición de cada elemento, $i \in R^I, j \in R^J$, z es el número de ventana y esta capa de agrupamiento se obtiene con el máximo de cada ventana dado por:

$$Y(x,y)^{l,P_l} = \max[r(i,j,z)^l], \qquad x \in R^{M_l}, \qquad y \in R^{N_l} \tag{2.3}$$

La salida del encoder, es la salida del módulo final L, $Y(x,y)^{L,P_L}$, que contiene las características segmentación $F(x,y)$ donde $X \gg M_L, Y \gg N_L$. Cabe aclarar que cada módulo genera características con diferente nivel de resolución. El primer módulo presenta características con bajo nivel de resolución y conforme aumenta la profundidad de la red, mayor es el nivel de resolución.

2.1.2 Decoder

El decoder propuesto por Long et al. [17] se compone en un conjunto capas de convoluciones traspuestas dadas por:

$$Y(x,y)^{l+L+1,P_{l+L}} = G(x,y)^{P_{l+1+L}} * Y(x,y)^{l,P_{l+L}} \tag{2.4}$$

donde $G(x,y)^{P_{l+1+L}}$ es la traspuesta de $K(x,y)^{P_l}$, cuyo tamaño de cada capa está definido por $x \in R^{M_{l+L}}, y \in R^{N_{l+L}}$, $donde\ M_{l+L+1} > M_{l+L}, N_{l+L+1} > N_{l+L}$. La entrada al bloque del decoder es $Y(x,y)^{L,P_L} = F(x,y)$. La salida de $Y(x,y)^{2L}$ es la capa de segmentación $S(x,y)$,

donde $M_{2L} = X$ y $N_{2L} = Y$. Si L es el número de capas del encoder, $2L$ es el número de capaz del decoder.

2.1.3 Entrenamiento

Las redes convolucionales son modelos supervisados que se entrenan a partir de imágenes previamente etiquetadas. El modelo original de las FCN utiliza el método de optimización de gradiente estocástico descendiente con momento.

El algoritmo de entrenamiento de la FCN es el *backpropagation* y la idea principal es que la red aprenda con la propagación hacia atrás mediante el gradiente de la función de costo (pérdida) con respecto a los diferentes pesos. Estos gradientes son usados para actualizar los pesos y consideran una secuencia de grupos de neuronas ocultas $h_1, h_2, \dots, h_k$ seguidas de una salida O, con respecto al que se calcula la función de pérdida $\mathcal{L}$. Si se asume que el peso de las conexiones entre los grupos de neuronas ocultas h_r a h_{r+1} es $w(h_r, h_{r+1})$, entonces, se puede derivar el gradiente de la función de pérdida con respecto a cualquiera de estos pesos usando la regla de la cadena de:

$$\frac{\partial \mathcal{L}}{\partial w_{(hr-1,hr)}} = \frac{\partial \mathcal{L}}{\partial O} \cdot \left[\frac{\partial O}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \forall \in \ 1 \cdots k \tag{2.5}$$

Como en la realidad podrían existir una cantidad exponencial de caminos entre h_1 y O, existe una variante generalizada de la regla de la cadena, también conocida como regla de la cadena multivariable. Esta regla calcula el gradiente en un grafo, donde existe más de un camino agregando la composición a través de todos los caminos entre h_1 y O. Por lo tanto, una vez que se generaliza la expresión anterior al caso donde existan ρ caminos desde h_r y O la ecuación (2.5) ahora queda de la siguiente forma:

$$\frac{\partial \mathcal{L}}{\partial w_{(hr-1,hr)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial O} \cdot \left[\sum_{[h_r,\ h_{r+1},\ \cdots,\ h_{k,}\ O] \in \rho} \frac{\partial O}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{La\ propagacion\ hacia\ atrás\ cálcula\ \Delta(h_r,O)=\frac{\partial \mathcal{L}}{\partial h_r}} \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \tag{2.6}$$

El término ($\Delta(h_r, O) = \frac{\partial \mathcal{L}}{\partial h_r}$) se anexa por un número de caminos que se incrementan exponencialmente, que parece sin solución a simple vista. Sin embargo, una pieza clave es que un grafo computacional no contiene ciclos, y es posible calcular esa agregación mediante la propagación hacia atrás calculando primero $\Delta(h_k, O)$ para nodos h_k más cercanos a O, y después de manera recursiva se calculan estos valores para nodos en capas anteriores en términos de capas posteriores. No obstante, el valor de $\Delta(O, O)$ para cada nodo de salida se inicializa de la siguiente manera:

$$\Delta(\boldsymbol{O}, \boldsymbol{O}) = \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial O}} \tag{2.7}$$

Este tipo de técnicas de programación dinámica se usa frecuentemente para calcular de manera eficiente cualquier tipo de funciones centradas de camino en grafos acíclicos dirigidos, que de otra forma requerirían un número exponencial de operaciones. La recursión para $\Delta(h_r, O)$ se puede derivar usando la regla de la cadena multivariable, que se muestra a continuación:

$$\Delta(h_r, O) = \frac{\partial \mathcal{L}}{\partial h_r} = \sum_{h:h_r \Rightarrow h} \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial h_r} = \sum_{h:h_r \Rightarrow h} \frac{\partial h}{\partial h_r} \Delta(h, O) \tag{2.8}$$

Debido a que cada h es en una capa posterior a h_r, $\Delta(h, O)$ se calcula cuando $\Delta(h_r, O)$ es evaluado. Sin embargo, se tiene que evaluar $\frac{\partial \mathcal{L}}{\partial h_r}$ para calcular la ecuación (2.8). Si se tienen uniones desde h_r a h con pesos $w(h_r, h)$, y sea a_h el valor calculado en la unidad oculta h justo antes de ser aplicada la función de activación $\Phi(\cdot)$ (es decir, $h = \Phi(a_h)$ donde a_h es una combinación lineal de sus entradas de unidades ocultas de capas anteriores que inciden en h), entonces, con la regla de la cadena, la expresión $\frac{\partial h}{\partial h_r}$ se puede derivar de la siguiente forma:

$$\frac{\partial h}{\partial h_r} = \frac{\partial h}{\partial a_h} \frac{\partial a_h}{\partial h_r} = \frac{\partial \Phi(a_h)}{\partial h} \cdot w(h_r, h) = \Phi'(a_h) \cdot w(h_r, h) \tag{2.9}$$

El valor de $\frac{\partial h}{\partial h_r}$ es usado en la ecuación (2.8), quien es repetido de forma recursiva en el paso hacia atrás, empezando por el nodo de salida. Las actualizaciones correspondientes en el paso hacia atrás se representan de la siguiente manera:

$$\Delta(h_r, O) = \sum_{h:h_r \Rightarrow h} \Phi'(a_h) \cdot w(h_r, h) \cdot \Delta(h, O) \tag{2.10}$$

Los gradientes se acumulan sucesivamente en el paso hacia atrás, y cada nodo es procesado con el mismo paso hacia atrás. Cabe destacar que en el cálculo de la ecuación (2.8) se repite para cada borde entrante en el nodo para calcular el gradiente con respecto a todos los pesos de los bordes. Finalmente, la ecuación 2.6 requiere el cálculo de $\frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}}$, que se realiza como se muestra en la ecuación 2.11:

$$\frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} = h_{r-1} \cdot \Phi'\left(a_{h_r}\right) \tag{2.11}$$

Aquí, el gradiente clave que se propaga hacia atrás es derivable con respecto a las capas de activación, y el gradiente con respecto a los pesos es calculado para cualquier borde incidente a la unidad correspondiente [32].

Este algoritmo se emplea en todas las variantes de las FCN, incluso, en casos particulares donde las FCN se combinan con las CNN tradicionales de clasificación. Por ejemplo, en el caso particular de la red FCN de long Et al. [17] en la que se tomó como encoders las redes de clasificación AlexNet, VGG16 y GoogleNet. De esta manera, se usaron lotes de 20 imágenes con factor de aprendizaje η fijo de 10^{-3}, 10^{-4}, 5^{-5} para las versiones de AlexNet, VGG16 y GoogleNet, respectivamente. Se utilizó un momento μ de 0.9, una caída de pesos de 5^{-4} o 2^{-4}, doble factor de aprendizaje para el bias, además de mantener el dropout de los modelos clasificadores originales. Esto mejoró la calidad de la segmentación, además de facilitar el desarrollo de nuevos modelos, ya que se parte de esquemas preestablecidos.

2.1.3.1 Optimizadores

Los algoritmos de optimización son modelos que permiten actualizar los parámetros o pesos de las redes neuronales y minimizar el valor de la función de pérdida para obtener los resultados más precisos posibles. Por ello, un amplio entendimiento de los algoritmos de optimización permite ajustar los hiper-parámetros para mejorar el rendimiento de los modelos de aprendizaje profundo. A continuación, se presentan los optimizadores más utilizados.

Gradiente estocástico descendiente con momento

Este optimizador acelera vectores de gradientes de la función de costo (pérdida) hacia un mínimo local y guía hacía la convergencia de los pesos en menos iteraciones.

Una función objetivo es el promedio de las funciones de pérdida por cada muestra en el conjunto de entrenamiento. Si se asume $f_i(\psi)$ como la función de costo del conjunto de entrenamiento con n muestras, un índice i y un vector de parámetros ψ, entonces se describe la función objetivo como:

$$f(\psi) = \frac{1}{n}\sum_{i=1}^{n} f_i\,(\psi) \tag{2.12}$$

El gradiente de esta función en ψ se calcula por medio de:

$$\nabla f(\psi) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i\,(\psi) \tag{2.13}$$

Con cada iteración del gradiente estocástico descendiente (SGD), se muestrea un índice $i \in \{1, \dots, n\}$ uniformemente distribuido para instancias de datos al azar y se calcula el gradiente $\nabla f_i(\psi)$ para actualizar ψ:

$$\psi \leftarrow \psi - \eta \nabla f_i(\psi) \tag{2.14}$$

donde, η es el factor de aprendizaje.

Para acelerar el descenso del gradiente, se utiliza el momento. Este consiste en una técnica para acelerar el descenso del gradiente, que acumula un vector de velocidad en direcciones de reducción persistente en un objetivo a través de las iteraciones, dada una función objetivo $f(\psi)$ a minimizar, el momento está dado por:

$$V_{t+1} = \mu V_t - \eta \nabla f(\psi) \tag{2.15}$$

$$\psi_{t+1} = \psi_t + V_{t+1} \tag{2.16}$$

Donde $\mu \in [0,1]$ es el coeficiente de momento y $\nabla f(\psi)$ es el gradiente en ψ.

ADAM

El optimizador de momento adaptivo (*adaptive momentum* o ADAM) es un método que calcula diferentes factores de aprendizaje adaptivos individuales para diferentes parámetros que estiman el primer y el segundo momento de los gradientes.

Uno de los componentes principales del ADAM es una media móvil ponderada exponencial (también conocida como *leaky averaging*) para obtener un estimado del primer y segundo momento del gradiente de la función de perdida, para ello se usan las siguientes variables de estado.

$$v_t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t, \tag{2.17}$$

$$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) {g^2}_t$$

Donde β_1 y β_2 son los parámetros de ponderación. Comúnmente, se utilizan $\beta_1 = 0.9$ y $\beta_2 = 0.999$. Es decir, la estimación de la varianza se mueve de manera más lenta que el término de momento. Cabe destacar que si se inicializa $v_0 = s_0 = 0$, se tiene gran cantidad de sesgo hacia valores más pequeños. Esto se puede arreglar usando la expresión dada por la ecuación 2.18 que normaliza los parámetros. A su vez se obtienen las variables de estado normalizadas, dadas por la ecuación 2.19

$$\sum_{i=0}^{t} \beta^i = \frac{1-\beta^t}{1-\beta} \tag{2.18}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_1^t} \; y \; \hat{s}_t = \frac{s_t}{1-\beta_2^t} \tag{2.19}$$

2.2 Técnicas para redes convolucionales eficientes

Uno de los problemas de las FCNs es su complejidad computacional. Por ello, se han propuesto diversas técnicas para reducir esta complejidad que se basan en reducir la cantidad de operaciones en las convoluciones o en la estructura de la red. Por ello, en esta sección se presenta una colección de diferentes técnicas para obtener redes convolucionales eficientes para segmentación semántica.

2.2.1 Capas convolucionales

Las capas convolucionales de las FCN suelen tener un alto costo computacional debido a que se suele aumentar el número de capas convolucionales para aumentar la precisión en las características de la segmentación. Por ello, numerosos trabajos han optado por enfocarse en reducir los tiempos de procesamiento de las capas convolucionales.

En la Figura 2.2 se puede apreciar la obtención de un mapa de características mediante el uso de estas capas convolucionales, en donde se puede ver que el resultado de una convolución entre una ventana de la entrada Y en el lado izquierdo con el filtro convolucional K resulta en un valor del total del mapa de características W marcado en rojo. Al pasar el filtro convolucional por toda la entrada, se genera un mapa de características en donde si el filtro es de un número de canales mayor a uno, tendrá un mapa de características de mayor dimensión que la entrada *Y*. En el caso de una convolución tradicional, el mapa de características se define como como $W \in \mathbb{R}^{C_i x d x d x C_o}$, donde C_i es el número de canales de entrada, C_o es el número de canales de salida y *dxd* representa el tamaño del *kernel* de cada convolución. De esta manera, una convolución estándar para entradas de dos dimensiones (como una imagen multicanal) tiene $dxdxC_ixC_o$ parámetros.

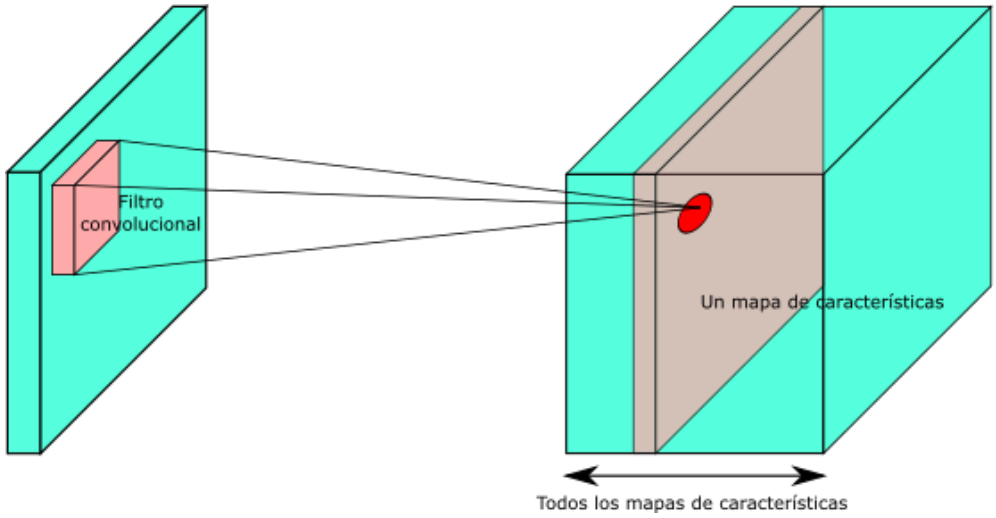

Figura 2.2 Filtro convolucional.

Entre las distintas técnicas para hacer las convoluciones más eficientes, están las factorizadas, las separables en profundidad, las dilatadas y las dilatadas separables en profundidad.

Convoluciones factorizadas: propuestas en [33], se componen de dos convoluciones consecutivas que factorizan el *kernel*. De esta manera, una convolución factorizada se compone de dos convoluciones de *1xd* y *dx1 kernels*, dejando así en $2xdxC_ixC_o$ parámetros, a diferencia de la convolución estándar que tiene $dxdxC_ixC_o$ parámetros. Por ejemplo, si se tiene un *kernel* de 3x3 los parámetros se reducen un 33% con respecto a las convoluciones estándar. Las convoluciones factorizadas están definidas como se muestra a continuación:

$$G_{k,l,C_o} = \sum_{d,d,C_i} W_{d,1,m,n} \cdot W_{1,d,m,n} \cdot F_{k+d-1,\ l+d-1,\ C_i} \tag{2.20}$$

Convoluciones separables en profundidad: propuestas en [34], consisten en separar la convolución estándar en dos. La primera es llamada convolución en profundidad y se utiliza para aplicar un filtro para cada canal de entrada, tal como se observa en la siguiente ecuación:

$$\hat{G}_{k,l,C_o} = \sum_{d,d} \widehat{W}_{d,d,C_i} \cdot F_{k+d-1,\ l+d-1,\ C_i} \tag{2.21}$$

Donde $\widehat{W}$ es el *kernel* convolucional de tamaño $dxdxc_i$, en que el c_i-ésimo filtro en $\widehat{W}$ es aplicado al c_i-ésimo canal en F para producir el c_i-ésimo canal de la salida filtrada $\hat{G}$. Por lo tanto, esta convolución en profundidad tiene C_ixdxd parámetros (teniendo en cuenta que C_i=C_o). La segunda convolución es llamada convolución por punto, y consiste en realizar una convolución de 1x1, para formar una combinación lineal que produce una salida de diferente número de canales, proyectando y combinando los canales de salida de la convolución en profundidad a un nuevo espacio de canales. Esta convolución por punto tiene C_ixC_o parámetros y se expresa de la siguiente manera:

$$G_{k,l,C_o} = \sum_{Co} \omega_{1,1,C_i} \hat{G}_{k,l,C_o} \tag{2.22}$$

Las convoluciones por profundidad y por punto dejan un total de $C_ixdxd + C_i$xC_o parámetros. Por ejemplo, si se tiene un tamaño de *kernel* de 3x3 y 103 *kernels*, se obtiene una

reducción del 88% de parámetros con respecto a las convoluciones estándar. En la Figura 2.3 se puede ver como en la convolución en profundidad solo se realiza una convolución para cada canal, para posteriormente ser proyectado a una nueva dimensión de canales mediante la convolución de 1x1.

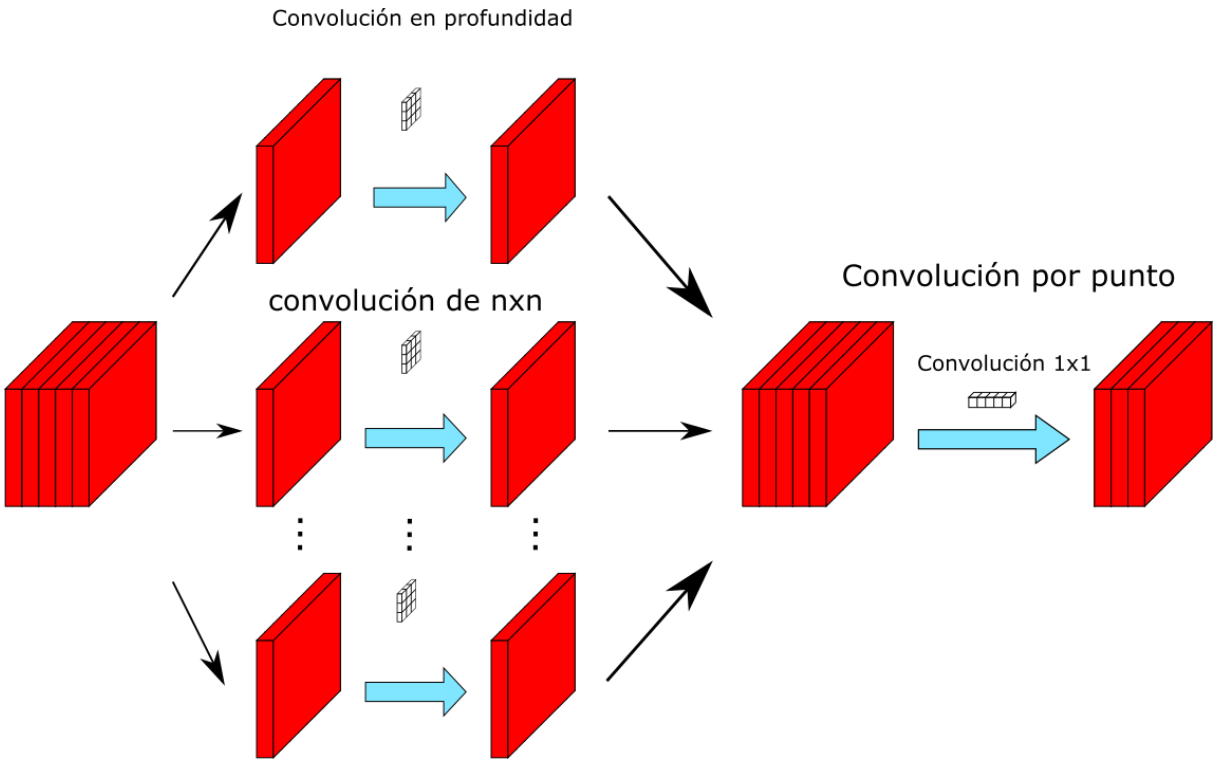

Figura 2.3 Convolución separable en profundidad.

Convoluciones dilatadas: se proponen en [35], y se introduce el factor de dilatación r, que consiste en aumentar el número de saltos en la cardinalidad del *kernel* en dos valores. Por ejemplo, la Figura 2.4 muestra tres *kernels* con diferentes niveles de dilatación. En la Figura 2.4a se puede ver un *kernel* de 3x3 con $r = 1$, la Figura 2.4b muestra un *kernel* de 5x5 con *r=2* y el mismo campo receptivo, pero solo usando 9 parámetros en lugar de 25. A su vez, la Figura 2.4c presenta un *kernel* de 3x3 con $r = 3$ con un campo receptivo de 7x7. Cuando $r = 1$, no existe dilatación y solo sería una convolución tradicional.

Este tipo de convolución ayuda a reducir el número de capas y el número de parámetros, a pesar de que su uso no está bien establecido en el ámbito de la segmentación semántica debido a que conforme crece el factor de dilatación, se pierde información local.

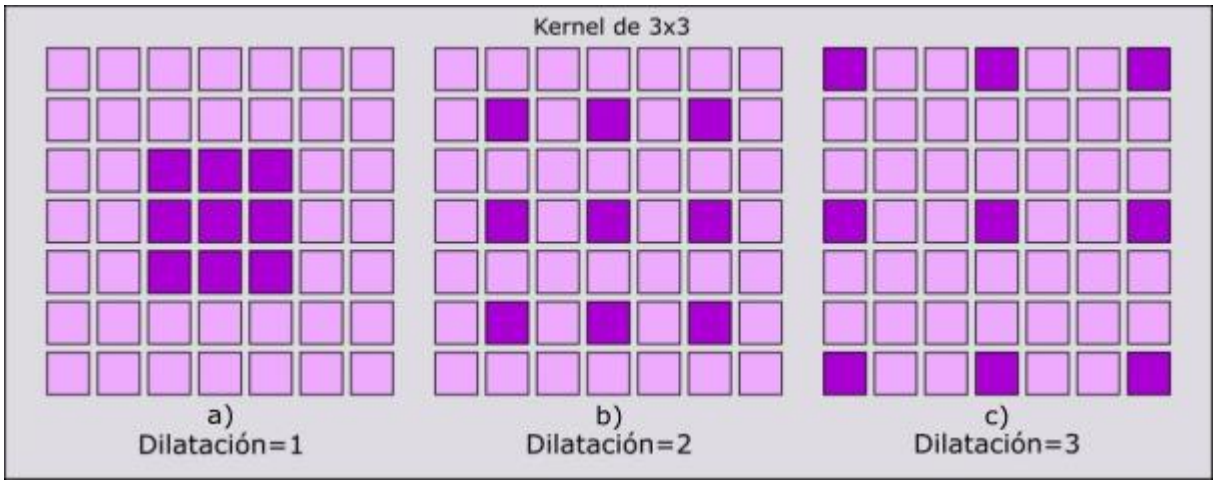

Figura 2.4 Convolución dilatada con diferentes factores de dilatación, 1,2 y 3 respectivamente.

Convolución dilatada separable en profundidad: introducida en [36], consiste en dos convoluciones paralelas separables en profundidad, una con factor de dilatación de $r=1$, G^1 y la segunda con un factor de dilatación $r>1$, G^r ambas dadas por la ecuación (2.20). Después, sus salidas son sumadas y pasadas por una convolución puntual dada por la ecuación G (2.20) con $d=1$ de $1x1$, como se observa en la Figura 2.5. Esta convolución mejora el desempeño porque aprende eficientemente la relación entre el contexto global y el local. Esta operación solo añade $C_i xdxd$ parámetros con respecto a la convolución separable en profundidad y tiene una reducción de 87% de parámetros respecto a la convolución estándar.

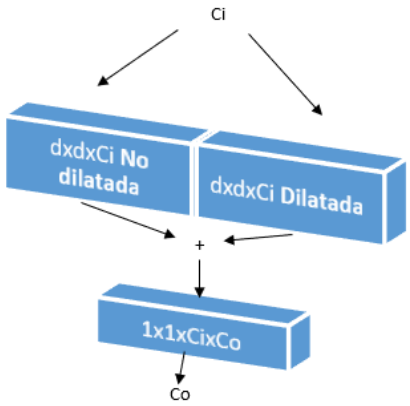

Figura 2.5 Convolución separable en profundidad multi-dilatada.

2.2.2 Otras técnicas

Aparte de modificar la operación de la convolución, existen otras técnicas para hacer eficiente una red neuronal y en esta sección se van a detallar algunos de estos métodos.

Compresión de la red: Han et al. [37] propone un método denominado poda, en el que se entrena una red neuronal para aprender las conexiones entre neuronas, luego se vuelve a entrenar eliminando un porcentaje de conexiones. Este método imita cómo el ser humano va perdiendo neuronas conforme envejece, pero no pierde la capacidad de aprender.

Por ejemplo, la Figura 2.6 presenta tres diferentes experimentos: en el primero (color morado), solo se utiliza la poda, sin reentrenar la red. En el segundo experimento (verde) se realiza un reentrenamiento con poda y el tercer experimento (marcado con rojo), se realiza un reentrenamiento iterativo el cual consiste en estar entrenando y eliminando conexiones hasta llegar a unos umbrales de precisión. Gracias a este concepto de poda, se logra llegar a un equilibrio entre precisión y cantidad de parámetros adecuado que se encuentra marcado con una flecha.

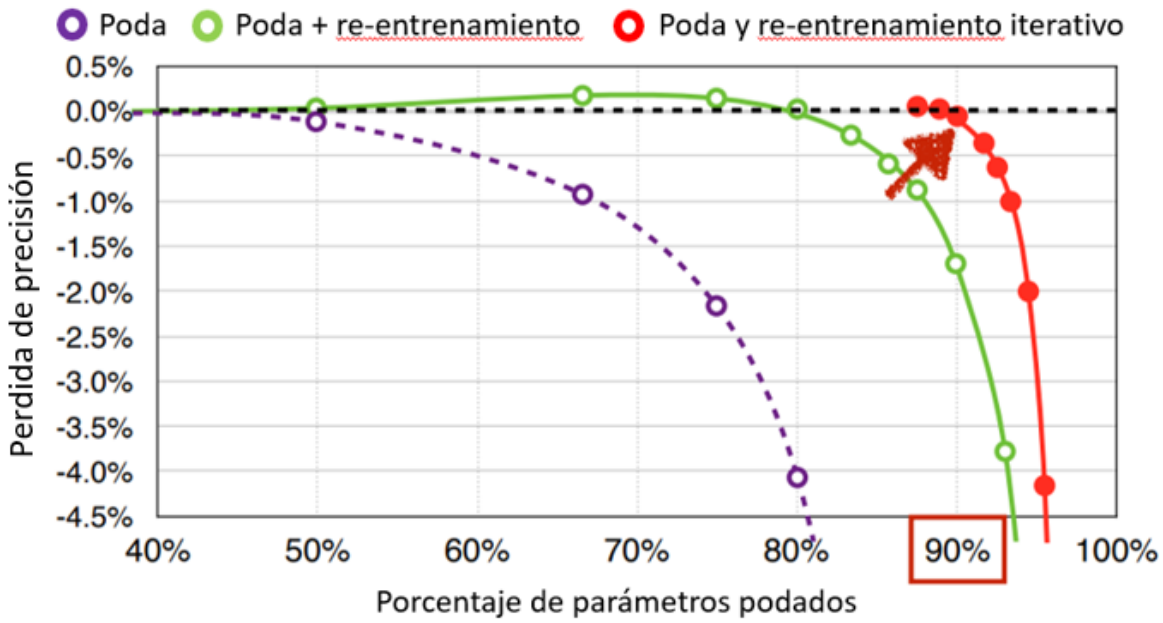

Figura 2.6 Resultados del método de poda y reentrenamiento iterativo.

Cuantización de la red: consiste en reducir la cantidad de bits del tipo de dato que se utiliza para realizar las operaciones dentro de la red. Comúnmente, se reduce de punto flotante de 32 bits a 16 bits [38].

Arquitecturas maestro-estudiante: en [39] se involucran dos redes, una estudiante y otra maestra. La estudiante es una red para segmentación con una arquitectura con menor número de parámetros y capas respecto a la red maestro. De esta manera, la red maestra es más profunda y tiene una arquitectura más compleja que la estudiante y se utiliza para complementar la información que otorga el *ground-truth.*

Durante el entrenamiento, la combinación maestro-estudiante hace que la red estudiante herede el aprendizaje del maestro. Durante la prueba, la red estudiante se utiliza para hacer la inferencia y conserva las características descritas por el modelo de aprendizaje de la red maestra. Otra ventaja de estos modelos es que se puede trabajar de manera semi supervisada ya que la red estudiante puede tomar como *ground-truth* las imágenes de salida de la red maestra, así poder implementar imágenes fuera de la base de datos que se esté usando.

2.3 Selección de modelos

Existen múltiples redes propuestas en la literatura que utilizan cualquiera de las estrategias mencionadas para desarrollar FCN's enfocadas a segmentación semántica. Por ello, para desarrollar el modelo a proponer en esta tesis, se realizó un análisis que consistió en diseñar un instrumento de ponderación por métricas. Este análisis se llevó a cabo en 2 etapas: la selección de modelos y la comparación.

En la selección, se utilizaron 3 criterios para elegir diferentes modelos de la literatura como base para el modelo a desarrollar. Debido a que el objetivo de esta investigación es desarrollar un modelo capaz de correr en tiempo real en una plataforma embebida, se hizo hincapié en los modelos que priorizan la eficiencia computacional. Estos criterios fueron:

1. Que el código del método propuesto esté disponible en Internet.

2. Que los modelos tengan un enfoque de segmentación semántica y eficiencia computacional.

3. Los datos de entrenamiento estén disponibles en la base de datos de Cityscapes.

A partir de estos criterios, se seleccionaron 18 modelos que fueron:

Tabla 2.1 Listado de las redes seleccionadas.

Nombre del artículo	**Nombre de la red**	**Año**	**Referencia**
A deep convolutional Encoder-Decoder Architecture for image segmentation	SegNet	2017	[19]
Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation	ERFNet	2018	[40]
A Lightweight Encoder-Decoder Network for Real-Time Semantic Segmentation	LEDNet	2019	[41]
Cross-guidance network for semantic segmentation	CGNet	2020	[42]
Fast Semantic Segmentation for Scene Perception	FSSNet	2019	[12]

Depth-Wise Asymmetric Bottleneck with Point-Wise Aggregation Decoder for Real-Time Semantic Segmentation	DABNet	2020	[43]
UNet: *Convolutional Network for Biomedical Image Segmentation*	UNet	2015	[44]
Speeding up Semantic Segmentation for Autonomous Driving	SQNet	2016	[45]
A Light-Weight, Power Efficient, and General Purpose Convolutional Neural Network	ESPNetV1 y V2	2018y 2020	[46, 47]
An Efficient Symmetric Network for Real-Time Semantic Segmentation	ESNet	2019	[48]
Exploiting encoder representations for efficient semantic segmentation	LinkNet	2017	[49]
Efficient Dense Modules of Asymmetric Convolution for Real-Time Semantic Segmentation	EDANet	2019	[50]
Feature Pyramid Encoding Network for Real-Time Semantic Segmentation	FPENet	2019	[51]
Exploring Context and Detail for Semantic Segmentation in Real-Time	ContextNet	2018	[52]
FastSCNN: Fast semantic Segmentation Network	FastSCNN	2019	[53]
An efficient semantic segmentation ConvNet for real-time robotics applications	Mininet-V2	2020	[36]

Estos modelos cumplen los criterios mencionados anteriormente y también están desarrollados bajo en el entorno de Pytorch (excepto Mininet), lo que facilita el realizar pruebas de comparativa de velocidad de inferencia, parámetros utilizados y número de operaciones.

La base de datos que se seleccionó para las pruebas fue Cityscapes [28] ya que es una de las más utilizadas en la literatura y tiene un amplio número de imágenes con ground-truth. Esta base de datos cuenta con 5,000 imágenes divididas en 30 clases y tiene escenarios urbanos

tomados desde la perspectiva frontal de un automóvil. Las imágenes fueron obtenidas de 50 ciudades de Alemania con diferentes condiciones climáticas y en diferentes épocas del año. Además, cuenta con imágenes manualmente seleccionadas para incluir una gran cantidad de objetos dinámicos y que varíe el entorno y el fondo de las imágenes.

2.3.1 Instrumento de análisis (etapa de comparación)

Una vez realizada la selección de FCNs eficientes, estas se compararon mediante una implementación en un sistema embebido para poder realizar un análisis de su funcionamiento. Esta implementación se realizó en el lenguaje de programación Python, ya que es el lenguaje de programación predominante en el desarrollo de CNNs eficientes para segmentación semántica.

Analizar las 18 FCNs para proponer una en esta tesis, es un proceso muy largo y complicado. Por ello, fue necesario buscar la manera de acotar la cantidad de FCNs, de manera que se puedan analizar aquellas que tienen mejor relación entre velocidad y precisión. Por ello, se desarrolló un instrumento de ponderación por métricas para selección de variables que utiliza tres variables relacionadas con eficiencia computacional:

Velocidad: se basa en el tiempo que la red tarda en hacer la inferencia. Para presentar la velocidad, es necesario hacer una normalización debido a que la diferencia entre cada modelo reportado en Cityscapes es de milésimas de segundo. Velocidad es la métrica que más peso tiene, ya que indica que tan eficiente es el modelo y está dada por:

$$S = \left(1 - \frac{Runtime}{Max(Runtime)}\right) * 40 \tag{2.23}$$

S está definida de manera que cuanto menor sea el tiempo de procesamiento, el resultado es mejor y se asemeja más a uno. A la velocidad se le otorga la ponderación de 40 ya que es la métrica con la que mejor se puede relacionar la eficiencia computacional. La velocidad en la base de datos de Cityscapes se mide mediante la herramienta $Runtime$ la cual mide el tiempo que toma en hacer la inferencia el modelo.

Para calcular el tiempo de procesamiento de los modelos, de manera general, es necesario una librería de Python llamada *time*, que permite calcular el tiempo transcurrido entre dos operaciones.

Precisión: el cálculo de la precisión en segmentación semántica se hace mediante el promedio del número de intersecciones sobre la unión (IoU) de cada clase, el cual se define por:

$$IoU = \frac{TP}{TP + FN + FP} \tag{2.24}$$

donde TP son los verdaderos positivos (pixeles clasificados correctamente), FN son los falsos negativos y FP son los píxeles falsos positivos. Estos parámetros son determinados con una matriz de confusión hecha con las clases de la base de datos utilizada. Un ejemplo se puede apreciar en la Figura 2.7, en donde se observa que la clase ‘camino’ tuvo 99 verdaderos positivos y un falso positivo donde un píxel de acera lo clasificó como ‘camino’. El resto de las etiquetas para la clase ‘camino’ fueron falsos negativos conformados por: 13 píxeles que se clasificaron cuatro como ‘acera’, dos como ‘pared’, dos como ‘terreno’, uno como ‘persona’, uno como ‘auto’, uno como ‘camión’, uno como ‘camioneta’ y otro como ‘motocicleta’. Con base en el cálculo de IoU de cada clase se obtiene un promedio, luego se promedia con el resto de las imágenes de la base de datos para obtener el mIOU el cual es tomado como referencia para el instrumento.

		Predicho																		
		Camino	Acera	Edificio	Pared	Valla	Poste	Semáforo	Señal de tráfico	Vegetación	Terreno	Cielo	persona	Auto	Camioneta	Autobús	Tren	Motocicleta	Bicicleta	conductor
Valor real	Camino	99	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Acera	4	93	1	0	0	0	0	0	0	1	0	4	0	0	0	0	0	0	0
	Edificio	0	0	97	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
	Pared	2	3	18	65	4	1	0	0	4	1	0	1	0	1	0	0	0	0	0
	Valla	0	1	14	4	71	1	0	1	4	0	0	0	0	1	0	0	0	1	0
	Poste	0	0	0	0	0	73	0	0	0	0	0	0	0	0	5	0	0	0	0
	Semáforo	0	0	0	0	0	0	83	0	0	0	0	0	0	0	0	0	0	0	0
	Señal de tráfico	0	0	0	0	0	0	0	86	0	0	0	0	0	0	0	0	0	0	0
	Vegetación	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0	3	5	0
	Terreno	2	0	0	0	2	0	0	0	0	74	0	0	0	0	0	0	0	0	0
	Cielo	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0	0
	persona	1	0	0	0	0	0	5	0	0	0	0	92	0	0	0	0	0	0	0
	Auto	1	0	0	0	0	0	0	0	0	0	0	0	78	0	0	0	5	0	0
	Camioneta	1	0	0	2	4	0	0	0	0	0	0	0	0	98	0	0	0	0	0
	Autobús	1	0	0	0	0	0	0	0	2	0	0	0	0	0	67	0	0	0	0
	Tren	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	94	0	0	0
	Motocicleta	1	0	0	0	0	0	0	0	0	3	0	0	0	2	0	0	85	0	0
	Bicicleta	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89	0
	conductor	0	0	0	0	0	0	0	2	0	0	0	2	0	0	0	0	0	0	80

Figura 2.7 Matriz de confusión de 20 clases de Cityscapes.

Para tener una clara representación de los resultados del instrumento se optó por trabajar con base en porcentajes, debido a esto es necesario acotar la ponderación de esta métrica en valores de 0 a 30.

Si bien, la precisión no se relaciona con la eficiencia computacional, si dice bastante acerca del funcionamiento de la red. Por ello para el cálculo de la precisión se describe como:

$$P = \frac{mIoU}{100} * 30 \tag{2.25}$$

La ecuación (2.24) se debe a que el mIOU se calcula como porcentaje que otorga valores de 0 a 100, por eso es necesario dividir entre 100 y multiplicar por 30 para acotarlo en valores de 0 a 30.

Reproducibilidad: se refiere a que tan factible es reproducir las redes propuestas en la literatura. Para esta métrica, se consideraron la disponibilidad del código y la disponibilidad de los pesos de entrenamiento. La disponibilidad del código se refiere a que el programa de la red y sus parámetros se encuentre disponible en Internet y se pueda correr.

Por ejemplo, en ocasiones la red está disponible, pero los parámetros se deben entrenar. Otras veces, los pesos de entrenamiento también están disponibles, lo que ayuda a que la red se pueda implementar rápidamente, ya que las FCNs suelen tardar hasta meses en entrenar. La ponderación de esta métrica esta descrita de la siguiente forma:

$$D = \begin{cases} 30 & Si\ código\ y\ pesos\ disponible \\ 15 & Si\ solo\ código \\ 0 & Si\ nada \end{cases} \tag{2.26}$$

El hecho de castigar con 15 puntos si un modelo no reporta pesos es porque de ser así, se tendría que entrenar y eso podría tomar hasta meses, dependiendo del método de entrenamiento y las épocas que se hayan utilizado en dichos modelos. Por lo que, a pesar de no ser la métrica con más peso, si fue la métrica más discriminante ya que no se tomaron en cuenta métodos que no presentaran el código con el que se elaboró.

La métrica de reproducibilidad se utiliza porque muchos autores reportan buenos resultados en la literatura, pero no existe forma de probar que funcionen tal como se reporta en los artículos.

Análisis de métricas: una vez que se obtienen las métricas de S, P y D, se realiza la siguiente sumatoria:

$$\Sigma = \mathrm{S} + P + D \tag{2.27}$$

Σ resume la relación entre velocidad y precisión con un ligero enfoque en el tiempo de procesamiento.

2.3.2 Resultados

Para encontrar Σ y conocer cómo funcionan las FCNs eficientes en una plataforma embebida, se hizo una comparación de dichas redes con la tarjeta GPU embebida Jetson TX2. Esta tarjeta

cuenta con un CPU ARM de cuatro núcleos y un segundo CPU activable por software de dos núcleos Nvidia Denver de 64 bits. Además, cuenta con una GPU de la arquitectura Pascal con 256 núcleos CUDA y de memoria tiene 8 GB de RAM y 32 GB de almacenamiento.

Estos experimentos se realizaron porque aun y cuando se reportan diversos resultados de las FCNs eficientes en la literatura, es importante corroborar el funcionamiento de dichos modelos en una plataforma embebida y que todos los modelos sean probados en las mismas condiciones. En esta primera instancia, se busca encontrar las redes más veloces y analizar qué parámetros podrían servir para determinar la eficiencia computacional. Sin embargo, antes de valorar las métricas de Σ, se habían considerado también otras métricas para determinar cuáles eran las que impactan de mayor manera en la selección de FCNs eficientes. Esta consideración de métricas se debe a que en la literatura enfocada a estas redes se miden los FLOPs y la cantidad de parámetros que se pueden entrenar en la red las cuales impactan directamente en el tiempo de entrenamiento, en vez de la velocidad de inferencia. Por lo tanto, la Tabla 2.1 muestra los resultados del desempeño de los modelos en términos de las siguientes métricas:

Cuadros por segundo (FPS): es el número de imágenes que el sistema procesa en un segundo. Este es el recíproco del valor $runtime$ de la métrica S del instrumento descrito anteriormente. De esta manera, FPS es el inverso de *S*.

FLOPS: número de operación de punto flotante utilizados en el modelo.

Parámetros: cantidad de pesos y bias que se deben entrenar en una red neuronal.

Tabla 2.2 Resultados de la comparativa obtenidos en la tarjeta Jetson TX2

Modelo	**FPS**	**FLOPs**	**Parámetros**	**mIOU***
FastSCNN	24.91	1.76 G	1.14 M	69.22
ContextNet	22.03	1.78 G	876.56 K	66.1
FPENet	12.83	1.58 G	115.12 K	68
ESPNet	9.88	3.45 G	201.54 K	60.3
DABNet	6.96	10.46 G	756.64 K	66.8

FSSNet	6.86	2.78 G	175.88 K	58.8
EDANet	6.43	8.95 G	689.49 K	65.1
LinkNet	6.26	21.9 G	11.54 M	76.4
CGNet	5.59	7.01 G	496.31 K	59.7
ENet	4.77	4.35 G	360.42 K	58.3
LEDNet	4.13	11.51 G	917.39 K	69.2
ESNet	3.28	24.35 G	1.66 M	70.7
ERFNet	3.23	26.86 G	2.07 M	68
ESPNetV2	2.31	0.597 G	1.8 M	59.1
SQNet	1.28	144.11 G	16.26 M	59.3
UNet	0.94	248.94 G	14.79 M	---
SegNet	0.8	326.26G	29.45 M	56.1

Como se logra apreciar en la Figura 2.8, no existe una relación directa entre la reducción de parámetros y FLOPS con la reducción de la velocidad. Por ejemplo, FPENet tiene 10 veces menos parámetros con respecto a Fast-SCNN, pero Fast-SCNN es 2 veces más rápido. Por lo que, para la realización del instrumento de análisis, se descartaron los FLOPs y los parámetros, y se consideró solamente los FPS que se relacionan inversamente con la velocidad de inferencia. Además, otro punto a considerar es que, ya que gran parte de los modelos de la Tabla 2.1 no contaban con los pesos ya entrenados, por lo que en esta tabla se utilizaron solo los datos de precisión (mIOU) presentados en los artículos correspondientes, los demás datos fueron obtenidos de la implementación en la tarjeta Jetson TX2.

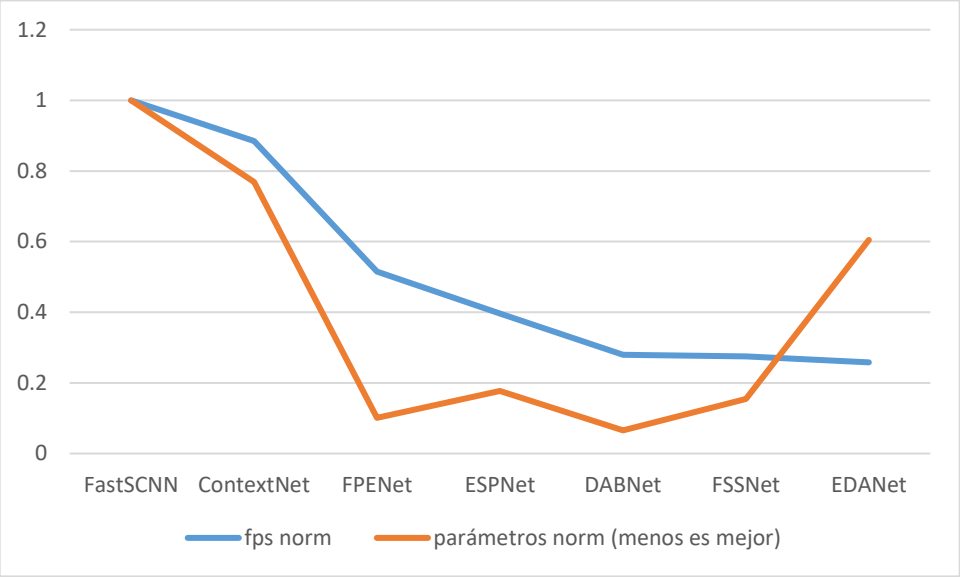

Figura 2.8 Gráfica de parámetros vs FPS en los 6 modelos más veloces

Una vez que se determinó que la ecuación 2.27 contiene la información suficiente para seleccionar los modelos, se realizó un experimento que consistió en la comparación de los modelos que quedaron después del descarte anterior mediante la implementación de la ecuación 2.27.

Luego de la implementación de los diversos modelos seleccionados, se decidió comparar los resultados obtenidos contra los del benchmark que se encuentra en la página web de la base de datos de Cityscapes https://www.cityscapes-dataset.com/benchmarks/. Esta comparación se realizó debido a que se encontraron algunas discrepancias entre los datos presentados en los artículos de las FCNs y los presentes en la página web. Estas diferencias pueden ser por diversos factores, ya que en la página web se presentan los mejores resultados que obtuvieron los modelos o se manejaron con diferentes resoluciones. Además, utilizando los datos de la página web se tiene una mejor representación de los tiempos, ya que están normalizados respecto al hardware que se utilizó, así como el algoritmo para calcular el tiempo de inferencia. Con esto, se pudo filtrar mejor las FCNs que no tienen sus resultados publicados en la página web de Cityscapes.

A partir del experimento mostrado en la Tabla 2.1 y de la información presentada en el benchmark de Cityscapes, se eligieron los métodos que presentaban datos de tiempo de

inferencia para así poder utilizar las métricas de IoU Class y Runtime presentes en dicho benchmark para el desarrollo del instrumento. De esta manera, la Tabla 2.2, muestra el resultado de la implementación del instrumento con los datos presentes en la base de datos de Cityscapes, además de presentar la precisión que tienen los modelos a mitad de resolución.

Tabla 2.3 Resultados del instrumento de análisis

Modelo	**Runtime**	**IoU Class**	**D**	**Σ**
Fast-SCNN*	0.0035	54.839	1	**84.12**
Mininet-V2	0.012	67.8	1	**82.34**
ESPNet	0.0089	60.3	1	**82.16**
ERFNet	0.02	69.7	1	**77.58**
EDANet	0.0092	67.3	0.5	69.06
Enet	0.013	58.3	0.5	63.82
CGNet	0.02	64.8	0.5	61.11
ContextNet	0.0238	66.1	0.5	58.96
SQNet	0.06	59.8	0.5	32.94
SegNet	0.06	57	0.5	32.1

Como se puede apreciar, los modelos con mayor ponderación también coinciden que son los más rápidos y que son más reproducibles. De esta manera, este análisis da como resultado que Fast-SCNN, Mininet-V2 y ESPNet como los modelos más plausibles para ser utilizados en el desarrollo del modelo a proponer en esta tesis.

Cabe aclarar que el modelo FastSCNN presenta una precisión diferente a la del benchmark de Cityscapes porque se tuvo que analizar e interpretar en Pytorch ya que el código original no está disponible.

CAPÍTULO 3
Propuesta

Del análisis presentado en el capítulo anterior, las redes Fast-SCNN, Mininet-V2 y ESPNet fueron los modelos seleccionados como base para la red a desarrollar en esta tesis. De esta manera, para poder presentar la FCN propuesta, es imperativo conocer la arquitectura de estas FCNs. Consecuentemente, en este capítulo se presenta la arquitectura de las redes seleccionadas, para después presentar la red propuesta en esta tesis.

Este capítulo se organiza de la siguiente manera: la sección 3.1 presenta Fast-SCNN, la sección 3.2 describe Mininet-V2, la sección 3.3 reporta ESPNet y finalmente, la sección 3.4 presenta la red propuesta, la cual se denominó Red de tiempo real Consciente del Contexto para sistemas embebidos (*CARTNet: Context aware and real time network for embedded systems*).

3.1 Fast-SCNN

Fast-SCNN [53] es una arquitectura de FCN eficiente que toma inspiración de dos tipos de arquitecturas:

Dos ramas: es una arquitectura con dos niveles de profundidad. La de mayor profundidad utiliza una imagen de entrada de menor resolución que se encarga de encontrar las características del contexto global. Por otro lado, la red de menor profundidad utiliza una entrada de resolución completa para extraer las características de los detalles finos de la segmentación.

Redes encoder-decoder con saltos de conexión: este tipo de arquitecturas se proponen en [22, 44] y consisten en saltos de conexiones que se presentan como una solución al problema de desvanecimiento de gradiente que se encuentra en algunas redes con muchas capas de profundidad. Esto se debe a que al hacer la propagación hacia atrás tantas veces como se vio en la ecuación (2.8), la función de pérdida tiende a hacerse cero ya que son multiplicaciones por números de 0 a 1. Por ello, la solución es saltarse algunas capas para generar nuevas rutas y reducir dicho desvanecimiento.

A diferencia de las redes con saltos de conexión anteriores, Fast-SCNN utiliza saltos para compartir los cálculos de características de las primeras capas entre las dos ramas de este modelo como se puede apreciar en la Figura 3.1.

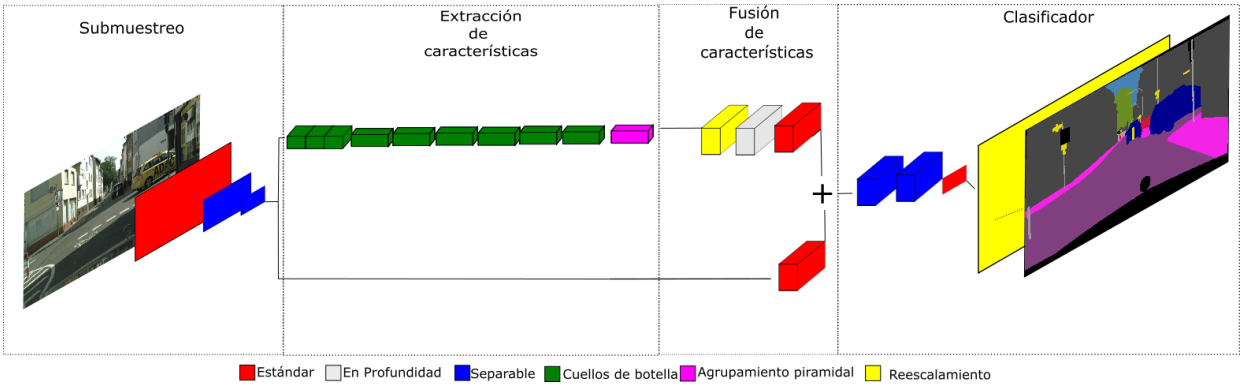

Figura 3.1 Arquitectura de Fast-SCNN.

Con estas bases, Fast-SCNN utiliza un módulo de submuestreo, un módulo de extracción de características globales, un módulo de fusión de características y un módulo de clasificación estándar. Esta arquitectura se puede apreciar en la Tabla 3.1, donde el bloque Conv2D es la convolución tradicional (ecuación (2.1)), el bloque DSConv es una convolución separable en profundidad (ecuación (2.21)), DWConv es una convolución separable en profundidad sin la convolución por punto del final (ecuación (2.20)), t factor de expansión del cuello de botella que representa las veces que va a expandirse los canales de entrada, c_o es el número de canales de salida, n las veces que se repite la operación y s es el *stride* o salto que tiene que dar la convolución.

Tabla 3.1 Arquitectura de Fast-SCNN

Entrada	Operación	t	c_o	n	s
1024x2048x3	Conv2D1 (2.1)	-	32	1	2
512x1024x32	DSConv1 (2.21)	-	48	1	2
256x512x48	DSConv2 (2.21)	-	64	1	2
128x256x64	Cuello de botella (BTN) Tabla 3.2	6	64	3	2
64x128x64	Cuello de botella (BTN)	6	96	3	2

	Tabla 3.2				
32x64x96	Cuello de botella (BTN) Tabla 3.2	6	128	3	1
32x64x128	PPM	-	128	-	-
32x64x128	Fusión de características	-	128	-	-
128x256x128	DSConv3 (2.21)	-	128	2	1
128x256x128	Conv2D2 (2.1)	-	19	1	1

Módulo de submuestreo: este módulo emplea 3 capas usadas para asegurar que el intercambio de características de bajo nivel sea válido y eficientemente implementado. La primera capa Conv2D1 es una convolución tradicional vista en la ecuación (2.1) y las últimas dos DSConv1 y DSConv2 son convoluciones separables en profundidad mencionado en la ecuación (2.21). Cabe aclarar que, aunque la convolución separable en profundidad sea más eficiente, se utiliza una convolución estándar en la primera capa ya que al tener muy pocos canales de entrada, no se beneficia tanto al utilizar una convolución separable en profundidad (DWS). Todas las capas usan $s = 2$ lo que significa que sería una convolución como la mostrada en la ecuación (2.21) solo que el paso de W se hace cada dos pixeles en lugar de 1. Las capas Conv2D1, DSConv1 y DSConv2 tienen normalización por lotes y utilizan la función de activación ReLU definida en la ecuación (2.2). El tamaño de los *kernels* de las capas del módulo de submuestreo es de 3x3.

Extractor de características globales: este módulo busca capturar el contexto global para la segmentación semántica, tomando la salida directa del módulo de submuestreo que es 1/8 de la resolución de la imagen original. Luego, se utilizan bloques eficientes de cuello de botella residual propuestos en MobileNet v2 [54] cuyas operaciones están dadas por la Tabla 3.2 donde m y n es el tamaño del bloque. Estos bloques de cuello de botella utilizan las DWS dadas por la ecuación (2.21) resultando en menor número de parámetros y operaciones de punto flotante. Finalmente, se añade un módulo de agrupamiento piramidal [51] para agregar información contextual basada en diferentes regiones.

Tabla 3.2 Bloque de cuello de botella residual [54]

Entrada	Operación	Salida
$\boldsymbol{m \times n \times Ci}$	1x1 conv2d y ReLU6	$m \times n \times tCi$
$\boldsymbol{m \times n \times tCi}$	3x3 DWS con s y ReLU6	$\frac{m}{s} \times \frac{n}{s} \times tCi$
$\boldsymbol{\frac{m}{s} \times \frac{n}{s} \times Ci}$	1x1 conv2d	$\frac{m}{s} \times \frac{n}{s} \times Ci'$

ReLU6 es una variante de ReLU en la que los valores de la cota superior de la función de activación se van a saturar a 6 el valor de esta función de activación. De esta manera, la ReLU6 está dada por:

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \; y \; x \leq 6 \\ 6, & x > 6 \end{cases} \tag{3.1}$$

Fusión de características: de manera similar a ICNet [55] y ContextNet [52], se suman las características para asegurar la eficiencia computacional. Si DSConv2 es la salida del módulo de aprendizaje a submuestrear y PPM la salida del módulo de extracción de características globales entonces esta suma está dada por:

$$FFM = PPM + DSConv2 \tag{3.2}$$

Esta operación es más eficiente ya que solo requiere de sumas, y a diferencia de otros métodos de fusión de características como los métodos de concatenación, no agrega más profundidad a las capas posteriores.

Clasificador: en esta etapa se utilizan dos DWS y una convolución por punto al final, definida en (2.20) pero con $d = 1$. Se realizó de esta manera ya que se encontró en [53] que añadiendo pocas capas después de la fusión de características se mejora la precisión.

Durante el entrenamiento se utiliza una función de activación *softmax* para normalizar la salida dentro de una distribución de probabilidad $\sigma: \mathbb{R}^K \longrightarrow [0,1]^K$ y está dada por la ecuación (3.3), ya que se emplea para el gradiente descendiente. Durante la inferencia, la función *softmax* se sustituye con una función más eficiente llamada *argmax* descrita por la ecuación (3.3), donde

la función $f: X \rightarrow Y$ y el argumento máximo se mapean sobre un subconjunto S de X que consiste en seleccionar que índice del vector contiene el valor mayor.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}} \; para \; i = 1, \dots, K \; y \; z = (z_1, \dots, z_K) \in \mathbb{R}^K \tag{3.3}$$

$$\underset{x \in S \subseteq X}{argmax} \, f(x) \coloneqq \{x | \mathrm{x} \in S \wedge \forall y \in S : f(y) \leq f(x)\} \tag{3.4}$$

Ambas funciones son monotónicamente incrementales, pero argmax tiene un tiempo de procesamiento de 0.03 milisegundos (calculado con la librería time en Python) menos que softmax para un vector de 3 parámetros.

3.2 Mininet-V2

Esta arquitectura presenta algunas mejoras respecto a su primera versión descrita en [56] y utiliza las redes multi-dilatadas y una rama extra que sirve para obtener información de detalles finos como se puede apreciar en la Figura 3.2, en lugar de utilizar conexiones saltadas.

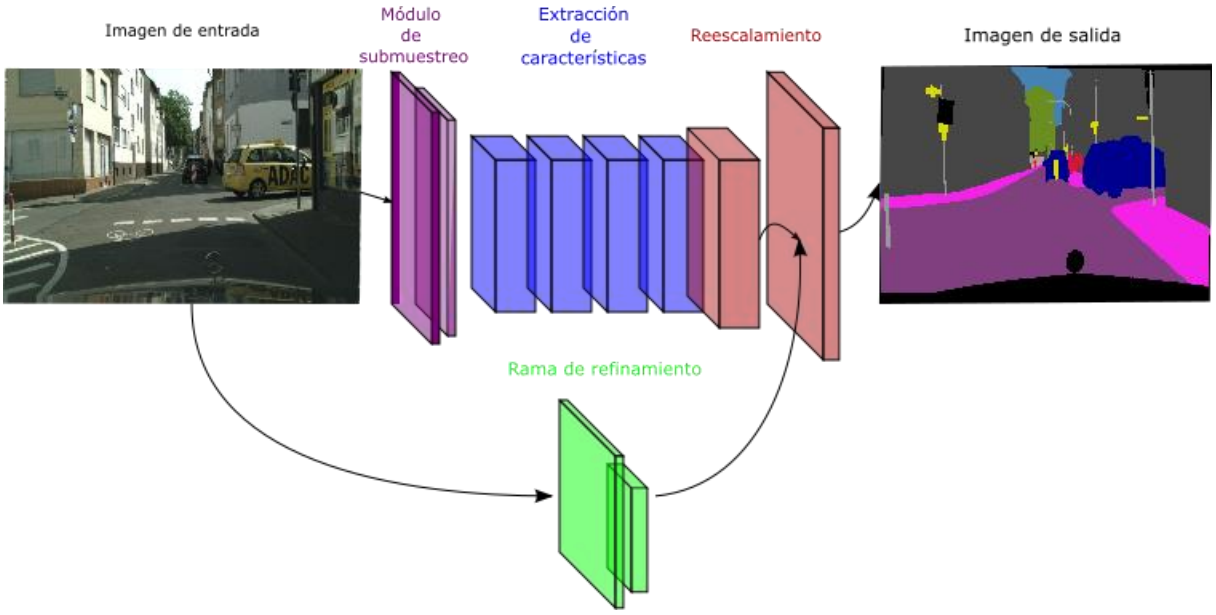

Figura 3.2 Esquema general de la arquitectura de Mininet-V2.

Los bloques principales de esta arquitectura están descritos por la Tabla 3.3. El componente principal de Mininet-V2 es el módulo convolucional, que consiste en DWS de 3x3 seguidos de una conexión residual dada por la ecuación (3.5), donde básicamente se tienen dos DWS

definidas en la ecuación (2.20) con un factor $r > 1$ y otra DWS dada por (2.20) con $d = 1$ que es una convolución por punto o de 1x1. Si el módulo convolucional propuesto se aplica en el bloque extractor de características, se agrega un *dropout* de 0.25 después de la convolución.

$$G_{k,l,C_o} = G'_{k,l,C_o} + G^r_{k,l,C_o} \tag{3.5}$$

Tabla 3.3a Primera parte de la arquitectura de Mininet-V2

Bloque	Capa	Tipo	Entrada	Salida
Submuestreo	d1	submuestreo	Imagen	512x256x16
	d2	submuestreo	d1	256x128x64
	m1	r=1	d2	256x128x64
	m2	r=1	m1	256x128x64
	m3	r=1	m2	256x128x64
	m4	r=1	m3	256x128x64
	m5	r=1	m4	256x128x64
	m6	r=1	m5	256x128x64
	m7	r=1	m6	256x128x64
	m8	r=1	m7	256x128x64
	m9	r=1	m8	256x128x64
	m10	r=1	m9	256x128x64
	d3	submuestreo	m10	128x64x128
Extracción de características	m10	r=1	d3	128x64x128
	m11	r=2	m10	128x64x128
	m12	r=1	m11	128x64x128
	m13	r=4	m12	128x64x128
	m14	r=1	m13	128x64x128
	m15	r=8	m14	128x64x128
	m16	r=1	m15	128x64x128
	m17	r=16	m16	128x64x128
	m18	r=1	m17	128x64x128
	m19	r=1	m18	128x64x128
	m20	r=1	m19	128x64x128
	m21	r=2	m20	128x64x128
	m22	r=1	m21	128x64x128
	m23	r=4	m22	128x64x128
	m24	r=1	m23	128x64x128
	m25	r=8	m24	128x64x128

Tabla 3.4b Segunda parte de la arquitectura de Mininet-V2

Bloque	Capa	Tipo	Entrada	Salida
Ref	d4	submuestreo	Imagen	512x256x16
	d5	submuestreo	d4	256x128x64
Reescalamiento	up1	reescalamiento	m25	256x128x64
	m26	r=1	up1+d5	256x128x64
	m27	r=1	m26	256x128x64
	m28	r=1	m27	256x128x64
	m29	r=1	m28	256x128x64
	salida	reescalamiento	m29	512x256xN

En Mininet-V2 existen operaciones de submuestreo compuestas por operaciones de agrupación por máximos definido en la ecuación (2.3) concatenadas con convoluciones con pasos (*strided*). Además, todos los módulos de reescalamiento son convoluciones traspuestas como muestra la ecuación (2.4). Cada bloque cumple una función distinta y se describen a continuación:

Bloque de submuestreo: este bloque hace el submuestreo de las características de manera eficiente, combinando técnicas de submuestreo como la agrupación por máximos visto en la ecuación (2.3), con módulos convolucionales estándares mencionados en la ecuación (2.1).

Bloque de extracción de características: este bloque es la parte principal del encoder. Consiste en una cantidad considerable de módulos de convolución con diferentes factores de dilatación como se muestra en la Figura 2.4, que es una reinterpretación de la ecuación (2.21), pero el *kernel* W se modifica para tener una dimensión $d + 2r - 2$ por lado para expandirlo sin agregar más parámetros.

Bloque de refinamiento (ref): este bloque extrae características espaciales de alta resolución realizando dos operaciones de submuestreo a la imagen de entrada. La meta de este bloque es extraer características adicionales que puedan ayudar con los detalles finos en la segmentación. En este bloque entra la imagen $I(x, y)^{RGB}$ a una operación de agrupación por máximos descrita en la ecuación (2.3) concatenadas con convoluciones con pasos *(strided).*

Después la salida de este bloque llamado d4 repite las operaciones anteriores de agrupación y convolución. La salida de este bloque se llama d5.

Bloque de reescalamiento: este bloque reescala mediante convoluciones traspuestas como se muestra en la ecuación (2.4) la salida del bloque de extracción de características. Después, se suma con el bloque de refinamiento de la capa d5 (ver Tabla 3.3). La última parte de este bloque consiste en varios módulos de convolución sin factor de dilatación y operaciones de reescalamiento. La resolución de salida es la mitad que la de entrada.

Esta arquitectura cuenta con una versión más pequeña llamada Mininet-V2-CPU enfocada para aplicaciones de CPU. Esta arquitectura es similar a la anterior solo que se le removieron las capas desde m3 a m10, m16 a m25 y de m28 a m29 para reducir el tiempo de inferencia.

3.3 ESPNetV2

Mehta [47] presenta un modelo novedoso que es computacionalmente eficiente y de bajo consumo de energía. Esto se logra gracias a las ventajas de las DWS mencionadas en la ecuación (2.21) y las convoluciones por punto dadas por la ecuación (2.20) con $d = 1$. Además de esto, se propone una nueva capa basada en DWS llamado *Extremely Efficient Spatial Pyramid of Depth-wise Dilated Separable Convolutions* (EESP), que está especialmente diseñado para sistemas embebidos y dispositivos de computación en el borde.

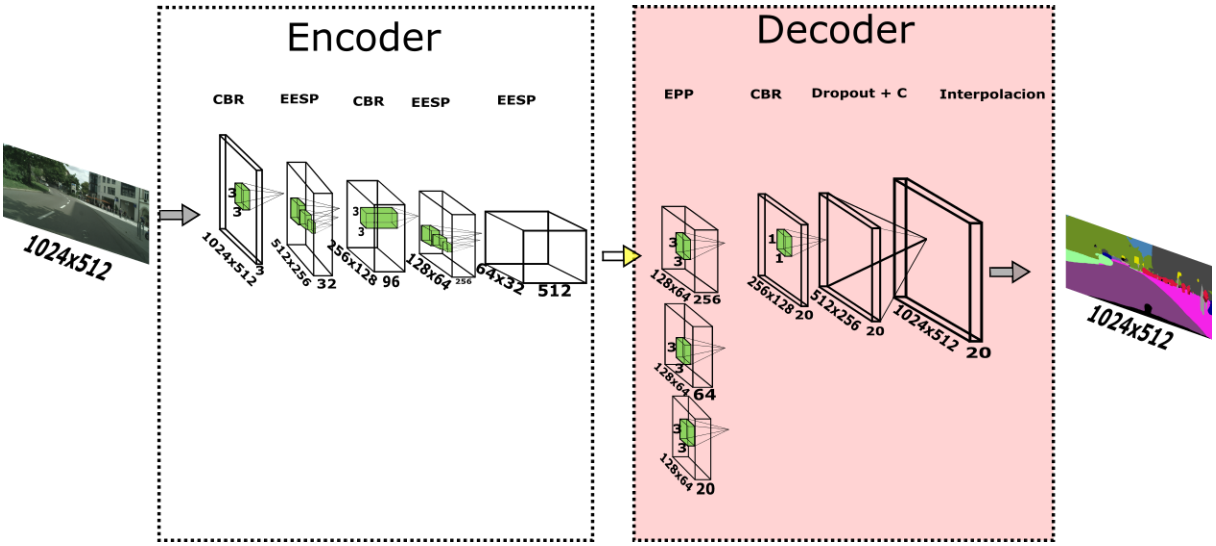

Figura 3.3 Arquitectura de EESPNetV2.

Como se observa en la Figura 3.3 EESPNetV2 está compuesta por una serie de módulos conocidos como CBR y EESP. El CBR es un conjunto de tres operaciones. La primera es una convolución dada por la ecuación (2.4), la segunda es una normalización por lotes (batch normalization) vista en [57] y finaliza con una función de activación similar a la ReLU vista en la ecuación (2.2). También la arquitectura de EESPNetV2 repite las capas EESP varias veces para incrementar la profundidad de la red y encontrar más características. En estas unidades se utiliza normalización por lotes y función de activación PreLU definida en [58] después de cada capa de convolución. Sin embargo, en la última capa de convolución por grupo se aplica una función de activación PreLU después de la suma por elementos. Para mantener la misma complejidad computacional a cada nivel espacial, los mapas de características se duplican después de cada operación de submuestreo.

La capa EESP que se muestra en la Figura 3.3, está inspirada por el módulo *Efficient spatial pyramid* (ESP) de la versión anterior de ESPNet [46] que se muestra en la Figura 3.4 y se basa en una estrategia de reducir, separar, transformar y unir. Es decir, la capa ESP primero proyecta los mapas de características de entrada a un nuevo espacio dimensional con una convolución por punto vista en la ecuación (2.1). Después, aprende las características en paralelo usando convoluciones dilatadas con diferentes factores r desde 1 hasta B, donde B es el número de ramas que tendrá este módulo de capas convolucionales. $ER = \frac{C_i}{B}$ son los canales de entrada para cada convolución, DConv2d-d es una convolución estándar con r >1 y d es el tamaño del *kernel,* los parámetros entre paréntesis que se muestran en la Figura 3.4 se distribuyen de la siguiente manera: el primer parámetro será los canales de entrada, el segundo los canales de salida y el tercero será r ej. (Ci, Co, r).

Diferentes factores de dilatación en cada rama permiten a la unidad ESP aprender las características de un campo receptivo. Esta factorización aprende especialmente las características en un espacio de baja dimensión, permitiendo a la unidad ESP ser más eficiente.

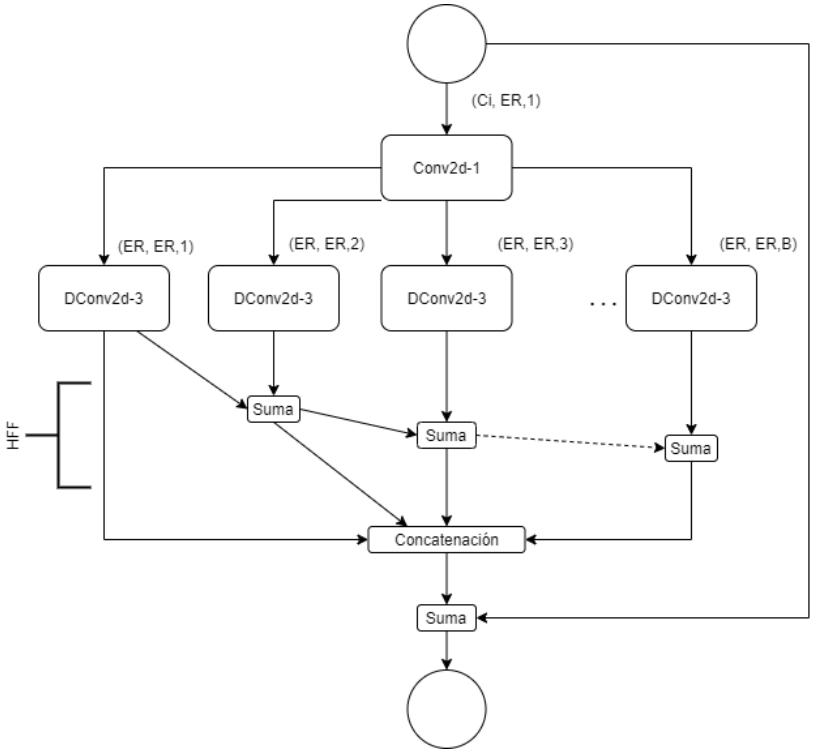

Figura 3.4 Capa ESP

Para hacer este módulo aún más eficiente computacionalmente, se remplaza las convoluciones por punto por convoluciones de grupo por punto (GConv2d), las cuales presentan las mismas características que en la ecuación (2.1) con $d = 1$ y $Ci = Co$. Después, se remplazan las costosas convoluciones estándar de 3x3 por DWS dilatadas (DDWS), que combinan las presentadas en la ecuación (2.21) con la teoría de la Figura 2.4 del capítulo anterior. Para remover los artefactos de cuadrícula causados por las convoluciones dilatadas, se fusionan los mapas de características usando un método de fusión jerárquica (HFF) [46] donde se suman las DDWS de la rama actual más la rama anterior para B=2 y para B>2 y se suma la DDWS de la rama actual más la suma de las anteriores. Este método fusiona los mapas de características de manera aditiva usando las convoluciones dilatadas de manera jerárquica. Es decir, los mapas de características de la rama con un menor campo receptivo se combinan con la rama con el siguiente campo receptivo más alto de cada nivel de jerarquía. La unidad resultante se puede ver en la Figura 3.5.

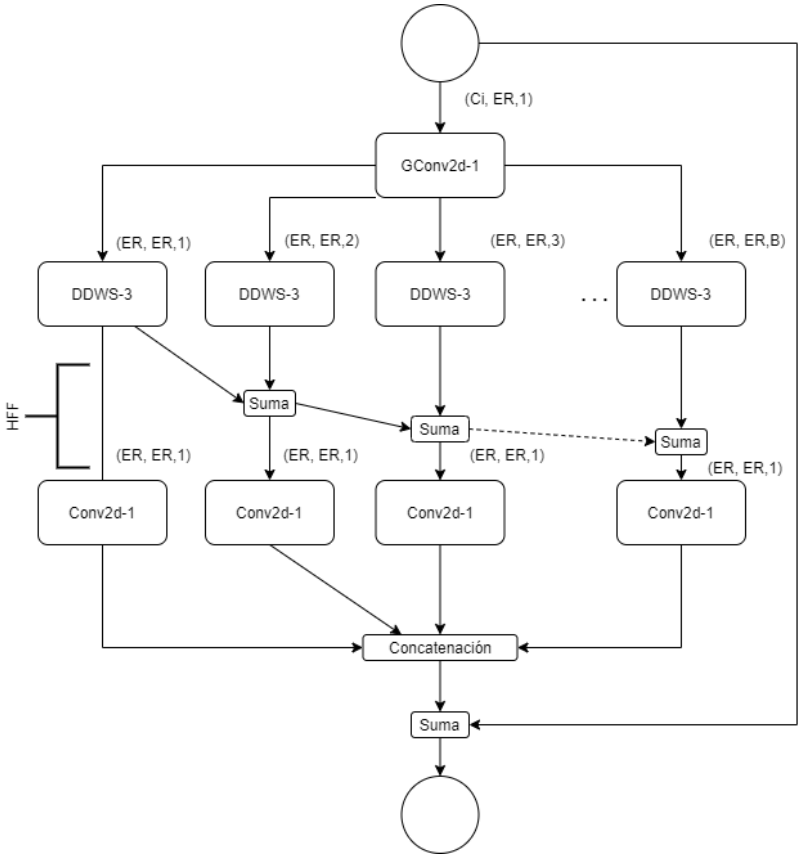

Figura 3.5 Primera versión del módulo EESP.

Se notó en [47] que calcular B número de convoluciones por punto en la Figura 3.5 de manera independiente, es equivalente a una sola convolución de grupo por punto con B grupos en términos de complejidad. Sin embargo, las convoluciones de grupo por punto son más eficientes en términos de implementación, porque maneja solo un *kernel* de convolución en lugar de B *kernels*. Por lo tanto, se intercambiaron las B convoluciones por punto por una sola convolución grupal por punto como se muestra en la Figura 3.6, esta es la unidad EESP.

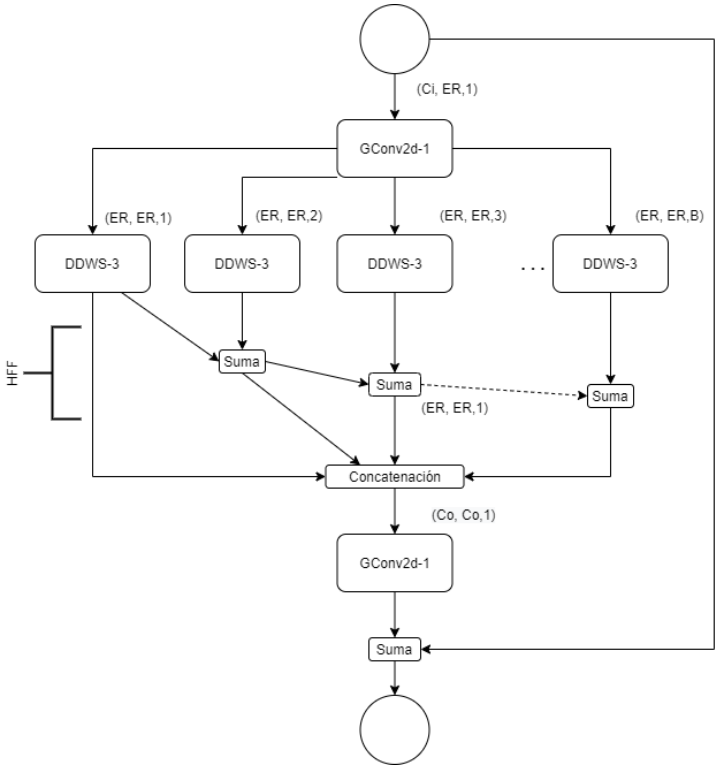

Figura 3.6 Módulo EESP

Para aprender características eficientes a múltiples escalas, se hicieron algunas modificaciones al bloque EESP en la Figura 3.6. Se sustituyeron las DWS dilatadas con sus contrapartes que tienen salto. También, se añadió una operación de agrupamiento por promedio vista en ecuación (2.3) en lugar de una conexión de identidad y una operación de concatenación que reemplaza la suma por elementos, lo cual ayuda a expandir las dimensiones de los mapas de características de manera eficiente. Dicha operación de concatenación consiste en agrupar los mapas de características, uno tras otro en un nuevo mapa de características de mayor dimensión. Para codificar de mejor manera las relaciones espaciales y aprender las características de manera eficiente, se añadió un atajo desde la imagen de entrada y la unidad actual de submuestreo mediante una conexión eficiente de amplio rango (línea roja de la Figura 3.7). Esta conexión primero submuestrea la imagen al mismo tamaño que tiene los mapas de características y después aprende las características usando dos convoluciones. La primera convolución es una convolución estándar de 3x3 que aprende las características espaciales mientras que la segunda es una convolución por punto que aprende las combinaciones lineales

entre la entrada y las proyecta a un espacio de alta dimensión. La unidad EESP con el atajo de conexión con rango amplio se pude apreciar en la Figura 3.7.

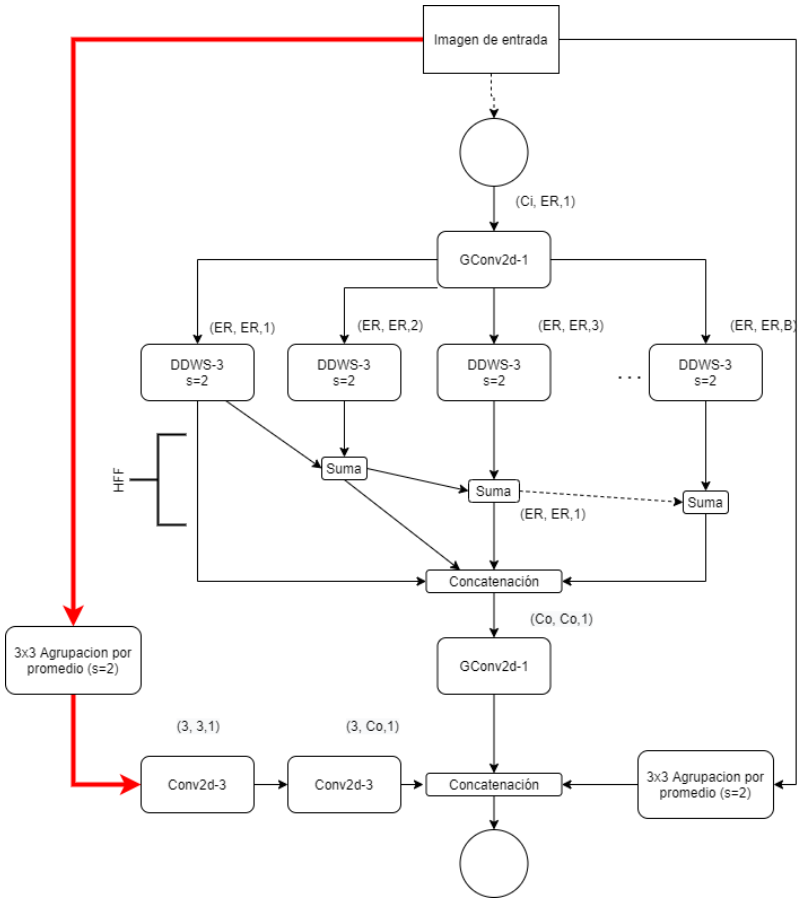

Figura 3.7 Versión de EESP con *stride*

A diferencia de las redes presentadas anteriormente, EESPNetV2 fue diseñada para diferentes aplicaciones, por lo que no cuenta con bloques tan definidos. Sin embargo, por la estructura que presentan se pueden extraer dos grandes bloques como se muestra en la Figura 3.3, que son el encoder y el decoder.

Encoder: es la parte de la red que se encarga de la extracción de características, presenta una sucesión de bloques CBR y EESP.

Decoder: en este bloque se realiza la reescalación del mapa de características resultante del bloque anterior, empieza con un módulo de agrupamiento piramidal eficiente (EPP) que se puede apreciar en la Figura 3.8.

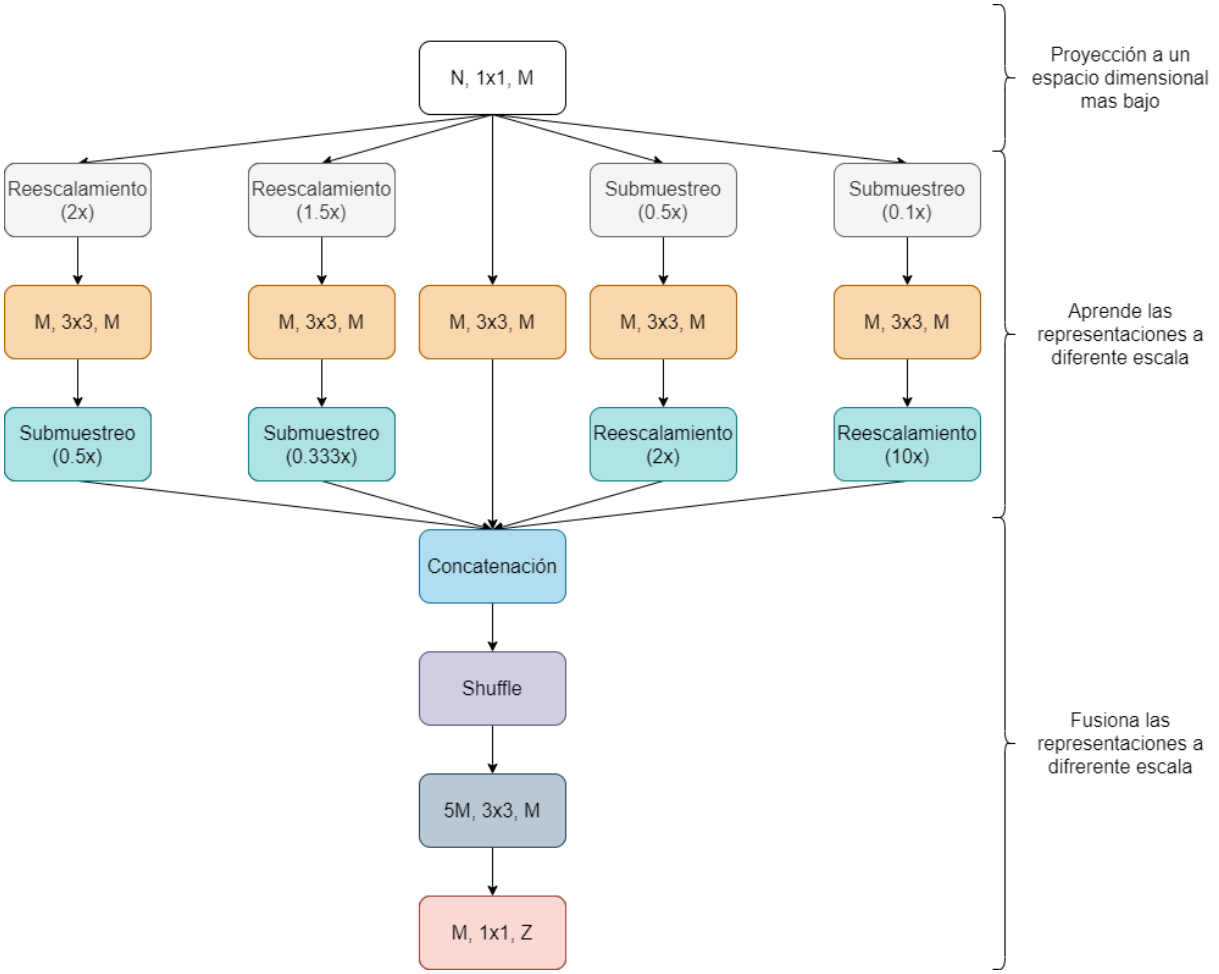

Figura 3.8 Módulo EPP.

Este módulo se diseñó para que EESPNetV2 aprendiera las características invariantes a la escala de manera eficiente. Para lograrlo primero proyecta los mapas de características de N-Dimensiones a un espacio de M-Dimensiones (N>>M) para luego aprender las características a diferentes escalas usando DWS vistas en la ecuación (2.21). Asumiendo que se tienen B ramas, se concatenan la salida de estas para producir una salida en un espacio de BM-dimensiones. Para facilitar el aprendizaje de las características entre escalas, se realiza la operación de *shuffle* definida en [59] que consiste en realizar la traspuesta de la matriz de entrada y reacomodarla para que no pierda las dimensiones iniciales, el algoritmo se puede describir como lo siguiente:

1. Reacomodo de entrada $G_{k,l,bM}$ k y l siendo las dimensiones del mapa de características de entrada.
2. Se obtiene ${G^T}_{k,l,bM}$ que es igual a $G_{l,k,bM}$ lo que significa que se cambian columnas por filas en las matrices a través de todos los mapas de características.
3. Se reacomoda la matriz $G_{k,l,bM}$ para tener las mismas dimensiones iniciales de cada mapa de características.

Después de la operación de *shuffle* se fusionan las características mediante una convolución de grupo (ecuación (2.21) pero *Ci=Co*) y finaliza con una convolución por punto dada por la ecuación (2.20) con d=1.

Luego de aplicar el bloque EPP se agrega un bloque CBR seguido de una capa de dropout, para finalizar con una capa de reescalamiento con una interpolación bilineal definida en [1].

3.4 CARTNet

A partir de los modelos Fast-SCNN, Mininet-V2 y ESPNetV2 se considera que para lograr una eficiencia computacional y un buen desempeño está demostrado que:

- El tipo de convolución base deben ser las convoluciones separables en profundidad ya que son las que presentan menor tiempo de procesamiento.
- Se debe considerar el contexto global y local de una imagen para lograr un mejor entendimiento de la escena a segmentar.

Es decir, con la red a proponer se busca implementar la idea principal que comparten Fast-SCNN y Mininet-V2, que es preservar la relación entre el contexto global y local que se presenta en las redes convolucionales. Esto es similar al sistema de percepción humano, ya que relaciona los objetos con el entorno que los rodea cuando los sujetos buscan objetivos dentro de una escena desconocida, lo cual se explica a continuación.

El cerebro procesa los objetos mediante los patrones multivoxel en el área occipital lateral, el área medial de la corteza visual y el surco intraparietal. Estas áreas codifican el lado de la escena donde es más probable que aparezca el objetivo [60]. En escenarios conocidos, por otra parte, el sistema de percepción humano trabaja usando el paradigma de las señales contextuales

y se sugiere que el hipocampo es crucial para guiar la búsqueda de objetos que aparecen consistentemente en la misma posición [61, 62]. La información de la escena también puede influir en el reconocimiento de objetos [63, 64] especialmente cuando la percepción del objeto no es clara [65]. En tales casos, las señales contextuales del área parahipocampal y el área medial (MPA) pueden trabajar en conjunto con las señales descendentes de la corteza prefrontal medial para realizar el reconocimiento de objetos [66, 67]. Estas señales contextuales pueden proporcionar información, que objetos se encuentran normalmente en una escena determinada y su ubicación más probable.

Para imitar estos mecanismos de reconocimiento de objetos en el sistema de percepción humano, se buscó diseñar una FCN que preserve la relación del contexto espacial y global. Para ello, se decidió utilizar los siguientes elementos:

- Fast-SCNN [53] basa su funcionamiento en tener ramas con diferentes niveles de resolución. Como ya se explicó en la sección 3.1, se tiene un nivel a resolución completa que sirve para obtener características de contexto local y ayudar con los detalles finos de la segmentación, mientras que la otra rama con una resolución menor ayuda a obtener la información del contexto global. Esta es la principal base de la estructura general de CARTNet.

- La solución de MiniNet-V2 [36] fue proponer las convoluciones multi-dilatadas que permiten obtener características adicionales que ayuden con los detalles finos en la segmentación. Estas convoluciones fueron las utilizadas para la etapa de extracción de características de CARTNet.

- Para finalizar, en ESPNetV2 [47] se propuso un bloque convolucional EESP similar a las convoluciones multi-dilatadas ya que también cuenta con diferentes niveles de dilatación. La diferencia principal es que inicia con una convolución de grupo que se concatena representada por la ecuación (2.21) pero con $C_i = C_o$. Además, en las convoluciones multi-dilatadas se utiliza una convolución representada en la ecuación (2.20) con $d = 1$ de 1x1 para proyectar los diferentes niveles de dilatación a un nuevo espacio de características.

Basado en las ideas anteriores, se realizó una combinación entre Fast-SCNN y Mininet-V2, que se puede ver en la Figura 3.9. Se basó principalmente en la estructura general de Fast-SCNN, sustituyendo las convoluciones de cuello de botella por las multi-dilatadas propuestas en Mininet-V2. En la Tabla 3.4 se puede apreciar el desglose de la arquitectura de CARTNet.

Similar a como está distribuido el modelo Fast-SCNN, CARTNet cuenta con cuatro grandes bloques: submuestreo, extracción de características, fusión y reescalamiento. A continuación, se describen los bloques.

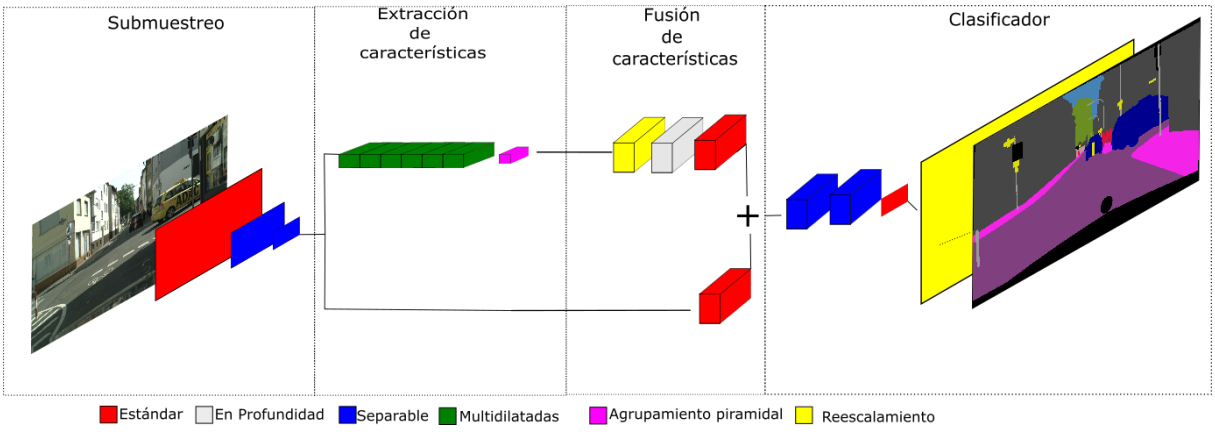

Figura 3.9 Arquitectura del modelo CARTNet

Tabla 3.5 Desglose por operación de CARTNet.

Bloque	*Tipo de operación (ecuación)*	*Nombre*	*Entrada*	*s*	*r*	*Ci*	*Co*
Submuestreo	Convolución Estándar (2.1)	Conv2D1	Imagen	2	0	3	32
	Convolución separable (2.20)	DSConv1	Conv2D1	2	0	32	48
	Convolución separable (2.20)	DSConv2	DSConv1	2	0	48	64
Extracción de características	Multi-dilatada (3.4) y (2.20 con d=1)	MD1	DSConv2	1	2	64	96
	Multi-dilatada (3.4) y (2.20 con d=1)	MD2	MD1	1	4	96	128
	Multi-dilatada (3.4) y (2.20 con d=1)	MD3	MD2	1	2	128	128
	Multi-dilatada (3.4) y (2.20 con d=1)	MD4	MD3	1	4	128	128

Fusión de características	Multi-dilatada (3.4) y (2.20 con d=1)	MD5	MD4	1	8	128	128
	Multi-dilatada (3.4) y (2.20 con d=1)	MD6	MD5	1	16	128	128
	Agrupamiento piramidal	PPM	MD6	-	-	128	128
	Reescalamiento	BIL	PPM	-	-	128	128
	DWS (2.21)	DWS1	BIL	-	-	128	128
Reescalamiento	Convolución Estándar (2.1)	Conv2D2	DWS1	-	-	128	128
	Convolución Estándar (2.1)	Conv2D3	Conv2D2	-	-	128	128
	suma	FFM	Conv2D2+Conv2D3	-	-	128	128
	Convolución separable (2.20)	DSConv5	FFM	1	-	128	128
	Convolución separable (2.20)	DSConv6	DSconv5	1	-	128	128
	Convolución Estándar (2.1)	Conv2D4	DSCov6	1	-	128	19

Submuestreo: se lleva a cabo con tres capas, donde la imagen de entrada $I(x, y)^{RGB}$ entra a una convolución estándar vista en la ecuación (2.1) (Conv2D1) ya que al tener solo tres canales de entrada no se beneficia tanto de las DWS. Luego, se encuentran dos convoluciones separables dadas por la ecuación (2.21) (DSConv1 y DSConv2), cada uno con s=2 lo cual entrega una salida a 1/8 de resolución de la imagen de entrada.

Extracción de características: este bloque representa la parte más profunda del modelo, contando con seis convoluciones multi-dilatadas (MD1-MD6) que tienen la tarea de extraer las características más abstractas de la red. Fueron seis capas ya que, si se agregaba una convolución más con índice de dilatación mayor, el tamaño del *kernel* excedía las dimensiones del mapa de características de entrada. En este caso particular, r solo representa la dilatación de G' en la ecuación (3.4). Este bloque finaliza con un bloque de agrupación piramidal [51].

Fusión: esta etapa consiste inicialmente en un reescalamiento de la última capa del bloque de extracción (PPM) [51] mediante interpolación bilineal (BIL) [1]. Luego, se tiene una DWS como se ve en la ecuación (2.21) (DWS1). Después esa capa DWS1 entra a una convolución estándar (Conv2D2) vista en ecuación (2.1). Luego, a la salida del bloque de submuestreo se le

aplica otra convolución estándar (Conv2D3) vista en ecuación (2.1), al final se realiza una suma por elementos de Conv2D2+Conv2D3.

Reescalamiento: esta parte final se encarga de regresar a la resolución original de la imagen mediante 3 principales convoluciones y una interpolación bilineal. La primera y la segunda son convoluciones separables (DSConv3 y DSConv4) como se ve en la ecuación (2.20), mientras que la capa final es una convolución estándar (Conv2D4) como se ve en la ecuación (2.1) con 19 canales de salida ya que es el número de clases que tiene el dataset de cityscapes.

CAPÍTULO 4
Resultados y conclusiones

Este capítulo presenta los diversos experimentos que se llevaron a cabo para obtener los resultados del desempeño del modelo CARTNet y las conclusiones de la tesis. Los experimentos para entrenar y probar se llevaron a cabo con la base de datos Cityscapes [28]. El entrenamiento se realizó con un equipo de sobremesa con un procesador Ryzen 7 1700, 16 GB de RAM y una GPU GTX 1070 de NVIDIA. El algoritmo que se utilizó para entrenar los modelos fue gradiente estocástico descendiente con momento de 0.9, tamaño de lote 4, factor de aprendizaje de 0.01 y decaimiento de pesos de 0.0001. Para las pruebas de evaluación se utilizó la GPU embebida Jetson TX2 de NVIDIA.

De esta manera, este capítulo se organiza de la siguiente forma: la sección 4.1 presenta los resultados cualitativos y cuantitativos del modelo propuesto. La sección 4.2 es una sección de discusión donde se desglosan los diversos experimentos realizados para llegar al desarrollo del modelo propuesto. Finalmente, la sección 4.3 muestra las conclusiones a las que se llegó con este trabajo, así como también plantea los trabajos a futuro.

4.1 Resultados del modelo propuesto

Para realizar una comparativa cuantitativa de los resultados de CARTNet en contra de las redes encontradas en la literatura se tomaron en cuenta los indicadores de FPS, FLOPs, parámetros y precisión (mIOU). En la tabla 4.1 se muestran dicha comparativa entre varios modelos de segmentación, en la cual se puede apreciar que CARTNet tiene el primer lugar en tiempo de inferencia con 26 cuadros por segundo, segundo en número de FLOPs con 1.25 gigas, cuarto en número de parámetros con 221, 827 y se obtuvo el 15vo lugar en cuanto a precisión con 52.12%.

En cuanto a FLOPS, se encuentra en segundo debido a que el primer lugar es una arquitectura de una sola rama. Aunque en número de parámetros no se haya conseguido el primer lugar, no está muy lejos de los primeros lugares y las redes por debajo tienen valores muy grandes.

Tabla 4.1 Comparativa de indicadores de desempeño entre distintos modelos de segmentación semántica eficientes.

Modelo	FPS	FLOPs	Parámetros	mIOU
CARTNet	**26.12 (1)**	1.25 G (2)	221.82 K (4)	52.12
FastSCNN	24.91	1.76 G	1.14 M	69.22
ContextNet	22.03	1.78 G	876.56 K	66.1
FPENet	12.83	1.58 G	**115.12 K (1)**	68
ESPNetV1	9.88	3.45 G	201.54 K (3)	60.3
DABNet	6.96	10.46 G	756.64 K	66.8
FSSNet	6.86	2.78 G	175.88 K (2)	58.8
EDANet	6.43	8.95 G	689.49 K	65.1
LinkNet	6.26	21.9 G	11.54 M	**76.4**
CGNet	5.59	7.01 G	496.31 K	59.7
ENet	4.77	4.35 G	360.42 K	58.3
LEDNet	4.13	11.51 G	917.39 K	69.2
ESNet	3.28	24.35 G	1.66 M	70.7
ERFNet	3.23	26.86 G	2.07 M	68
ESPNetV2	2.31	**0.597 G (1)**	1.8 M	59.1

No obstante, CARTNet no tiene buena precisión, lo que se traduce en que no detecta objetos muy pequeños y si bien esta condición no representa un problema de funcionalidad, podría ser una limitante dependiendo de la aplicación en que se va a implementar. Al respecto se observó que las clases en donde tiene CARTNet desempeño bajo, son aquellas que representan objetos irrelevantes en el escenario. Por ejemplo, si se observa la Figura 4.1 se puede apreciar que se logra distinguir las clases más críticas como lo son el camino, la acera, automóviles y personas.

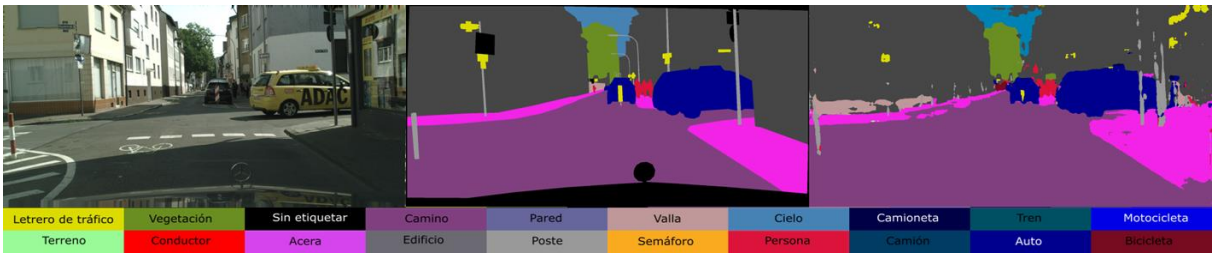

Figura 4.1 Frankfurt 294: imagen segmentada por CARTNet (derecha), el Ground-truth (centro) y la imagen original (izquierda)

Otro ejemplo está presente en la Figura 4.2, donde se aprecia que las clases con las que CARTNet tuvo más problemas son las de ‘cielo’ y ‘poste’. La clase ‘cielo’ es poco relevante

ya que en el caso del manejo autónomo (aplicación para la cual fue diseñada la base de datos) no es necesario ubicar el cielo, tal como se menciona en [68]. Sin embargo, los vehículos, vegetación, edificios, aceras y personas los detecta correctamente.

Otra situación presente es el caso de la clase "sin etiquetar" que corresponde a un pequeño número de pixeles que se tendrían que ignorar por parte de la red (se observa en negro). Como se puede ver en las Figuras 4.1 a 4.4, CARTNet no detecta esa clase, disminuyéndose drásticamente el desempeño porque se generan múltiples falsos negativos de las clases 'calle' y 'banqueta'.

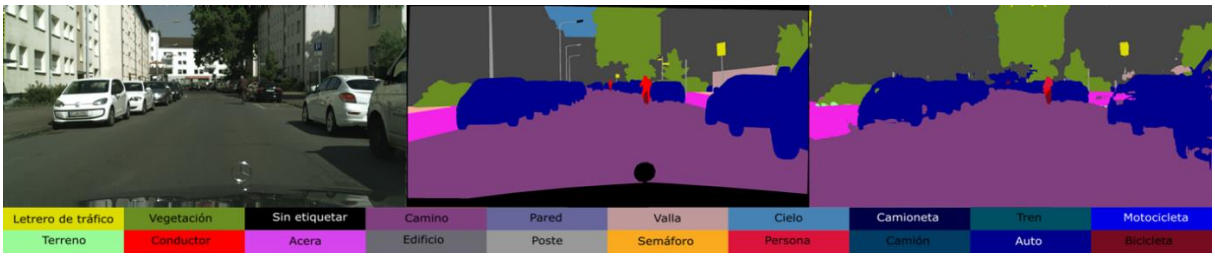

Figura 4.2 Frankfurt 576: imagen segmentada por CARTNet (derecha), el Ground-truth (centro) y la imagen original (izquierda)

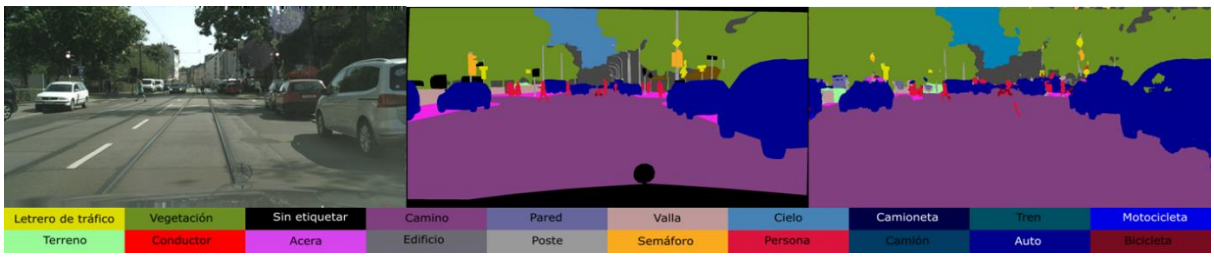

Figura 4.3 Frankfurt 1236: imagen segmentada por CARTNet (derecha), el Ground-truth (centro) y la imagen original (izquierda)

El peor escenario posible es donde se presentan estas dos problemáticas: una escena con clases de objetos muy finos como postes y grandes parches de clases sin etiquetar como se puede ver en la Figura 4.4. En este caso, existen clases con detalles finos como postes de semáforos y dos grandes parches sin etiquetar que en la imagen original que corresponden a botes de basura (en negro a la izquierda) y un puesto callejero de ropa (café a la derecha) que fueron clasificados con pixeles de automóvil.

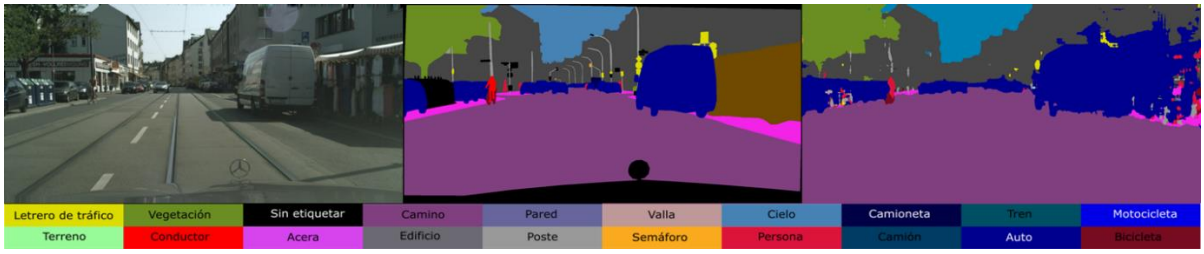

Figura 4.4 Frankfurt 1751: imagen segmentada por CARTNet (derecha), el Ground-truth (centro) y la imagen original (izquierda).

En la Figura 4.5 se muestra una comparativa entre diversas redes dentro del estado del arte, entre las cuales esta ENet [69], ERFNet [40], ESPNetV1 [46] y Mininet-V2 [36]. En esta figura se puede apreciar que a pesar de que CARTNet se encuentra bajo en cuanto a precisión, no existe mucha diferencia en los resultados de CARTNet y las demás redes. Nuevamente, donde más falla es en los detalles finos como puede ser en la segmentación de la acera en la segunda y tercera imagen, ya que se logran apreciar algunos cortes.

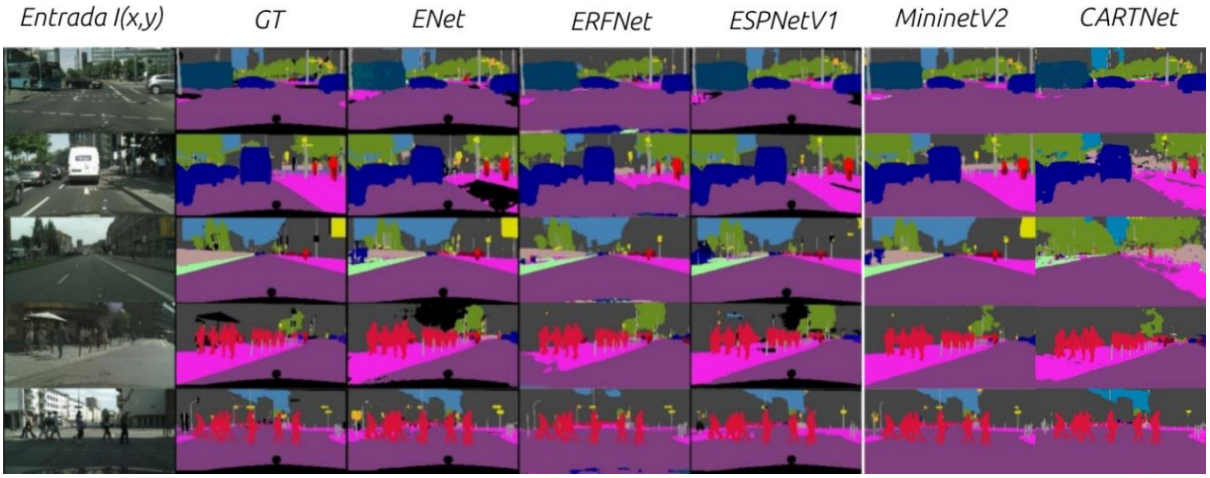

Figura 4.5 Comparativa cualitativo entre CARTNet y el estado del arte.

4.2 Discusión

Para entender el comportamiento que presenta CARTNet, se presentan los experimentos que se llevaron a cabo para llegar a la arquitectura final de esta red.

El primer experimento fue combinar Fast-SCNN con EESP para analizar el comportamiento de los bloques EESP. El segundo fue una versión temprana de CARTNet la cual se denominó CARTNet-300 que se realizó para probar la arquitectura con un entrenamiento con pocas

épocas. Después se realizó un análisis mediante los mapas de activación para saber internamente cómo se comportaba la red. Finalmente, se llevó a cabo un experimento en el que se modificó la manera de hacer el reescalamiento en la etapa final de la red.

4.2.1 Fast-SCNN con EESP

Como se mencionó en el capítulo anterior, las convoluciones multi-dilatadas tienen un funcionamiento muy similar al bloque EESP de ESPNetV2, por lo que se realizó un experimento que consistió en aprovechar la estructura que ya se tenía de Fast-SCNN y añadir el encoder de ESPNetV2 a la etapa de extracción de características, quedando una red experimental como lo define la Tabla 4.2, donde CBR significa convolución, normalización por lotes (Batch) y ReLU, s es el tamaño del *stride* o salto que tiene que dar la convolución, r el coeficiente de dilatación, Ci los canales de entrada y Co los canales de salida:

Tabla 4.2 Cambios en el bloque de extracción de características.

Bloque	*Tipo de operación*	*Nombre*	*Entrada*	*s*	*r*	*Ci*	*Co*
Extracción de características	EESP Fig 3.7	EESP1	DSConv2	1	1	64	64
	CBR Ec. 2.21 con ReLU	CBR	EESP1	2	1	64	96
	EESP	EESP2	CBR	1	1	96	128
	EESP	EESP3	EESP2	1	1	128	128
	Agrupamiento piramidal	PPM	EESP3	-	-	128	128

Este experimento tiene los mejores resultados en cuanto a velocidad con: FPS=34.18, parámetros=193,991 y tamaño en memoria= 466.91 MB y precisión de 46% con 300 épocas de entrenamiento. Esto derivó en una caída de 6% con respecto a CARTNet. Posteriormente, esta red experimental se entrenó con 1000 épocas y solo obtuvo 2% de mejora.

4.2.2 CARTNet-300

El primero de los experimentos para desarrollar CARTNet obtuvo una precisión de 45.60% y consistió en entrenar la red a 300 épocas con el optimizador de gradiente estocástico descendiente con momento de 0.9, tamaño de lote = 4 (limitado por la memoria de la tarjeta gráfica) decaimiento de pesos de 0.0001. A este experimento se le llamo CARNet-300. Debido

al bajo desempeño, se realizó un análisis de activaciones y las características que generaba para entender por qué tenía la precisión tan baja.

4.2.3 Análisis de los mapas de activación

En búsqueda de mejorar el resultado de CARTNet-300, se llevó a cabo un análisis de los mapas de activación. Para explicar este análisis, en la Figura 4.6 se puede ver un ejemplo de un mapa de activación de la primera capa del bloque de clasificación de CARTNet-300 (izquierda) y Fast-SCNN (derecha).

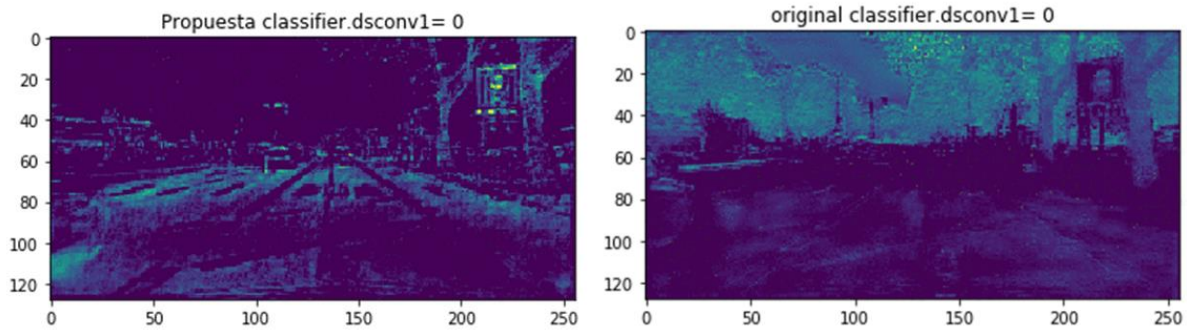

Figura 4.6 Ejemplo de mapas de activación de CARTNet-300 (izquierda) contra Fast-SCNN (derecha).

Del análisis de los mapas de activación se pudo interpretar que en algunos mapas de la red Fast-SCNN se podía discernir que el patrón presentado en los mapas de activación tiene relación con las clases del ground-truth, como es el caso de la Figura 4.6, en el que se puede distinguir que la atención de ese mapa de activación está en la parte superior, y se ve una clara división entre el cielo y los árboles, mientras que en la parte de CARTNet-300 se ve que la atención es en el centro pero sin alguna forma clara. Por ello, se llegó a la conclusión que se necesitaba entrenar más la red ya que da la impresión de que no está generalizando bien. Es decir, CARTNet encontraba características que no correspondían a la estructura de los objetos. Considerando estos resultados, se decidió realizar un experimento que consistió en aumentar las épocas. Este experimento se basó en que en el artículo de Fast-SCNN, se demostró empíricamente que al aumentar el número de épocas de entrenamiento aumentaba la precisión de la red similar al efecto que tiene el aumentar el tamaño de dataset. Por lo que se decidió finalmente entrenar CARTNet con un mayor número de épocas, lo que generó un aumento del 7% en precisión. Este resultado es el que se presentó en los resultados de la sección 4.1.

4.2.4 Métodos de reescalamiento

El siguiente experimento consistió en analizar diferentes métodos respecto al proceso de reescalamiento para poder seleccionar el método que presentara la mejor precisión sin perder eficiencia. Se empezó con utilizar el modelo de convolución traspuesta, lo que causo que la precisión cayera entre 30-35% aun y cuando se variaba 1 o 2 convoluciones dadas por la ecuación (2.4) seguidas del método de interpolación. Además, las convoluciones traspuestas presentaban artefactos de tablero de ajedrez, por lo que se deshecho esta alternativa. Además, de acuerdo con Wojna et al. [70] las convoluciones traspuestas tienen menos precisión en el reescalamiento durante el proceso de la segmentación semántica. Por lo tanto, se optó por dejar el método de reescalamiento final como bloques de convolución dados por la ecuación (2.21) seguidos de interpolación bilineal [1], que son los que tienen mejor relación entre precisión y tiempo de inferencia.

4.3 Conclusiones

En esta tesis se desarrolló un modelo de segmentación semántica que se denominó CARTNet y corre en tiempo real en un sistema embebido con GPU. CARTNet se enfoca en el contexto global y local en una imagen y se divide en 2 ramas de diferente resolución y 4 bloques con diferentes tareas. Estas tareas consisten en: submuestrear la imagen de entrada para enrutarla en las dos ramas, extraer características abstractas de los objetos con significado semántico, fusionar características de las dos ramas de diferente resolución y reescalar para obtener una salida de la misma resolución que la imagen de entrada.

Los resultados de este modelo fueron evaluados con la base de datos Cityscapes con imágenes de paisajes de distintas ciudades de Alemania tomadas desde el toldo de un automóvil. La resolución de las imágenes es de 1024x2048x3, y las pruebas se realizaron en una plataforma con GPU embebida Jetson TX2 de NVIDIA. De esta manera, CARTNet obtuvo el mejor desempeño de velocidad entre las FCN eficientes con 26.12 FPS, el segundo con menor número de operaciones de punto flotante con 1.25 G, el cuarto con menos parámetros 221.82 K y 52.12% de precisión mIOU. Esto significa que, si bien no se logró una precisión muy alta, la tarea principal de realizar segmentación semántica en tiempo real se logró, ya que CARTNet es un modelo compacto se puede correr de manera satisfactoria en sistemas embebidos.

Una de las ventajas que presenta este modelo con respecto a los demás, es que al procesar las imágenes en una cantidad de tiempo tan rápida y con una buena calidad, se crea un espacio de tiempo para la ejecución de otras tareas de procesamiento como puede ser la visualización o la toma de decisiones derivadas de la información obtenida, como podría ser en un coche autónomo donde se podría implementar un algoritmo de evasión de obstáculos o un robot manipulador que pueda clasificar objetos en una línea de producción.

Como trabajo a futuro, se plantea un entrenamiento basado en el esquema de maestro-estudiante ya que este permite un entrenamiento semi-supervisado en el que se pueden generar imágenes que no están presentes en la base de datos. Con este modelo de red maestro se puede generar *ground-truths.* Este modelo no se pudo trabajar en esta tesis ya que se requería realizar diversos experimentos en distintas GPUs que se encuentran en el Laboratorio PVR. No obstante, debido a las medidas tomadas por la pandemia del virus SARS-COV-2 no fue posible desarrollar estas pruebas.

CAPÍTULO 5 Referencias

[1] R. C. Gonzalez y R. E. Woods, Digital Image Processing, Upper Saddle River, New Jersey: Prentice Hall, 2001.

[2] M. I. Chacón Murguia, Procesamiento digital de imágenes, México, D.F.: Trillas, 2007.

[3] S. J. D. Prince, Computer Vision Models, Learning, and Inference, New York, New York, USA: Cambrigde University Press, 2012.

[4] L. Luo , D. Chen y D. Xue, «Retinal blood vessels semantic segmentation method based on modified U-Net,» de *Chinese Control And Decision Conference*, Shenyang, China, 2018.

[5] S. Heath, Embedded systems design, Burlington, Massachsetts: Elsevier cience, 2003.

[6] E. Ashford Lee y S. Arunkumar Seshia, Introduction to Embedded systems a Cyber-Physical Systems Approach, MIT Press, 2017.

[7] I. Sommerville, Software Engineering, Boston: Addison-Wesley, 2011.

[8] Z. Ju, J. Chen y J. Zhou, «Image segmentation based on the HSI color space and an improved mean shift,» de *IET International Conference on Information and Communications Technologies*, Beijing, China, 2013.

[9] L. Wentao, Y. Xiaocheng, H. Jiaqi, W. Long y X. Weiqiang, «Vehicle semantic detection on the highway via the moving platform,» de *10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics*, Shangai, China, 2017.

[10] X. Gao, M. Cai y J. Li, «Improved RGBD semantic segmentation using multi-scale features,» de *Chinese Control And Decision Conference (CCDC)*, Shenyang, China, 2018.

[11] R. Dong, X. Pan y F. Li, «DenseU-Net-Based Semantic Segmentation of Small Objects in Urban Remote Sensing Images,» *IEEE Access,* vol. 7, pp. 65347-65356, 2019.

[12] X. Zhang, Z. Chen, J. Wu, L. Cai, D. Lu y X. Li, «Fast Semantic Segmentation for Scene Perception,» *IEEE Transactions on Industrial Informatics,* vol. 15, nº 2, pp. 1183-1192, 2019.

[13] L. Tai, Y. Haoyang y M. Liu, «PCA-aided fully convolutional networks for semantic segmentation of multi-channel fMRI,» de *18th International Conference on Advanced Robotics (ICAR)*, Hong Kong, China, 2017.

[14] Y. Feng y L. Wang, «A Weakly-Supervised Approach for Semantic Segmentation,» de *IEEE 3rd Information Technology,Networking,Electronic and Automation Control Conference*, Chengdu, China, China, 2019.

[15] J. Kim, B. Choi y I.-S. Kweon, «OBJECT DETECTION USING HIERARCHICAL GRAPH-BASED SEGMENTATION,» de *IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, Canada, 2013.

[16] T.-R. Liu y S.-C. Chan, «A hierarchical semantic image labeling method via random forests,» de *TENCON, IEEE Region 10 International Conference*, Macao, China, 2015.

[17] J. Long, E. Shelhamer y T. Darrell, «Fully Convolutional Networks for Semantic Segmentation.,» *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.,* pp. 6810-6818, 2018.

[18] G. Lin, C. Shen, A. Van dan Hengel y I. Reid, «Efficient piecewise training of deep structured models for semantic segmentation,» de *IEEE Conf. Computer Vision and Pattern Recognition*, Las Vegas, Estados Unidos, 2016.

[19] V. Badrinarayanan, A. Kendall y R. Cipolla, «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, nº 12, pp. 2481-2495, 2017.

[20] F. Yu y V. Koltun, «Multi-Scale Context Aggregation by Dilated Convolutions,» de *International Conference on Learning Representations*, San Juan, Puerto Rico, 2016.

[21] Z. Wu, C. shen y A. Van den Hengel, «Wider or Deeper: Revisiting the ResNet Model for Visual Recognition,» 2016.

[22] E. Shelhamer, K. Rakelly, J. Hoffman y T. Darrell, «Clockwork Convnets for Video Semantic Segmentation,» *arXiv:1608.03609 ,* 2016.

[23] M. FAyyaz, M. H. Saffar, M. Sabokrou, M. Fathy, R. Klette y F. Huang , «STFCN: Spatio-Temporal FCN for Semantic Video Segmentation,» *arXiv:1608.05971 ,* 2016.

[24] D. Tran, L. Bourdev, R. Fergus, L. Torresani y M. Paluri, «Deep End2End Voxel2Voxel Prediction,» de *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Las vegas, Nevada, EEUU, 2016.

[25] E. Romera , J. M. Álvarez, R. Arroyo y L. M. Bergasa, «Efficient ConvNet for Real-time Semantic Segmentation,» *IEEE Intelligent Vehicles Symposium, Proceedings,* pp. 1789-1794, 2017.

[26] Y. Li, J. Shi y D. Lin, «Low-Latency Video Semantic Segmentation.,» *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.,* pp. 5997-6005, 2018.

[27] C. Wu, H.-P. Cheng, S. Li, H. Li y Y. Chen, «ApesNet: a pixel-wise efficient segmentation network for embedded devices.,» *IET Cyber-Physical Systems: Theory & Applications.,* vol. 1, nº 1, pp. 78-85, 2016.

[28] M. Cordts , M. Omran, S. Ramos , T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth y B. Schiele, «The Cityscapes Dataset for Semantic Urban Scene Understanding,» *Computer Vision and Pattern Recognition,* 2016.

[29] G. J. Brostow , J. Fauqueur y R. Cipolla, «Semantic object classes in video: A high-definition ground truth database,» *Pattern Recognition Letters.,* 2008.

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn y A. Zisserman, «The PASCAL Visual Object Classes (VOC) Challenge,» *International Journal of Computer Vision,* vol. 88, pp. 303-338, 2009.

[31] T.-Y. Lin, M. Maire , S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick y P. Dollár, «Microsoft COCO: Common Objects in Context,» 2015.

[32] C. C. Aggarwal, Neural Networks and Deep Learning, Cham, Switzerland: Springer, 2018.

[33] M. Wang, B. Liu y H. Foroosh, «Factorized Convolutional Neural Networks,» de *ICCV*, Venice, Italy, 2017.

[34] M. Z. B. C. D. K. W. W. T. W. M. A. H. A. Andrew G Howard, «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,» *arXiv preprint arXiv:1704.04861,* 2017.

[35] F. Yu y V. Koltun , «Multi-scale context aggregation by dilated convolutions,» de *ICLR*, San Juan, Puerto Rico, 2016.

[36] I. Alonso, L. Riazuelo y A. C. Murillo, «Mininet: An efficient Semantic Segmentation ConvNet for Real-Time Robotic Applications,» *IEEE TRANSACTIONS ON ROBOTICS,* pp. 1552-3098, 2020.

[37] S. Han, J. Tran , J. Pool y W. J. Dally, «Learning both Weights and Connections for Efficient Neural Networks,» de *Nips*, Palais des Congrès de Montréal, Montréal CANADA, 2015.

[38] S. Han, H. Mao y W. J. Dally, «DEEP COMPRESSION: COMPRESSION DEEP NEURAL NETWORKS WITH PRUNINGM TRAINED QUANTIZATION AND HUFFMAN CODING,» de *ICLR*, San Juan, Puerto Rico, 2016.

[39] J. Xie, B. Shuai, J.-F. Hu, J. Lin y W.-S. Zheng, «Improving Fast Segmentation With Teacher-Student Learning,» de *BRITISH MACHINE VISION CONFERENCE*, Newcastle, EN, 2018.

[40] E. Romera, J. M. Álvarez, L. M. Bergasa y R. Arroyo , «ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation,» *IEEE Transactions on Intelligent Transportation Systems ,* vol. 19, nº 1, pp. 263 - 272, 2018.

[41] Q. Zhou, Y. Wang, J. Liu, X. Jin y L. J. Latecki, «An open-source project for real-time image semantic segmentation,» *Science China Information Sciences volume,* vol. 62, p. 227101 , 2019.

[42] Z. Zhang y Y. Pang, «CGNet: cross-guidance network for semantic segmentation,» *Science China Information Sciences,* vol. 63, 2020.

[43] G. Li , S. Jiang, I. Yun, J. Kim y J. Kim, «Depth-Wise Asymmetric Bottleneck With Point-Wise Aggregation Decoder for Real-Time Semantic Segmentation in Urban Scenes,» *IEEE Access,* vol. 8, pp. 27495 - 27506, 2020.

[44] O. Ronneberger, P. Fischer y T. Brox , «U-Net: Convolutional Networks for Biomedical Image Segmentation,» de *Medical Image Computing and Computer-Assisted Intervention*, 2015.

[45] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedman, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich , B. Nessler y S. Hochreiter, «Speeding up Semantic Segmentation for Autonomous Driving,» de *Conference on Neural Information Processing Systems*, 2016.

[46] S. Mehta , M. Rastegari, A. Caspi, Shapiro, Linda y Hajishirzi, «ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation,» de *ECCV*, 2018.

[47] S. Mehta, M. Rastegari, L. Shapiro, Hajishirzi y Hannaneh, «ESPNetv2: A Light-weight, Power Efficient, and General Purpose convolutional Neural Network,» de *Conference on Computer Vision and Pattern Recognition*, 2020.

[48] Y. Wang, Q. Zhou, J. xiong, X. Wu y X. Jin, «ESNet: An Efficient Symmetric Network for Real-Time Semantic Segmentation,» de *Pattern Recognition and Computer Vision*, 2019.

[49] A. Chaurasia y E. Culurciello, «LinkNet: Exploiting encoder representations for efficient semantic segmentation,» de *IEEE Visual Communications and Image Processing*, St. Petersburg, FL, USA, 2017.

[50] S.-Y. Lo, H.-M. Hang, S.-W. Chan y J.-J. Lin, «Efficient Dense Modules of Asymmetric Convolution for Real-Time Semantic Segmentation,» de *ACM International Conference on Multimedia in Asia*, 2019.

[51] M. Liu y H. Yin, «Feature Pyramid Encoding Network for Real-time Semantic Segmentation,» de *BMVC*, 2019.

[52] R. P. K. Poudel, U. Bonde , S. Liwicki y C. Zach, «ContextNet: Exploring Context and Detail for Semantic Segmentation in Real-time,» de *BMCV*, Newcastle, En, 2018.

[53] R. P. K. Poudel, S. Liwicki y R. Cipolla, «Fast-SCNN: Fast Semantic Segmentation Network,» de *BMVC*, Cardiff, En, 2019.

[54] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen, «MobileNetV2: Inverted Residuals and Linear Bottlenecks,» de *Conference on Computer Vision and Pattern Recognition*, Utah, USA, 2018.

[55] H. Zhao, X. Qi, X. Shen, J. Shi y J. Jia, «ICNet for Real-Time Semantic Segmentation on High-Resolution Images,» de *European Conference on Computer Vision*, Munich, Germany, 2018.

[56] L. Riazuelo, A. C. Murillo y I. Alonso, «Enhancing V-SLAM Keyframe Selection with an Efficient ConvNet for Semantic Analysis,» de *International Conference on Robotics and Automation*, Montreal, QC, Canada, 2019.

[57] S. Loffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» de *International Conference on Machine Learning*, 2015.

[58] K. He, X. Zhang, S. Ren y J. Sun, «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,» de *IEEE International Conference on Computer Vision*, Santiago, Chile, 2015.

[59] N. Ma, X. Zhang, H.-T. Zheng y J. Sun, «ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design,» de *European Conference on Computer Vision*, Munich, Germany, 2018.

[60] T. J. Preston, F. Guo, K. Das, B. Giesbrecht y M. P. Eckstein, «Neural Representations of Contextual Guidance in Visual Search of Real-World Scenes,» *Journal of Neuroscience,* vol. 33 , nº 18, pp. 7846-7855, 2013.

[61] M. M. Chun y E. A. Phelps, «Memory deficits for implicit contextual information in amnesic subjects with hippocampal damage,» *Nature Neuroscience,* vol. 2, p. 844–847 , 1999.

[62] A. J. Greene, W. L. Gross, C. L. Elsinger y S. M. Rao , «Hippocampal differentiation without recognition: An fMRI analysis of the contextual cueing task,» *Learning Memory,* vol. 14, pp. 548-553, 2007.

[63] I. Biederman, R. J. Mezzanotte y J. C. Rabinowitz, «Scene perception: Detecting and judging objects undergoing relational violations,» *Cognitive Psychology,* vol. 14, nº 2, pp. 143-177, 1982.

[64] J. L. Davenport y M. C. Potter, «Scene Consistency in Object and Background Perception,» *Psychology Science,* vol. 15, nº 8, pp. 559-564, 2004.

[65] A. Torralba y A. Oliva , «The role of context in object recognition,» *Trends in Congnitive Sciences,* vol. 11, nº 12, pp. 520-527, 2007.

[66] M. Bar, «Visual objects in context,» *Nature Reviews Neuroscience,* vol. 5, p. 617–629, 2004.

[67] M. V. Peelen y T. Brandman, «Interaction between Scene and Object Processing Revealed by Human fMRI and MEG Decoding,» *The Journal of Neuroscience,* vol. 37, nº 32, pp. 7700-7710, 2017.

[68] B. Chen , C. Gong y J. Yang, «Importance-Aware Semantic Segmentation for Autonomous Vehicles,» *IEEE Transactions on Intelligent Transportation Systems,* vol. 20, nº 1, pp. 137 - 148, 2019.

[69] A. Paszke, A. Chaurasia, S. Kim y E. Culurciello, «ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation,» *arXiv:1606.02147 [Computer Vision and Pattern Recognition],* 2016.

[70] Z. Wojna, J. Uijlings, S. Guadarrama, N. Silberman, L.-C. Chen, A. Fathi y V. Ferrari, «The Devil is in the Decoder,» de *British Machine Vision Conference*, London, UK, 2017.

[71] M. Nordin Mokti y R. A. Salam, «Hybrid of Mean-shift and median-cut algorithm for fish segmentation,» de *International Conference on Electronic Design*, Penang, Malaysia, 2008.

[72] V. Badrinarayanan, A. Kendall y R. Cipolla, «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, nº 12, pp. 2481 - 2495, 2017.

[73] X. Zhang, Z. Chen, J. Wu, L. Cai, D. Lu y X. Li, «Fast Semantic Segmentation for Scene Perception,» *IEEE Transactions on Industrial Informatics,* vol. 15, nº 2, pp. 1183 - 1192, 2019.

[74] A. Tao, K. Sapra y B. Catanzaro, «Hierarchical Multi-Scale Attention for Semantic Segmentation,» de *arXiv*, 2020.

[75] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik , I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg y D. Hassabis, «Human-level control through deep reinforcement learning,» *Nature,* nº 518, pp. 529-533, 2015.