UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
MAESTRÍA EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
IIMAS-UNAM

APRENDIZAJE PROFUNDO MULTITAREA PARA VISIÓN EN UN SISTEMA EMBEBIDO

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRÍA EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
ING. KRISTEN ISMAEL VEGA GAMBOA

TUTOR:
DR. PAUL ERICK MÉNDEZ MONROY
IIMAS-Mérida

MÉRIDA, JUNIO 2022

Agradecimientos

A todo el equipo del programa de posgrado en la Universidad Autónoma de México, por su compromiso y apoyo durante toda la maestría.
A mi director de tesis, por su constante apoyo y guía durante todo el proceso.
A los integrantes del jurado de tesis, por sus observaciones y comentarios.
Al Consejo Nacional de Ciencia y Tecnología por la beca brindada para mi formación académica.
A los profesores del programa del posgrado por su guía, atención y comprensión durante los cursos.
Al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica por su apoyo en el proyecto UNAM-PAPIIT IA102620.

Resumen

En este trabajo, se desarrolló un sistema multitarea para estimación del mapa de profundidad, de clasificación y localización de objetos en una imagen RGB monocular de transito urbano. El sistema es una arquitectura de aprendizaje profundo de redes neuronales convolucionales, que para su inferencia requiere transferencia de aprendizaje y una estrategia de entrenamiento propuesta.

Por un lado, la primera red está basada en una red MobileNetV2 modificada con decoder de Unet que se encarga de la estimación de la profundidad y por otro lado, una red Faster-RCNN con núcleo MobileNetV2 para realizar la clasificación y localización de objetos. Dichas dos sub-redes comparten parte del encoder correspondiente a la MobileNetV2, lo que llevó al desarrollo de una estrategia de entrenamiento con transferencia de aprendizaje y de inferencia de dicho sistema.

Los principales objetivos del sistema son reducir el tiempo de inferencia y el uso de recursos computacionales, para mejorar su rendimiento en un sistema embebido. La red se entrenó y evaluó usando los conjuntos de datos de KITTI suite, de estimación de la profundidad, clasificación y localización de objetos, para entrenar en sus respectivas tareas. El trabajo se enfocó en llevar acabo la metodología del diseño, implementación y prueba de las redes y de la arquitectura propuesta.

Se entrenaron las redes MobilenetV2, Faster-RCNN y SSD, usadas en la implementación y pruebas del ensamble propuesto. En los resultados de dicha evaluación, aunque la precisión no está al nivel del estado de arte, se puede decir que sugieren que la red ensamblada propuesta es capaz de utilizar los mapas generados en la MobilenetV2 de la estimación de la profundidad en la inferencia de la detección de objetos.

Índice General

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Problemática

En los sistemas de navegación autónoma es fundamental realizar las tareas de localización y mapeo simultáneo (SLAM, por sus siglas en inglés), que entre sus tareas principales están en el reconocimiento de la clase y la posición relativa de las entidades circundantes en el ambiente, esto genera parte de la información que el sistema de navegación necesita para tomar una decisión inteligente (Santana & Medeiros, 2009).

Actualmente, en el campo de la visión computacional, los sistemas de aprendizaje profundo de arquitectura basada en redes neuronales convolucionales (CNN, por sus siglas en inglés) están entre los de mejor desempeño para las tareas necesarias para la navegación autónoma, tales como, estimación de profundidad (Spek, Dharmasiri & Drummond, 2018), (Bokovoy, Muravyev & Yakovlev, 2019), (Wofk, Ma, Yang, Karaman & Sze, 2019) y de reconocimiento de objetos (C.-E. Wu, Chan, Chen, Chen & Chen, 2019), (H. H. Wu y col., 2019), (Girshick, Donahue, Darrell & Malik, 2014) (Cai & Vasconcelos, 2019). También, se ha ampliado la investigación en el uso de las CNNs en la resolución de problemas multitarea y se han desarrollado sistemas con estrategias de transferencia de aprendizaje, como por ejemplo (Wofk y col., 2019) y (Hazirbas, Ma, Domokos & Cremers, 2016), que hacen uso de una red previamente entrenada en la clasificación de objetos para su sistema de estimación del mapa de profundidado y de segmentación semántica, respectivamentamente.

En este trabajo, se desarrolló un sistema de aprendizaje profundo basado en

arquitecturas CNNs para la resolución de problemas multitarea de aplicación en la visión computacional. El sistema utiliza una imagen 2D RGB para llevar acabo las tareas de estimación de profundidad, de localización y clasificación de objetos, con el fin de ejecutarlas de manera más eficiente en comparación que por separado, buscando ampliar la viabilidad para su ejecución en sistemas móviles embebidos, en aplicaciones de navegación autónoma.

1.2. Objetivos

1.2.1. General

Desarrollar un sistema de arquitectura de red de aprendizaje profundo multitarea basada en redes neuronales convolucionales para la estimación del mapa de profundidad, localización y clasificación de objetos, con un rendimiento viable para su utilización en dispositivos embebidos.

1.2.2. Específicos

1. Entrenar las tres redes convolucionales a utilizar para el desarrollo del sistema propuesto.

    a) Implementar y evaluar de red basada en MobilenetV2 para estimación de profundidad.

    b) Implementar y evaluar de red Faster-RCNN y SSD, para el reconocimineto de objetos.

2. Implementar del sistema propuesto, ensamble de la red para llevar acabo las tres tareas propuestas a partir de un imagen 2D RGB monocular.

3. Implementar la estrategía de entrenamiento con transferencia de conocimiento para el sistema propuesto.

4. Implementar y evaluar el rendimiento en hardware dedicado tipo GPU y de la tarjeta Jetson TX2.

1.3. Estructura de la tesis

La tesis está organizada de la siguiente forma:

Capítulo 2 - Repasa los principales conceptos sobre las redes de aprendizaje profundo y sus componentes, necesarios para llevar acabo el proyecto.

Capítulo 3 - Realiza un resumen y análisis de los trabajos similares, que sirvieron de inspiración, de base teórica.

Capítulo 4 - Conforma una descripción de las tareas realizadas en la metodología del diseño de las redes, su implementación, estrategia de entrenamiento y pruebas de rendimiento.

Capítulo 5 - Muestra los resultados de las pruebas de rendimiento de los modelos en las condiciones descritas en la capítulo previo.

Capítulo 6 -Se finaliza por dar una conclusión en base a los objetivos y resultados obtenidos.

Anexo -Se incluye las partes más importantes del codigó de las definiciones de implementación de los modelos, como referencia.

Capítulo 2

Marco teórico

En este capítulo se presenta un resumen de los tipos de redes neuronales convolucionales usadas en este trabajo, una descripción de las tareas de visión computacional, información sobre el hardware del sistema embebido donde se desea que el sistema funccione y la composición de los conjuntos de datos usados en el entrenamiento y evaluación de las redes.

2.1. Redes neuronales convolucionales

Las redes neuronales convolucionales profundas (Yann LeCun, Bengio & Hinton, 2015) (DCNN, por siglas en inglés) son un grupo de modelos que hacen uso de las redes neuronales multicapa, operaciones convoluciones, entre otras operaciones matemáticas, para transformar y realizar acabo una clasificación, regresión o estimación, al optimizar una función de coste. En el área de la visión computacional son de amplio estudio por que se encuentran entre los sistemas de mayor capacidad de reconocimiento de objetos, estimación de la profundidad, entre otras.

La parte más básica de una DCNN, es la red neuronal que se basa en el modelo del perceptrón (Rosenblatt, 1958), inspirado en el entendimiento del funcionamiento biológico de las neuronas, con el objetivo de recrear su capacidad de regresión. Unas décadas más tarde, gracias a los aportes del trabajo de (Y. LeCun y col., 1989) de aplicación del método de *Backpropagation* para el entrenamiento de redes neuronales multicapa y demostró su relevancia en la clasificación de objetos en la visión computacional.

Tabla 2.1: Componentes de la red MobileNetV2. Edición de la tabla 2. Fuente: (Howard y col., 2017).

Entrada	Operación
$224^2 \times 3$	conv2d
$112^2 \times 32$	*bottleneck*
$112^2 \times 16$	*bottleneck*
$56^2 \times 24$	*bottleneck*
$28^2 \times 32$	*bottleneck*
$14^2 \times 64$	*bottleneck*
$14^2 \times 96$	*bottleneck*
$7^2 \times 160$	*bottleneck*
$7^2 \times 320$	conv2d 1x1
$7^2 \times 1280$	avgpool 7x7
$1 \times 1 \times 1280$	conv2d 1x1

Por lo general, una CNN se conforma de bloques de convoluciones, operaciones de agrupación y capas de neuronas. Si bien no existe un número exacto de capas neuronales a apartir del cual definir el aprendizaje profundo (Yann LeCun y col., 2015), si se han estudiado ampliamente sobre los retos y estrategias para entrenar DCNNs de decenas de capas (He, Zhang, Ren & Sun, 2016) (Ioffe & Szegedy, 2015).

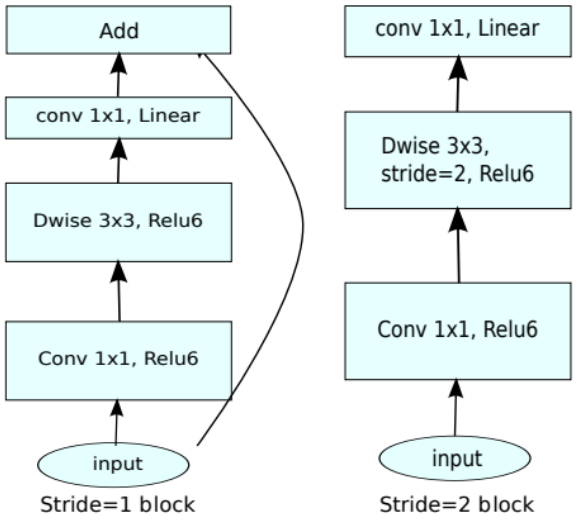
Dentro de este trabajo se emplearon varios modelos de redes neuronales convolucionales, la MobileNetV2, Unet, Faster-RCNN y SSD, a continuación se presenta una descripción de cada una.

2.1.1. MobileNetV2

Es un modelo de CNN propuesto por (Sandler, Howard, Zhu, Zhmoginov & Chen, 2018), capaz de ser entrenado para las tareas clasificación de objetos en imágenes y está ideado para su inferencia en dispositivos móviles, sistemas embebidos de bajo consumo o dispositivos sin aceleración por hardware dedicado que hacen uso solo del CPU.

La red se conforma de varios bloques básicos de construcción de embotellamiento residual (*bottleneck*) que se basan en el uso de bloques residuales (He

y col., 2016) que consisten en una convolución 2D (con2d) 1x1 de activación ReLu, seguido de una convolución separable en profundidad (*Depthwise Separable Convolution*, Dwise) de activación ReLu, con un conexión de adicción entre los *bottlenecks*, (ilustrado en la figura 2.1). En la tabla 2.1, se enlistan las capas de bloques de la propuesta de MobileNetV2 para la tarea de clasificación.



Mobilenet V2

Figura 2.1: Bloques de construcción tipo *bottleneck* de MobileNetV2. Fuente: (Sandler, Howard, Zhu, Zhmoginov & Chen, 2018).

2.1.2. Unet

Es un modelo de DCNN propuesto por (Ronneberger, Fischer & Brox, 2015) para segmentación semántica de imágenes biológicas, con una arquitectura encoder-decoder con concatenaciones entre ambas. Como se muestra en la figura 2.2, su encoder se conforma de operaciones de convolución, que puede ser sustituido por un encoder de otras redes de clasificación de objetos como puede ser MobileNetV2. Su decoder es una sucesión de bloques conformados por convoluciones y operaciones de sobre resolución *upsampling*.

La operación *Max pooling downsampling* consiste en una convolución 2D que toma el valor máximo registrado en el kernel, con un paso *stride* de 2 x 2,

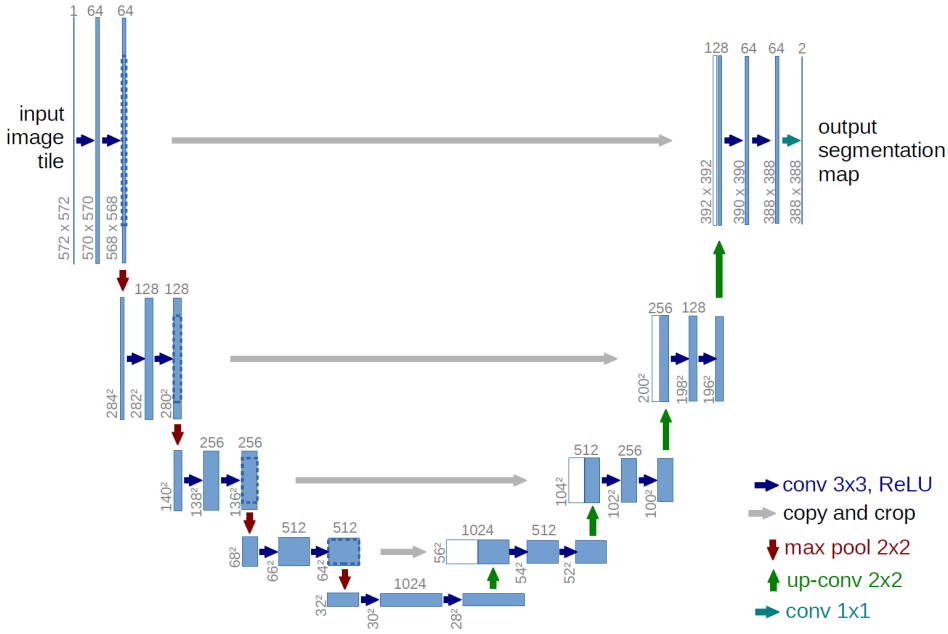


Figura 2.2: Representación de arquitectura de la Unet. Fuente: (Ronneberger, Fischer & Brox, 2015).

para reducir las dimensiones de los mapas de características.

Las conexiones *skip conections* (flechas grises en la Figura 2.2) consisten en llevar acabo una unión entre las diferentes capas que conforman una determinada CNN, usualemente entre partes del encoder y decoder. Dicha unión, frecuentemente se utiliza de dos tipos, una primera como la usada en MobileNetV2, que lleva acabo una operación de adición entre los elementos, la segunda, como la utilizada en Unet que ejecuta una operación de concatenación.

2.1.3. Faster R-CNN

La propuesta de (Ren, He, Girshick & Sun, 2017) es una arquitectura que ha influido notablemente en el desarrollo de CNNs para localización y clasificación de objetos. La Faster R-CNN presenta una red de propuesta de regiones (RPN, por sus siglas en inglés), para realizar la estimación de las propuestas (proposals) de las áreas de interés (RoI, por sus siglas en inglés) para la Fast R-CNN (Girshick, 2015) que estima la clase y coordenadas de esa RoI, a través de la aplicación de una operación de Max *Pooling* no uniforme, seguida de una

capa completamente conexa. Se construye desde una red base (*Backbone*), por ejemplo MobileNetV2 o Resnet, que toma la imagen RGB de entrada y genera los mapas de características (features maps) que sirven de entrada para la RPN, que su vez genera la entrada de RoIs para *Pooling* para la regresión de las coordenadas de los *bounding boxes* y clasificación del objeto en la Fast-RCNN. Dicho *Backbone* suele encontrarse pre-entrenado para tarea de clasificación de objetos. En la figura 2.3, se ilustra el flujo de datos de la Faster R-CNN.

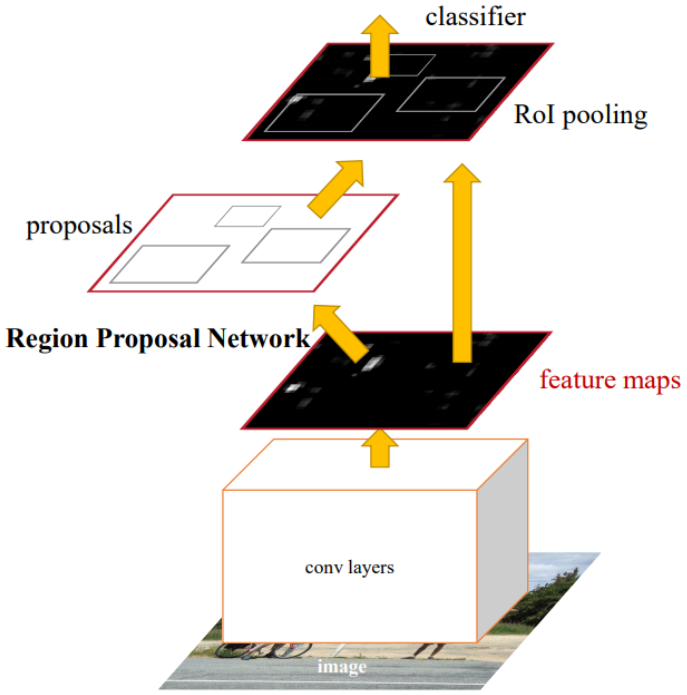


Figura 2.3: Representación de la Faster-RCNN. Fuente: (Ren, He, Girshick & Sun, 2017).

2.1.4. *Single Shot MultiBox Detector*

Es un modelo de CNN propuesto por (W. Liu y col., 2016a), está diseñado para realizar las tareas clasificación y localización de objetos en imágenes, puede utilizar un *backbone* VGG, MobileNetV2, etc. Su modelo consiste en agregar capas de convoluciones que reducen el tamaño de los mapas de caracteristícas progresivamente y permiten estimaciones a multiples escalas. Cada capa de caracteristícas extra produce un grupo de detecciones usando un conjunto de filtros convolucionales, cada uno de tamaño 3 x 3 x p, número de canales que

produce a su vez, una puntuación para una clase de objeto y la estimación de su *bounding box* de posición relativa a la ubicación de la representación del objeto en el mapa de caracteristícas en cuestión.

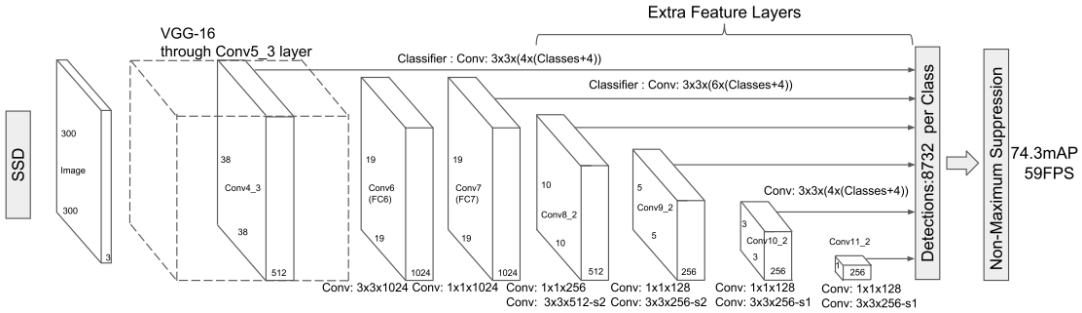


Figura 2.4: Representación de la arquitectura de la red SSD con *backbone* VGG-16. Fuente: (W. Liu y col., 2016a).

2.2. Tareas de visión computacional

En esta sección se describen las tareas de visión computacional que se desea que el sistema realice para este trabajo. Se describen en que consisten la entrada y salida que se espera del sistema.

2.2.1. Clasificación de objetos

Esta tarea consiste en la capacidad de un sistema para identificar la clase de un objeto a partir de sus características, típicamente visuales expresadas en una o varias imágenes obtenidas con una o varias cámaras fotográficas (Shen, 2019).

Se pueden encontrar una amplia cantidad de propuestas basadas en CNNs para realizar dicha tarea, entre las populares más recientes está la MobileNetV2. Dicho modelo fue diseñado para ocupar la menor cantidad de recursos de cómputo, memoria y operaciones de cálculo, con el objetivo de ser ejecutado en dispositivos móviles.

En esta tarea el trabajo contempla que dada una imagen 2D RGB de dimensiones, ancho por alto por 3 canales, realice una regresión tal que a

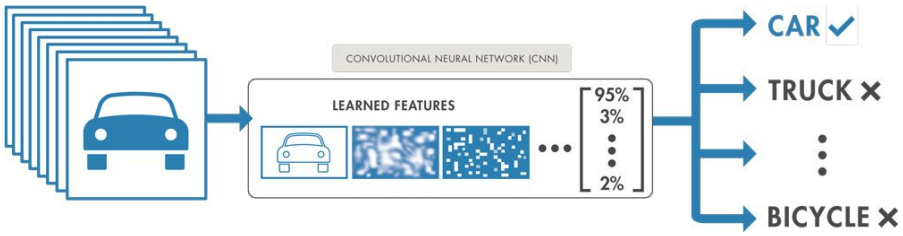


Figura 2.5: Representación de clasificación de una imagen por CNN. Fuente: (MATLAB, 2021).

cada detección de *bounding box* de posible objeto en la imagen, se le asigne una clase.

2.2.2. Detección de objetos

Esta tarea consiste en la estimación de la proyección de un objeto en una imagen, se determinan las dimensiones del área y la posición de la proyección de dicho objeto. Esta información es necesaria para la estimación de la pose relativa del objeto a la cámara.

Entre los algoritmos más usados para tal fin, se encuentra la propuesta de Faster-RCNN que consiste en una CNN que realiza la estimación del tamaño y posición de las regiones de interes en una imagen que a su vez sirven típicamente de entrada a otra red basada en CNNs que realiza la clasificación del objeto en dicha imagen (Shen, 2019).

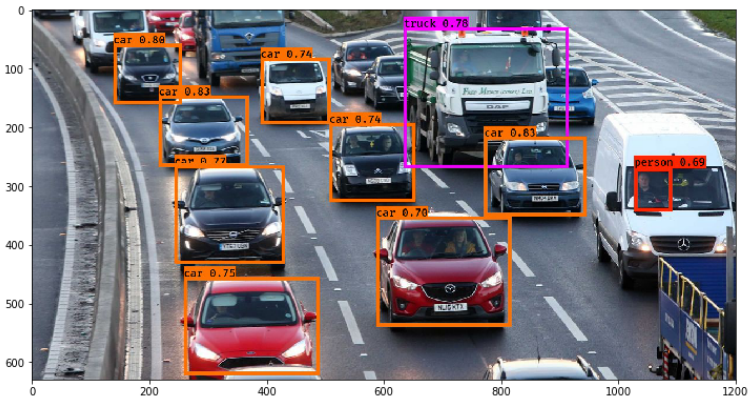


Figura 2.6: Ejemplo de localización y clasificación de objetos. Fuente: (PulkitS, 2018).

En este trabajo, se desea que dada una imagen 2D RGB de dimensiones, ancho por alto por 3 canales, se realice una regresión tal que se determine un área de dimensiones ancho x alto de la proyección de un objeto detectado sobre el plano 2D de la misma cámara.

2.2.3. Estimación de mapa de profundidad

Es una tarea indispensable para la navegación y consiste fundamentalmente en la reconstrucción de los objetos del entorno, lo que permite en conjunto tener un sistema de localización y mapeo simultaneos (SLAM, por sus siglas en inglés), la creación de una imagen tridimensional de la distribución del entorno, etc. Generalmente con el fin de realizar navegación autónoma, reconocimiento o interactuar con el ambiente circundante.

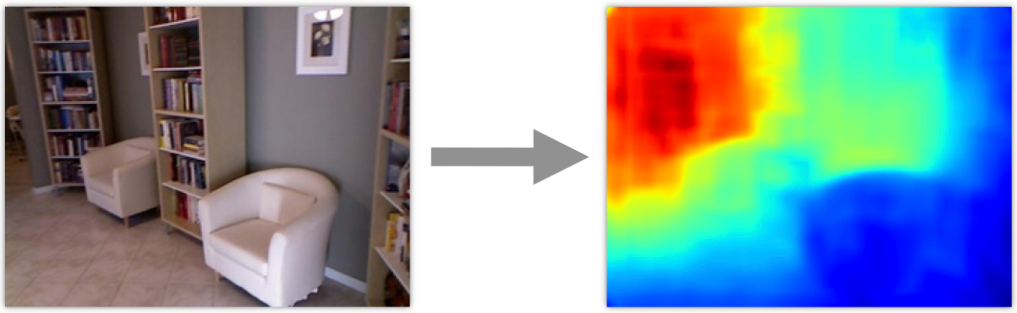


Figura 2.7: Ejemplo de estimación de mapa de profundidad. A la izquierda imagen de entrada, a la derecha el mapa de profundidad, donde la escala de colores representa azul más cercano y rojo más lejano. Fuente: (Eigen, Puhrsch & Fergus, 2014).

Se desea que dada una imagen 2D RGB de dimensiones, ancho por alto por 3 canales, se realice una regresión tal que se determine una imagen del mismo ancho x alto, correspondiente al mapa de profundidad con valores de distancia del entorno respecto a la cámara fotográfica (Kim & Hwang, 2021).

En el contexto de CNNs, entre las propuestas interesantes se encuentra la de (Wofk y col., 2019) que consta de una red Unet con encoder de MobileNetV2 que logra la estimación del mapa de profundidad de una imagen.

2.3. Tarjeta de desarrollo Jetson TX2

Es un dispositivo desarrollado por la empresa NVIDIA para aplicaciones de cómputo embebido con inteligencia artificial. Tiene capacidad de cómputo y ancho de banda para ejecutar redes neuronales con un bajo consumo eléctrico. Ideado para aplicaciones de aprendizaje maquinal en particular de ejecución de CNNs, en un formato compacto pensado para el desarrollo y comercialización de dispositivos autónomos.

El hardware es de arquitectura heterogénea que cuenta con una unidad central de procesamiento de arquitectura ARM de cuatro núcleos Cortex A57 y un acelerador de procesamiento de gráficos de arquitectura Pascal con 256 núcleos CUDA, con una potencia de más de 1 TFLOP en precisión simple y un consumo eléctrico configurable con un máximo de hasta 15 vatios. Además, cuenta con soporte para una amplia gama de periféricos, como conexión SATA, M.2 para unidades de memoria de alta velocidad y soporte para varias cámaras de resolución 1080p.

Un problema usual para la implementación de sistemas de aprendizaje profundo en un sistema embebido como la Jetson TX2, es su limitada capacidad de cálculo y memoria de ejecución (RAM) en comparación a un sistema de escritorio o servidor. Por lo que es prioritario la optimización y mejora de los modelos para reducir el uso de los recursos computacionales e incrementar su uso en sistemas embebidos.

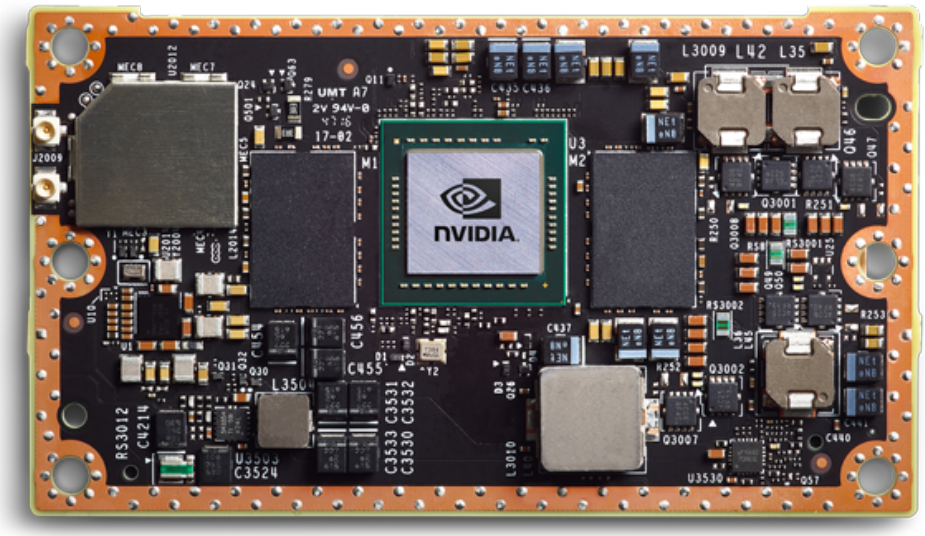


Figura 2.8: Foto del módulo de la Jetson Tx2. Fuente: (NVIDIA, 2019).

2.4. Conjunto de Datos

Las base de datos KITTI Vision Benchmark Suite (Geiger, Lenz & Urtasun, 2012) contiene varios conjuntos de entrenamiento y validación de competencias *Benchmarks* para varias tareas de navegación autónoma. Para fines de este trabajo se usaron dos conjuntos, KITTI Depth y KITTI Object.

El conjunto de KITTI Depth tiene más de 80 mil imágenes RGB cada uno su respectivo mapas de profundidas de fondo verdadero, está dividido en dos sub-conjuntos, uno de entrenamiento y el otro de validación. Los mapas de profundidad tienen una densidad de alrededor de 10 porciento de datos de validos, con una dimensión de 375 pixeles de altura por 1242 pixeles de largo con valores que representan la profundidad detectada por un sensor tipo Lidar.

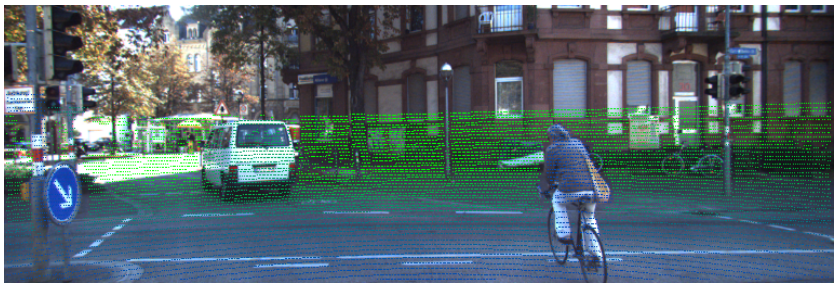


Figura 2.9: Muestra de imagen RGB con fondo verdadero de profundida de KITTI Depth. Fuente: (slicer, 2021).

El conjunto de KITTI Object tiene más de 14 mil imágenes RGB, cada una de ellas con su respectivo archivo de anotaciones con cuadros delimitadores y la clase vinculada a cada una. Las imágenes están en la misma resolución que las de KITTI Object. Se divide en tres sub conjuntos, de entrenamiento, prueba y validación. Entre dichos sub-conjuntos existen 8 clases y en la distribución de que se muestra en la figura 2.11

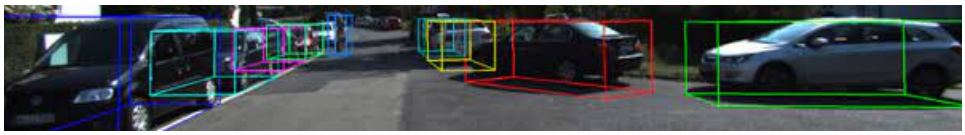


Figura 2.10: Muestra de imagen con etiquetas de posición 3D de KITTI Object. Fuente: (Geiger, Lenz & Urtasun, 2012).

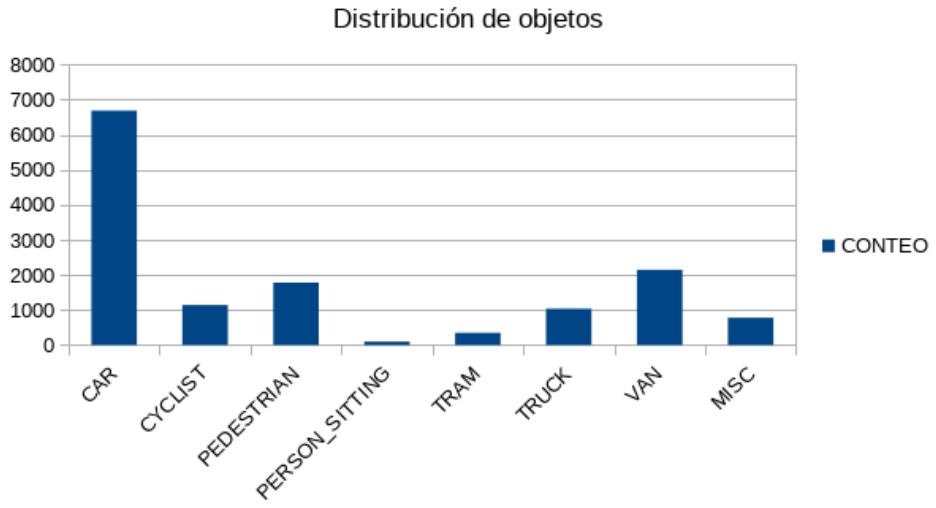


Figura 2.11: Distribución de clases en el conjunto KITTI Object.

Capítulo 3

Trabajos relacionados

Se presenta un resumen de las obras con los fundamentos que sirvieron como base para este trabajo. Se muestran según corresponda a su respectiva tarea de visión computacional. Donde, el reconocimiento de objetos incluye a la localización y clasificación de objetos. También se abordan trabajos sobre la estimación de mapas de profundidad con una imagen monocular. Finalmente, se analiza sobre las redes multi-tarea que incluyan alguna de las tareas previas, entre otras en la visión computacional.

3.1. Redes de aprendizaje profundo para reconocimiento de objetos

Un sistema de reconocimiento de objetos por visión computacional debe ser capaz, entre otras cosas, de realizar las tareas de localización y clasificación de objetos. En ese sentido, los desarrollos con uso de sistemas basados en redes neuronales para la estimación de la clase de una determinada imagen surgieron desde los años 80's del siglo pasado, luego, surgieron las técnicas de localización del objetos en una imagen.

Actualmente, los sistemas basados en CNNs para el reconocimiento de objetos suelen ser clasificados como de una o dos etapas (*stages*), en la figura 3.2, se illustra este concepto, por redes de una etapa, se refiere a los modelos que realizan la regresión de la ubicación de los *bounding boxes* y la clasificación del objeto contenido en ellos, usando un decoder unificado. Por su parte, las redes

de dos etapas, derivan de adicciones a la propuesta de R-CNN (Girshick, Donahue, Darrell & Malik, 2013) y usan una segunda red encargada de optimizar la generación de las regiones de interés de los *bounding boxes*.

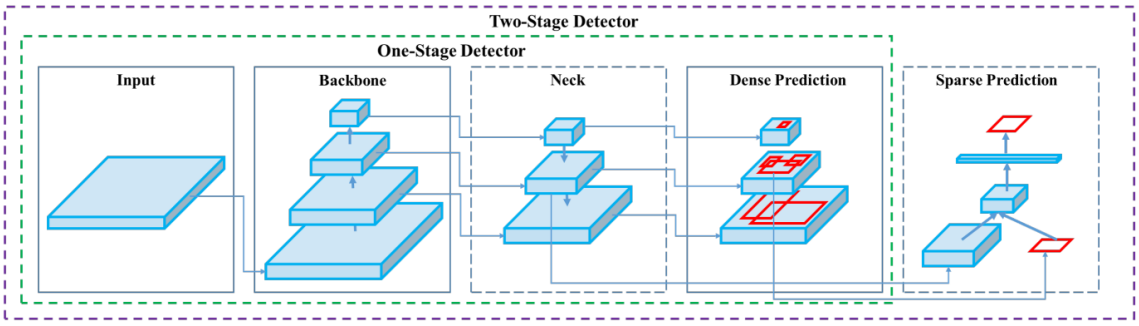


Figura 3.1: Representación de tipos de redes de reconocimiento de objetos. Fuente: (Lin y col., 2016).

Entre las redes más pupulares de una etapa, está YOLO (Redmon, Divvala, Girshick & Farhadi, 2016) que es capas de encontrar un máximo de 98 *bounding boxes* o SSD (W. Liu y col., 2016b) que estima 8732 *bounding boxes*. Por otra parte, entre las redes de dos etapas podemos encontrar la Faster-RCNN, Feature Pyramid Networks (Lin y col., 2016) o Cascade R-CNN (Cai & Vasconcelos, 2019). Y más recientemente, han surgido propuestas como (Soviany & Ionescu, 2018) que son un sistema con características de ambas.

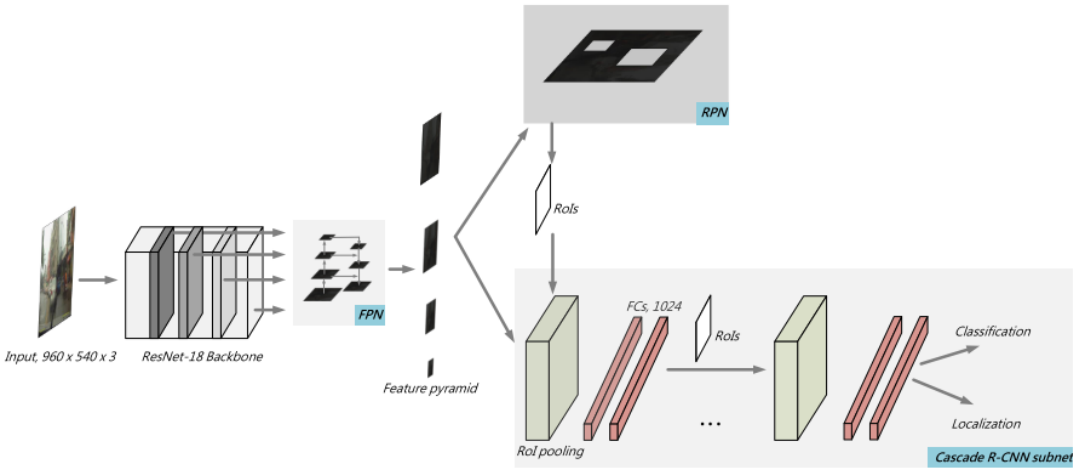


Figura 3.2: Representación de arquitectura de IMMVP. Fuente: (C.-E. Wu, Chan, Chen, Chen & Chen, 2019).

En la literatura, se encuentran trabajos como el de *Wu et al.* (C.-E. Wu y col., 2019) que propone una red profunda de arquitectura de dos etapas, conformada por *backbone* de ResNet18 seguida de una red piramidal de características (FPN, por sus siglas en inglés) que finaliza por una Cascade R-CNN, capaz de detectar peatones, automóviles y ciclistas. También, lleva acabo una estrategia de etiquetado de valores esperados en cosas como el clima u hora del día, que permite considerables mejoras en la presición.

Por otra parte, en (Y. Liu, Cao, Lasang & Shen, 2019) abordan una estrategia que nombraron Modular Feature Fusion Detector (Y. Liu, Cao, Lasang & Shen, 2021) que permite mejorar la capacidad de detección de objetos pequeños y reducir el uso de recursos computacionales. Estos dos últimos trabajos presentan resultados de rendimiento en el sistema embebido Jetson TX2.

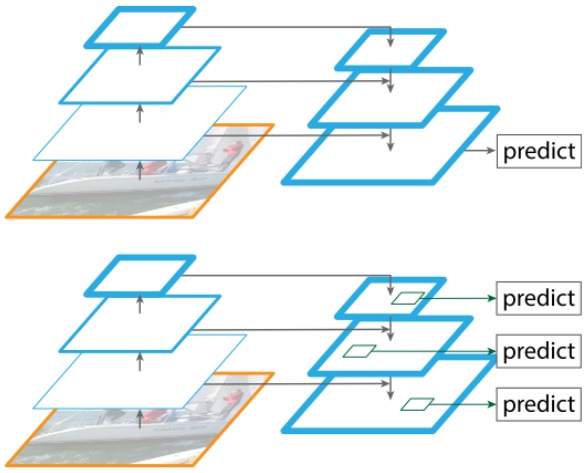


Figura 3.3: Representación de la Feature Pyramid Network (FPN) . Fuente: (Lin y col., 2016)

Otro trabajo interesante es, la red de piramidal de características (FPN, por sus siglas en inglés) (Lin y col., 2016) consiste en bloques de mapas de características laterales al encoder de una *backbone*, por ejemplo Resnet o MobileNet. Se puede utilizar en unión con otras redes independientemente de la tarea a resolver. En la figura 3.3 se muestra a la izquierda los mapas generados por el *backbone* y a la derecha los obtenidos por las operaciones de recomposición *upsamble* de la FPN. En la figura 3.4 se ilustra las operaciones que generan los niveles de la piramide. Finalmente, se aplica una convolución 3 x 3 para obtener los mapas finales.

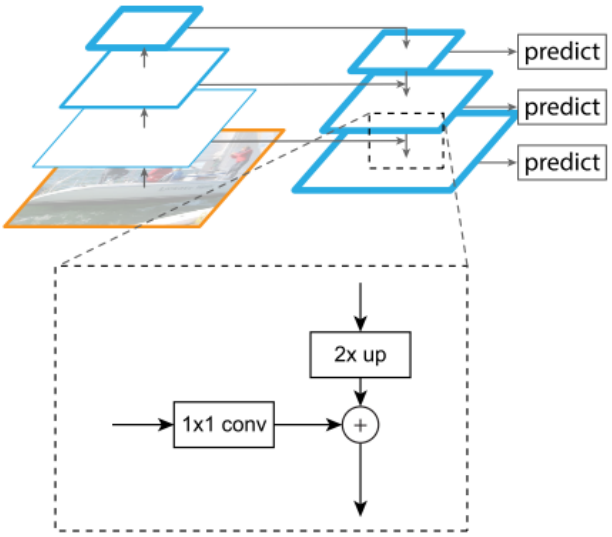


Figura 3.4: Representación de bloque de FPN. Fuente: (Lin y col., 2016).

3.2. Redes de aprendizaje profundo para estimación de profundidad

Con respecto al cálculo de la profundidad, una propuesta interesante es la de (Bokovoy y col., 2019) que consta de una red ResNet50 con modificaciones en el Encoder. Esta red toma imágenes de una cámara monocular y logra un tiempo de ejecución de 16 cuadros por segundo (fps) en una Jetson TX2 y 0.65 de error cuadratico medio.

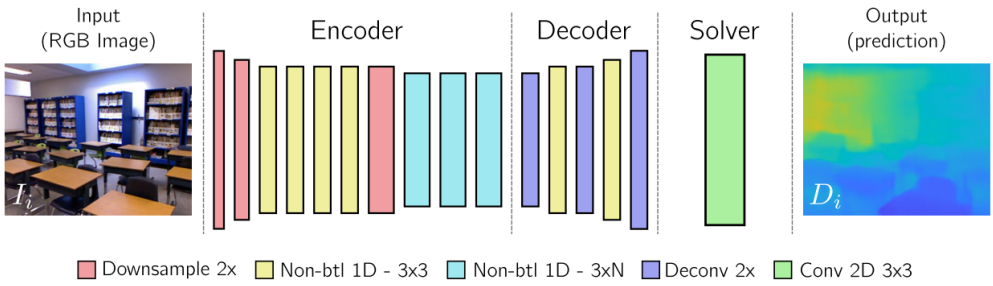


Figura 3.5: Representación de arquitectura de CReaM. Fuente: (Spek, Dharmasiri & Drummond, 2018).

Por otra parte, se encuentra la propuesta de (Spek y col., 2018) llamada

CReaM, que consiste en una red de arquitectura encoder-decoder, que hace uso de bloques *Non-bottleneck* que no modifican el conteo de canales.

En el trabajo de Spek *at al*, incluye un sistema para comprimir la arquitectura, a través de una estrategia que logra aprender el funcionamiento de una red más grande, con baja variación en la precisión, pero proporciona una considerable mejora en el tiempo de ejecución, ese sistema fue inspirado en las propuestas de (Paszke, Chaurasia, Kim & Culurciello, 2016) y (Bergasa, Arroyo, Romera & Alvarez, 2018) que fueron diseñas con el propósito de realizar segmentación semántica y con alto rendimiento y eficiencia. La CReaM logra ejecutarse a una velocidad de 30 fps en la Jetson TX2.

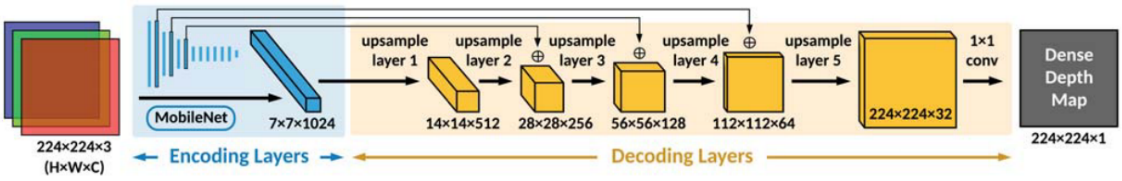


Figura 3.6: Representación de la arquitectura de FastDepth. Fuente: (Wofk, Ma, Yang, Karaman & Sze, 2019).

Otro trabajo interesante es FastDepth (Wofk y col., 2019) que es una DCCN de arquitectura encoder-decoder. Hace uso de la estrategia propuesta por (Yang y col., 2018) llamada NetAdapt para realizar una poda de red (*Network Pruning*) y reducir el tiempo de inferencia. La red toma imágenes de una cámara monocular y logra un tiempo de ejecución de 179 fps en una Jetson TX2 con una alta precisión.

3.3. Redes multi-tarea de aprendizaje profundo en visión computacional

En este tipo de redes profundas, se refiere a la capacidad de un sistema de extraer información de una imagen y hacer regresiones para varios problemas de clasificación o regresión. Sistemas de CNNs que a través de su arquitectura logran compartir internamente la mayor cantidad de hiper-parámetros, para la resolución de varias tareas, como por ejemplo la estimación de la profundidad,

el reconocimiento de objetos o la segmentación semántica.

Se desea un sistema que utilice, de existir, la información de correlación entre la posición de un objeto, su morfología, ubicación y posible clase para resolver las tareas de visión computacional.

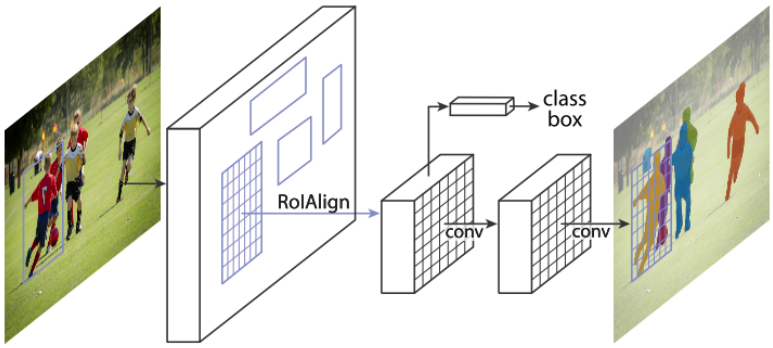


Figura 3.7: Representación de arquitectura Mask R-CNN. Fuente: (He, Gkioxari, Dollár & Girshick, 2020)

En este sentido, se puede encontrar la propuesta de (S. Liu & Liu, 2017) que utiliza un sistema de dos etapas para el recocimiento de objetos. Este toma como entrada imágenes preprocesadas de diferentes tipos (Infrarojo, mapas de movimiento, etc) e implementa transferencia de aprendizaje entre algunas capas especificas de su arquitectura. La propuesta de Mask RCNN (He, Gkioxari, Dollár & Girshick, 2020) consiste en un arquitectura basada en Faster R-CNN, con un primer estado conformado por la RPN y que adiciona una etapa de generación de mascara de segmentación semántica que se ejecuta en paralelo a la Fast R-CNN y genera una mascara binaria para cada RoI. La figura 3.7 representa el flujo de información de este segundo estado.

En este trabajo, para la tarea de estimación de los mapas de profundidad, se basa en el modelo de FastDepth con *backbone* de red MobileNetV2, debido a sus cualidades de alta presición y bajo uso de recursos de cómputo para la inferencia. Por otra parte, la Faster RCNN y SSD son utilizas como base para la detección de objetos en el ensamble propuesto que proporciona una reducción en el uso de recursos de cómputo, a la vez que su estrategia de sistema multitarea pretende ser base para una mejora en presición en la detección de objetos.

Capítulo 4

Metodología

En este capítulo se describen las actividades realizadas en todo el desarrollo del presente trabajo. Se enlistan y detallan de manera sucesiva los pasos principales del proceso de desarrollo hasta la prueba del sistema de aprendizaje profundo propuesto.

A continuación, se presenta una lista resumen de las actividades planeadas y realizadas:

1. Selección de las redes de aprendizaje profundo.
2. Configuración del entorno de desarrollo.
3. Funciones de coste y métricas de rendimiento.
4. Implementación de los modelos.
5. Estrategia de entrenamiento e inferencia.
6. Pruebas de rendimiento.

4.1. Selección de las redes de aprendizaje profundo

Se comenzó por realizar las selección de las CNNs a utilizar como base del diseño. Para lo cual se aplicaron los siguientes criterios principales:

1. Menor volumen del modelo.

2. Menor cantidad de hyper-parámetros.

3. Menor número de operaciones.

4. Menor uso de memoria.

5. Disponibilidad en software compatible.

En base a los criterios descritos, se seleccionaron tres arquitecturas para la realización del modelo multitarea. La red de estimación de profundidad que se eligió es una arquitectura Unet (Ronneberger y col., 2015) con encoder MobileNetV2 (Alhashim & Wonka, 2018). Para la sub-red de localización y clasificación de objetos se decidió por una Faster-RCNN (Ren y col., 2017) con base MobileNetV2, principalmente por su amplia documentación en la literatura, disponibilidad de su implementación en la framework de Tensorflow (Martín Abadi y col., 2015).

También, se eligió la red SSD (W. Liu y col., 2016a), como una segunda opción para la tarea de localización y clasificación de objetos, con el proposito de probar su rendimiento bajo una estrategia similar a la aplicada con la Faster R-CNN, tener punto de comparación entre sus número de hiper-parámetros y tiempo de inferencia.

4.2. Configuración del entorno de desarrollo

Con los modelos a trabajar, lo primero es instalar el Framework del módulo Keras de Tensorflow, en el lenguaje Python. Se debe adecuar la implementación disponible en Keras al modelo base de la MobileNetV2.

Se realizó la instalación de los paquetes de software necesarios para el entrenamiento de las redes con aceleración por tarjeta de video (GPU) de escritorio y para las pruebas de inferencia en el sistema embebido Jetson TX2. En la tabla 4.1 se muestra las versiones de los recursos de software instalados y usados para este trabajo.

Se configura la funcionalidad con la biblioteca cuDNN (Chetlur y col., 2014) compatible con Tensorflow para tener acceleración por GPU en ambos dispositivos. La tarjeta jetson TX2 se usa en una configuración de consumo eléctrico de 10 vatios.

Tabla 4.1: Versiones de software utilizadas en el desarrollo.

Paquete	PC	Jetson TX2
Sistema operativo	Ubuntu 18 Kernel 5.3	Ubuntu 18 Kernel 5.3
Tensorflow-gpu	2.4	2.2
CUDA	11.2	10.2
cuDNN	7.4.2	7.3
Python	3.8	3.8

Tabla 4.2: Recursos de hardware.

Caracteristíca	Tesla M40	Jetson TX2
Núcleos CUDA	3072 Maxwell	256 Pascal
VRAM	12GB	>8GB*
Tflops	~ 7	~ 1
Bandwidth	288.4 GB/s	59.7 GB/s

Además se debe implemetar el gestor del conjunto de datos de KITTI Depth para su uso en entrenamiento y evaluación de la estimación de la profundidad. Por otro lado, el conjunto de KITTI Object se encontra disponible en la API de ("TensorFlow Datasets", 2021) adecuado para ser usado en el entrenamiento y evaluación de la sub red de localización y clasificación de objetos.

4.3. Funciones de coste y métricas de rendimiento

Para evaluar las arquitecturas propuestas se usan las métricas sugeridas por autores de los *benchmarks* de KITTI (Geiger y col., 2012) para sus respectivas competencias de estimación de la profundidad (Eigen, Puhrsch & Fergus, 2014) y de detección de objetos.

Las métricas para la evaluación del rendimiento en la estimación de la profundidad son:

1. El error logarítmico invariante a la escala (SiLog).

$$SiLog = \frac{1}{n}\sum_i d_i^2 - \frac{1}{n^2}(\sum_i d_i)^2 \tag{4.1}$$

con:

$$d_i = log\ y_{gt} - log\ y_{pred} \tag{4.2}$$

Donde, y_{gt} e y_{pred} son las imagenes de n pixeles i indexados de valores observados y predichos, respectivamente.

2. El error cuadrático relativo (sqErrorRel) en porcentaje.

$$sqErrorRel = \frac{1}{n}\sum_i \frac{(y_{gt} - y_{pred})^2}{y_{gt}^2} \tag{4.3}$$

3. El error absoluto relativo (absErrorRel) en porcentaje.

$$absErrorRel = \frac{1}{n}\sum_i \frac{(y_{gt} - y_{pred})}{y_{gt}} \tag{4.4}$$

4. La raiz cuadrada del error cuadrático medio de las inversas (iRMSE) en Km^{-1}.

$$iRMSE = \sqrt{\frac{1}{n}\sum_i (y'_{gt} - y'_{pred})^2} \tag{4.5}$$

Donde, y'_{gt} e y'_{pred} son inversas de y_{gt} e y_{pred}, respectivamente.

En el entrenamiento de la red MobileNetV2-Unet, la función de pérdida es el error cuadrático medio:

$$MSE = \frac{1}{n}\sum_i (y'_{gt} - y'_{pred})^2 \tag{4.6}$$

En el entrenamiento de la Faster-RCNN, se minimiza una función de pérdida ($Loss$) multitarea, como:

$$L = L_{cls} + L_{box} \tag{4.7}$$

Donde, L_{cls} y L_{box} son las funciones de pérdida para la estimación de la clase y de estimación de las dimensiones de la ventana ($anchor$) en cuestión, respectivamente.

La equación 4.7 es de la siguiente forma:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum L_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{box}} \sum p_i^* \cdot L_1^{smooth}(t_i - t_t^*) \tag{4.8}$$

Donde, i es el índice del *anchor*, p_i es la probabilidad de que el *anchor* sea un objeto. El fondo verdadero p_i^* es uno si la ventana es positiva o cero si es negativa. t_i es un vector que representa las coordinadas parametrizadas del *Bounding Box* predicho. Y t_i^* es el *boundig box* asociado con el *anchor* positivo. L_{cls} es la función de pérdida de entropía cruzada para clasificación binaria para la detección del objeto contra fondo y landa es un parametro de balanceo.
Se tiene que:

$$L_{cls}(p_i, p_i^*) = -p_i^* log p_i - (1 - p_i^*) log(1 - p_i) \tag{4.9}$$

y:

$$L_1^{smooth}(x) = \begin{cases} 0.5x^2, & \text{sí } |x| < 1. \\ |x| - 0.5, & \text{caso contrario.} \end{cases} \tag{4.10}$$

Las métricas para la evaluación del rendimiento en la localización y clasificación de objetos:

1. La intersección sobre la unión (IoU).

$$IoU = \frac{\text{Área del traslape}}{\text{Área de la unión}} \tag{4.11}$$

2. Matriz de confusión.

   La matriz de confusión se conforma de 4 elementos:

   Verdaderos Positivos (VP) : Es cuando el valor real es positivo y la predicción indica que es positivo.
   Verdaderos Negativos (VN) : Es cuando el valor real es negativo y la predicción indica que es negativo.
   Falsos Negativos (FN) : Es cuando el valor real es negativo y la predicción indica que es positivo.
   Falsos Positivos (FP) : Es cuando el valor real es positivo y la predicción

Figura 4.1: Representación de la matriz de confusión. Fuente: (Barrios, 2021)

indica que es negativo.

Entonces, con dichos valores se calculan las siguientes métricas:

a) La Precisión (*Precision*):

$$\text{Precisión} = \frac{VP}{VP + FP} \tag{4.12}$$

b) La Exactitud (*Accuracy*) :

$$\text{Exactitud} = \frac{(VP + VN)}{(VP + FP + FN + VN)} \tag{4.13}$$

c) La Sensibilidad (*Recall*) :

$$\text{Sensibilidad} = \frac{VP}{VP + FN} \tag{4.14}$$

d) La Especificidad ($Especificity$):

$$\text{Especificidad} = \frac{VN}{VN + FP} \tag{4.15}$$

4.4. Implementación de los modelos

La red propuesta resulta de la unión de dos redes, la primera es una red tipo encoder-decoder similar a la FastDepth, su encoder es de tipo MobilenetV2 y el decoder es tipo Unet con *skip conections* de concatenación entre sus capas. Esta, MobileNetV2-Unet realiza la estimación del mapa de profundidad, usa como base una MobileNetV2 previamente entrenada para la clasificación de objetos con el conjunto de datos de ImageNet (Deng y col., 2009), que es entrenada con el conjunto de datos de KITTI Depth para la tarea de estimación del mapa de profundidad.

La arquitectura de los bloques de construcción de la MobileNetV2-Unet se enlistan en la tabla 4.3, tiene un encoder conformado por bloques residuales inversos estandar de MobileNetV2, seguido de un encoder con bloques *Upsampling* que está formado de una convolución 2D transpuesta, seguido de una capa de *Batch normalization* y termina por una capa de activación ReLu. Además, tiene cuatro *skip conections* de concatenación, entre los bloques de salida de los *bottlenecks* y los *Upsampling*.

Por otra parte, la segunda red es una Faster R-CNN con modelo base (*Backbone*) de MobilenetV2. Esta se encarga de realizar las tareas de localización y clasificación de objetos. Y es entrenada con el conjunto de datos de KITTI Object para las tareas de localización y reconocimineto de objetos.

La arquitectura resultante, es la propuesta que se basa en el hecho que las dos redes anteriores tienen un encoder de tipo MobileNetV2. De tal forma que el diseño propuesto consiste en unir las dos redes por el encoder y usar los mapas de características de la salida de este encoder como las entradas a los decoder de cada sub-red, manteniendo la configuración de los *skip conections*.

En la figura 4.2 se muestra un digrama que representa el flujo de datos del modelo propuesto. Se representa que el encoder tipo MobilenetV2 está conectado con el decoder de cada sub-red, justo en el último *bottleneck* de creación de mapas de caracteristícas en lo que antes era el final del encoder del diseño original.

El decoder de la Faster R-CNN consiste en una RPN que está conectado al mapa de caracteristícas, salida del encoder de la MobileNetV2. La RPN consiste en una convolución 2d 3x3 de 512 filtros, seguida de dos convoluciones en paralelo, una de regresión de las coordinadas de los *bounding boxes* de las

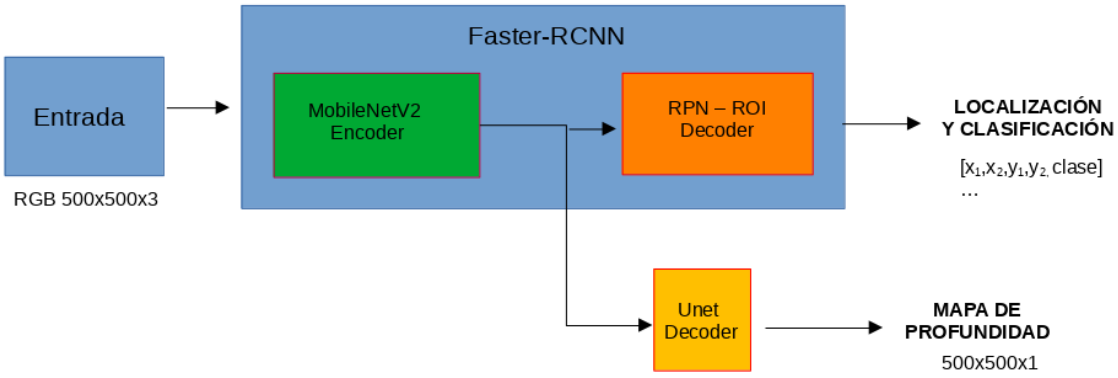


Figura 4.2: Ilustración de la red propuesta. Fuente: propia.

regiones propuestas y una segunda para la clasificación entre existencia de objeto vs fondo. Entonces, la RPN proporciona las propuestas de las áreas de intéres (*RoIs*) para la parte de la Fast R-CNN, que consta de una capa de *pooling*, que extrae nuevos mapas de características para terminar por unas capas densas de estimación de la clase y de regresión de las coordinadas de los *bounding boxes* de los objetos detectados.

Por otro lado, se implementa la misma propuesta pero sustituyendo la red Faster R-CNN por una red SSD300 con *backbone* MobileNetV2, siguiendo una metodología similar.

4.4.1. Estrategia de entrenamiento e inferencia

La estrategia de entrenamiento para las dos redes, consiste primero, en utilizar un *Backbone* de MobileNetV2 pre-entrenada, en el entrenamiento de la red Unet-MobileNetV2, usando el conjunto de datos de KITTI Depth para la estimación de la profundidad.

Entonces, los hiper-parámetros del encoder de la MobileNetV2 son transferidos al encoder que es parte de la red Faster-RCNN-MobileNetV2 y que es entrenada para la tarea de clasificación y localización de objetos con el conjunto de datos de KITTI Object, manteniendo los parámetros del encoder inmutables durante dicho entrenamiento.

Finalmente, se ensambla el modelo propuesto, con los hiper-parámetros del encoder de la MobileNetV2 y de los dos decoders y se realizan pruebas de inferencia cumpliendo con el flujo de datos propuesto. Todos los entrenamientos

Tabla 4.3: Componentes de la red MobileNetV2-Unet, n es el número de repeticiones del bloque.

No.	Entrada	Operación	Canales	n
1	$500^2 \times 3$	conv2d	32	1
2	$250^2 \times 32$	*bottleneck*	16	1
3	$250^2 \times 16$	*bottleneck*	24	2
4	$125^2 \times 24$	*bottleneck*	32	3
5	$63^2 \times 32$	*bottleneck*	64	4
6	$32^2 \times 64$	*bottleneck*	96	3
7	$32^2 \times 96$	*bottleneck*	160	3
8	$16^2 \times 160$	*bottleneck*	320	1
9	$32^2 \times 512$	*Upsample*	512	1
10	$64^2 \times 256$	*Upsample*	256	1
11	126×128	*Upsample*	128	1
12	$250^2 \times 64$	*Upsample*	64	1
13	$500^2 \times 1$	conv2d_T	1	1

emplean el optimizador Adam (Kingma & Ba, 2014).

En resumen, para el entrenamiento de las dos redes e inferencia como el modelo propuesto, consiste en:

1. Entrenamiento de la MobileNetV2-Unet para estimación de profundidad.
2. Transferencia de hiper-parámetros del encoder de MobileNetV2-Unet, al encoder de la red Faster-RCNN-MobileNetV2.
3. Entrenamiento de la red Faster-RCNN-Mobilenetv2 para la localización y clasificación de objetos, con encoder inmutable.
4. Ensamble del modelo por tranferencia de aprendizaje y realización de las pruebas de rendimiento.

4.4.2. Pruebas de rendimiento

Para las pruebas de desempeño se realiza el entrenamiento de la red Unet-MobileNetV2 con KITTI Depth usando un sub conjunto de pares de 4 mil

imágenes RGB con sus correspondientes mapas de profundidad, se dividió en 80 % para entrenamiento, 20 % prueba. Durante el entrenamiento se realiza una mezcla para evitar pasar en secuencia las escenas del conjunto. Y se evalua el rendimiento con otro sub conjunto de mil imágenes.

Acto seguido, se realiza la estrategia de transferencia de aprendizaje al encoder de la red Faster-RCNN-MobileNetV2. Luego, se realiza el entrenamiento con KITTI Object usando un sub conjunto de 4 mil imágenes con anotaciones de clases y *bounding boxes* de objetos, se dividió en 80 % para entrenamiento, 20 % prueba y se utiliza otro sub conjunto de mil imágenes con anotaciones para la evaluación de rendimiento. En el caso de la SSD300 se entrena y prueba con un el mismo sub conjunto.

Una vez se emsambla el sistema, se procede a realizar las pruebas de rendimiento de la inferencia. Estas consisten en la ejecución con otro sub conjunto de validación y se generan resultados de la métricas de desempeño para las tres tareas del sistema.

Capítulo 5

Resultados

En este capítulo se muestran los resultados de las pruebas de evaluación de desempeño de las redes y del sistema ensamblado siguiendo lo descrito en el capítulo anterior. Se comienza por evaluar las dos redes por separado, cumpliendo con sus diseños iniciales. Posteriormente, se evalua el rendimiento del modelo propuesto ensamblado, cumpliendo con la estrategia de transferencias de aprendizaje y del flujo de datos propuesto.

5.1. Estimación de la profundidad

Se muestran los resultados de las pruebas de evaluación de desempeño de la red MobileNetV2-Unet. Dicha red se conforma de un total de 6.5 millones de hiper-parámetros de los cuales 6.4 millones son entrenables. La configuración de entrenamiento fue un factor de aprendizaje inicial de $1x10^{-3}$, con un tamaño de *batch* de 12 imagenes, por unas 100 épocas.

En la figura 5.2 se muestra una imagen de entrada y su mapa de profundidad predicho por la red. En la tabla 5.1 se muestra el resultado de la evaluación de rendimiento de solo la foto de la figura 5.2. Y en la tabla 5.2 se muestra el resultado promediado de la evaluación de rendimiento para todo el sub conjunto de evaluación.

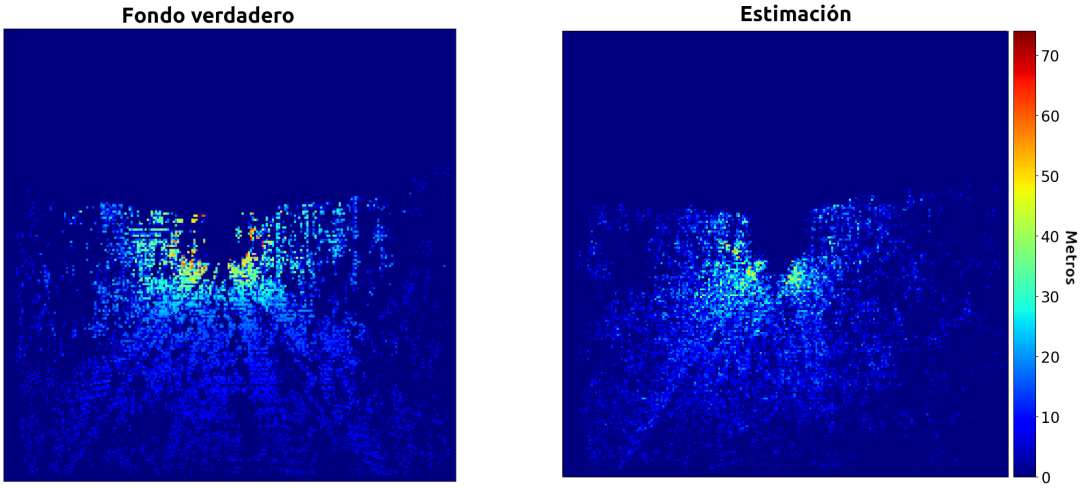


Figura 5.1: Comparación del fondo verdadero de la profundidad y la salida predicha por la sub red de profundidad. Entrenamiento sin interpolación o filtro reductor de ruido aplicado. Y coloreada de azul a rojo como representación de la distancia. Fuente: Propia.

Tabla 5.1: Resultados estimación de profundidad para la imagen de la figura 5.1, con la red MobileNetV2-Unet. Fuente: Propia.

SILog	sqErrorRel	absErrorRel	iRMSE
26	36	45	11

Tabla 5.2: Resultados promediados de estimación de la profundidad para todo el sub-conjunto de evaluación, con la red MobileNetV2-Unet. Fuente: Propia.

SILog	sqErrorRel	absErrorRel	iRMSE
26	58	68	46

5.2. Localización y clasificación de objetos

Se muestran los resultados de las pruebas de evaluación de desempeño de la red Faster-RCNN-MobileNetV2 en versión separada y ensamblada, habiendo aplicado la metodología propuesta. Dicha red se conforma de un total de 135 millones de hiper-parámetros. La configuración de entrenamiento fue un factor de aprendizaje inicial de $1x10^{-3}$, un tamaño de *batch* de 4, por unas 20 épocas.

En las figuras 5.3 y 5.4 se muestra una imagen con las anotaciones de las cuadriculas de reconocimiento de los objetos.

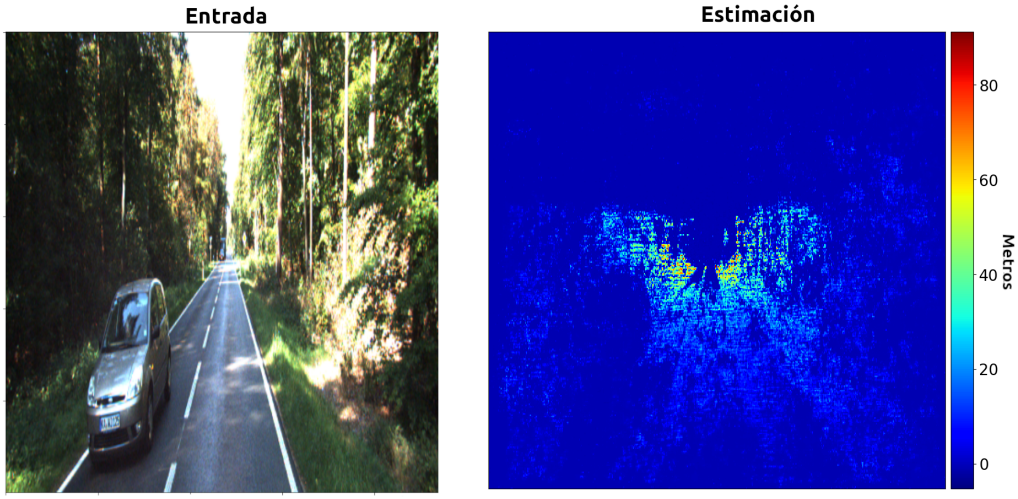


Figura 5.2: Imagen de entrada y salida de de la red de profundidad, izquierda y derecha respectivamente. Fuente: Propia.

Figura 5.3: Imagen de entrada con las anotaciones de *bounding boxes* y probabilidad de clases inferido por la Faster-RCNN MobileNetV2. Fuente: Propia.

Figura 5.4: Imagen de entrada con las anotaciones de *bounding boxes* y probabilidad de clases inferido por la Faster RCNN MobileNetV2. Fuente: Propia.

Tabla 5.3: Resultados de evaluación de la red Faster R-CNN de clasificación y localización de objetos en modo separado. Fuente: Propia.

Clase	Precisión media (AP)	Sensibilidad
Car	0.24	0.2
Van	0.12	0.08
Truck	0.02	0.02
Pedestrian	0.25	0.18

Se muestran los resultados de las pruebas de evaluación de desempeño de la red SSD300-MobileNetV2 en versión separada y ensamblada, habiendo aplicado la metodología propuesta. Dicha red se conforma de un total de 6.8 millones de hiper-parámetros. La configuración de entrenamiento fue un factor de aprendizaje inicial de $1x10^{-3}$, un tamaño de *batch* de 16 imagenes, por unas 100 épocas. En la tabla 5.8 se muestra la evolución del error contra las épocas durante el entrenamiento en separado y en la tabla 5.5 está la precisión y sensibilidad alcanzados en dicho entrenamiento.

Finalmente, se presenta el resultado de la inferencia con la red propuesta ya ensamblada y habiendo aplicado la estrategia de transferencia aprendizaje, se procedió a realizar la evaluación de rendimiento para la tarea de clasificación y localización.

El entrenamiento de sistema ensamblado para las tareas de localización y clasificación de objetos y localización de objetos, entrenado con 10 épocas y de más parámetros de entrenamiento similares a su versión separada, logró los

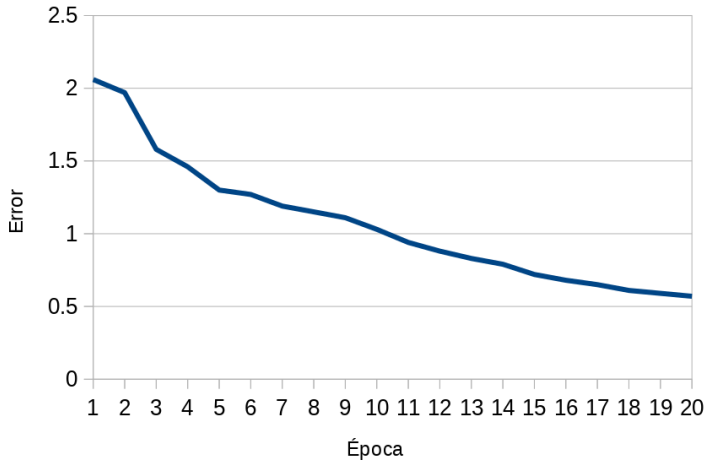


Figura 5.5: Evolución del promedio de los errores de la Faster R-CNN separada. Fuente: Propia.

Tabla 5.4: Resultados de evaluación de la red Faster R-CNN de clasificación y localización de objetos ensamblada. Fuente: Propia.

Clase	Precisión media (AP)	Sensibilidad
Car	0.04	0.01
Pedestrian	0.02	0

resultados que se muestran el la tabla 5.4.

En cuanto a los tiempos de ejecución de la inferencia de las redes en el sistema de desarrollo que cuenta con la GPU, la red MobilenetV2-Unet se ejecuta a una tasa de 18 fps, la Faster R-CNN infiere a una velocidad de 11 fps.

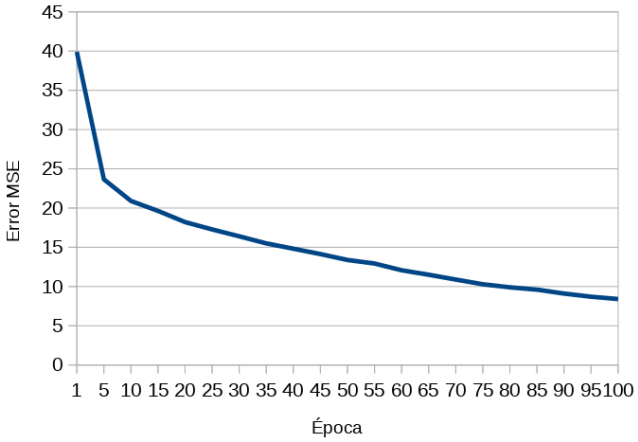


Figura 5.6: Evolución del error cuadrático medio (*MSE*) de la MobileNetV2-Unet. Fuente: Propia.

Tabla 5.5: Resultados de evaluación de la red SSD300 de clasificación y localización de objetos separada. Fuente: Propia.

Clase	Precisión media (AP)	Sensibilidad
Car	0.25	0.19
Van	0.13	0.02
Truck	0.11	0.16

Figura 5.7: Imagen muestra de inferencia de la red SSD300 de clasificación y localización de objetos separada. Fuente: Propia.

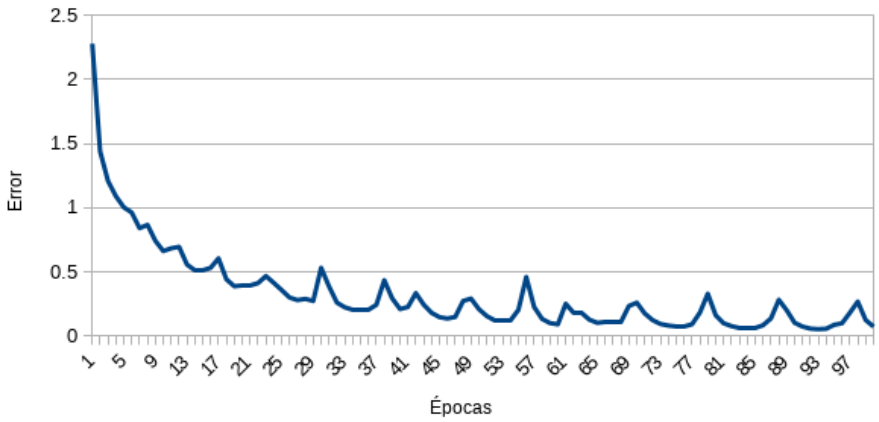


Figura 5.8: Evolución del error de la SSD300 separada. Fuente: Propia.

5.3. Discusión

Se mostraron los resultados de los entrenamientos de las redes por separado y el resultado de la inferencia en el modelo propuesto para la tarea de clasificación y localización de objetos. En cuanto a los hiper-parámetros del sistema propuesto, la estrategia con Faster R-CNN reduce un total de 6 millones, sin afectar el resultado del rendimiento para la tarea estimación de la profundidad, debido a la estrategia de inferencia propuesta. Maś del 60 porciento de esos hiper-parámetros están en las capas densas de la etapas de la RPN y RoI de la Faster R-CNN.

Para la tarea de profundidad se obtuvo un error Silog de 26, en comparación en la competencia de KITTI Depth Prediction está publicada información desde el Top 46 que inicia en 17,9. En la tarea de clasificación y localización de objetos, la Faster R-CNN en su entrenamiento por separado consiguío una precisición media de 0.25 AP en la clase de autos, mientras en el benchmark se plantea un mínimo de 0.5 AP, se presenta mayor sensibilidad y precisión para la detección de la clase autos, probablemente debido a la discribución de clases de la base de datos de KITTI object.

Probablemente es posible mejorar el rendimiento de la red Faster R-CNN y la SSD300 con entrenamientos de más épocas y llevando a cabo una estrategia de aumento de datos. Debido a que la red SSD obtuvo baja presición para la detección de elementos pequeños, en particular para la clase Peatones, esto debido principalmente a causa de la distribución de dicha clase en la base de datos y de la estrategia de entrenamiento ejecutada.

Si bien, en el sistema ensamblado para la clasificación y estimación de objetos con la Faster R-CNN, entrenado usando el encoder generado durante el entrenamiento de la estimación de la profundidad con la configuración de parámetros mencionado, solo logró un 4 porciento de precisión para la clase autos, que es insuficiente para su uso, pero si sugiere que se puede hacer entrenamiento y obtener información de dicho encoder para esta tarea, lo que puede entenderse como que dichos mapas generados no son tan buenos para llevar acabo esa tarea.

Conclusión

En este trabajo, se propusó un modelo basado en dos redes neuronales convolucionales, ambas con encoders de tipo MobileNetV2, la primera con un decoder de tipo Unet para la estimación del mapa de profundidad y la segunda con decoder de la Faster R-CNN para la localización y clasificación de objetos. También, se utilizó el decoder de la SSD300. La entrada del sistema es una imagen RGB monocular desde la cual se infieren dos salidas, la primera un mapa de profundidad con un canal de valores que representan de la distancia estimada en metros y la segunda salida es una tupla de estimación de cuadros delimitadores asociados a una clase detectada.

También, se propusó una estrategia de entrenamiento y flujo de información en el modelo para su inferencia con ambas redes neuronales convolucionales, que consiste en realizar el entrenamiento por separado en cada una de las dos redes en su correspondiente tarea y entonces realizar la transferencia de aprendizaje y ensamblar el modelo en un sistema de inferencia.

El objetivo principal fue disminuir el uso de los recursos de cómputo, al reducir la cantidad de hiper-parámetros por tener un solo encoder compartido y medir su influencia en el rendimiento. El principal incoveniente fue la baja precisión obtenida con el ensamble y estrategia propuesto, es muy probable que mejorando el desempeño de las redes base del ensamble, incremente la presición del mismo. Como trabajo futuro, se puede mejorar la estrategia de entrenamiento, con una etapa de aumento de datos, entrenar con otros conjuntos de datos diferentes a KITTI, como por ejemplo, NYU Depth. E implementar mejoras en la sub-red de clasificación y localización de objetos, como uso de una red SSD de mayor resolución y ampliar el alcance de la estrategia para el entrenamiento del encoder de la red ensamblada.

Apéndice A

Anexo 1

En este apartado, se muestran las partes más relevantes del código de la definición de los modelos de las CNN utilizadas en este trabajo.

```
def Model_5(input_shape=(500, 500,3)):

    mobilenet_model = keras.applications.MobileNetV2(input_shape=input_shape,
                                                      include_top=False)
    block_1 = mobilenet_model.get_layer('block_1_expand_relu').output
    block_3 = mobilenet_model.get_layer('block_3_expand_relu').output
    block_6 = mobilenet_model.get_layer('block_6_expand_relu').output
    block_13 = mobilenet_model.get_layer('block_13_expand_relu').output
    block_16 = mobilenet_model.get_layer('block_16_project').output

    up_16 = UpSampleBlock(512)(block_16)
    merge_16 = keras.layers.Concatenate()([up_16, block_13])

    up_17 = UpSampleBlock(256)(merge_16)
    merge_17 = keras.layers.Concatenate()([up_17[:,0:63,0:63,:], block_6])

    up_18 = UpSampleBlock(128)(merge_17)
    merge_18 = keras.layers.Concatenate()([up_18[:,0:125,0:125,:], block_3])

    up_19 = UpSampleBlock(64)(merge_18)
    merge_19 = keras.layers.Concatenate()([up_19, block_1])
    output = keras.layers.Conv2DTranspose(1, 3, strides=2, padding='same',
                                          activation='linear')(merge_19)

    model = keras.Model(inputs=mobilenet_model.input, outputs=output)

    return model
```

Figura A.1: Código de definición de la red MobileNetV2-Unet de estimación de profundidad en Keras Python. Fuente: Propia.

```python
base_model = MobileNetV2(include_top=False, input_shape=(img_size, img_size, 3))
feature_extractor = base_model.get_layer("block_13_expand_relu")
output = Conv2D(512, (3, 3), activation="relu", padding="same", name="rpn_conv")(feature_extractor.output)
rpn_cls_output = Conv2D(hyper_params["anchor_count"], (1, 1), activation="sigmoid", name="rpn_cls")(output)
rpn_reg_output = Conv2D(hyper_params["anchor_count"] * 4, (1, 1), activation="linear", name="rpn_reg")(output)
rpn_model = Model(inputs=base_model.input, outputs=[rpn_reg_output, rpn_cls_output])
return rpn_model, feature_extractor
```

Figura A.2: Código de definición de la RPN para la red Faster-RCNN en Keras Python. Fuente: Propia.

```python
input_img = rpn_model.input
rpn_reg_predictions, rpn_cls_predictions = rpn_model.output
input_gt_boxes = Input(shape=(None, 4), name="input_gt_boxes", dtype=tf.float32)
input_gt_labels = Input(shape=(None, ), name="input_gt_labels", dtype=tf.int32)
rpn_cls_actuals = Input(shape=(None, None, hyper_params["anchor_count"]), name="input_rpn_cls_actuals", dtype=tf.float32)
rpn_reg_actuals = Input(shape=(None, 4), name="input_rpn_reg_actuals", dtype=tf.float32)
frcnn_reg_actuals, frcnn_cls_actuals = RoIDelta(hyper_params, name="roi_deltas")(
                                        [roi_bboxes, input_gt_boxes, input_gt_labels])

loss_names = ["rpn_reg_loss", "rpn_cls_loss", "frcnn_reg_loss", "frcnn_cls_loss"]
rpn_reg_loss_layer = Lambda(train_utils.reg_loss, name=loss_names[0])([rpn_reg_actuals, rpn_reg_predictions])
rpn_cls_loss_layer = Lambda(train_utils.rpn_cls_loss, name=loss_names[1])([rpn_cls_actuals, rpn_cls_predictions])
frcnn_reg_loss_layer = Lambda(train_utils.reg_loss, name=loss_names[2])([frcnn_reg_actuals, frcnn_reg_predictions])
frcnn_cls_loss_layer = Lambda(train_utils.frcnn_cls_loss, name=loss_names[3])([frcnn_cls_actuals, frcnn_cls_predictions])

frcnn_model = Model(inputs=[input_img, input_gt_boxes, input_gt_labels,
                            rpn_reg_actuals, rpn_cls_actuals],
                    outputs=[roi_bboxes, rpn_reg_predictions, rpn_cls_predictions,
                             frcnn_reg_predictions, frcnn_cls_predictions,
                             rpn_reg_loss_layer, rpn_cls_loss_layer,
                             frcnn_reg_loss_layer, frcnn_cls_loss_layer])
```

Figura A.3: Código de definición de la red Faster-RCNN, parte de la Fast R-CNN en Keras Python. Fuente: Propia.

```python
def get_model(hyper_params,img_size):
    #Backbone
    base_model = MobileNetV2(include_top=False, input_shape=(img_size, img_size, 3))

    input = base_model.input
    first_conv = base_model.get_layer("block_13_expand_relu").output
    second_conv = base_model.output
    #
    # SSD Extra Feature Layers
    extra1_1 = Conv2D(256, (1, 1), strides=(1, 1), padding="valid", activation="relu", name="extra1_1")(second_conv)
    extra1_2 = Conv2D(512, (3, 3), strides=(2, 2), padding="same", activation="relu", name="extra1_2")(extra1_1)
    #
    extra2_1 = Conv2D(128, (1, 1), strides=(1, 1), padding="valid", activation="relu", name="extra2_1")(extra1_2)
    extra2_2 = Conv2D(256, (3, 3), strides=(2, 2), padding="same", activation="relu", name="extra2_2")(extra2_1)
    #
    extra3_1 = Conv2D(128, (1, 1), strides=(1, 1), padding="valid", activation="relu", name="extra3_1")(extra2_2)
    extra3_2 = Conv2D(256, (3, 3), strides=(2, 2), padding="same", activation="relu", name="extra3_2")(extra3_1)
    #
    extra4_1 = Conv2D(128, (1, 1), strides=(1, 1), padding="valid", activation="relu", name="extra4_1")(extra3_2)
    extra4_2 = Conv2D(256, (3, 3), strides=(2, 2), padding="same", activation="relu", name="extra4_2")(extra4_1)

    # SSD Heads
    pred_deltas, pred_labels = get_head_from_outputs(hyper_params, [first_conv, second_conv, extra1_2, extra2_2, extra3_2, extra4_2])

    return Model(inputs=input, outputs=[pred_deltas, pred_labels])
```

Figura A.4: Código de definición de la red SSD300 en Keras Python. Fuente: Propia.

Bibliografía

Alhashim, I. & Wonka, P. (2018). High Quality Monocular Depth Estimation via Transfer Learning. arXiv. doi:10.48550/ARXIV.1812.11941

Barrios, J. (2021). Matriz de confusión. 04.03.2022. Recuperado desde https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/

Bergasa, L. M., Arroyo, R., Romera, E. & Alvarez, M. (2018). ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 1-10. Recuperado desde https://github.com/Eromera/erfnet

Bokovoy, A., Muravyev, K. & Yakovlev, K. (2019). Real-time Vision-based Depth Reconstruction with NVidia Jetson. arXiv: 1907.07210. Recuperado desde http://arxiv.org/abs/1907.07210

Cai, Z. & Vasconcelos, N. (2019). Cascade R-CNN: High Quality Object Detection and Instance Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1-1. arXiv: 1906.09756. Recuperado desde http://arxiv.org/abs/1906.09756

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B. & Shelhamer, E. (2014). cuDNN: Efficient Primitives for Deep Learning. arXiv. doi:10.48550/ARXIV.1410.0759

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. En *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.

Eigen, D., Puhrsch, C. & Fergus, R. (2014). Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. arXiv. doi:10.48550/ARXIV.1406.2283

Geiger, A., Lenz, P. & Urtasun, R. (2012). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. En *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Girshick, R. (2015). Fast R-CNN. arXiv. doi:10.48550/ARXIV.1504.08083

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv. doi:10.48550/ARXIV.1311.2524

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation, 580-587. doi:10.1109/CVPR.2014.81. arXiv: 1311.2524

Hazirbas, C., Ma, L., Domokos, C. & Cremers, D. (2016). FuseNet: incorporating depth into semantic segmentation via fusion-based CNN architecture. En *Asian Conference on Computer Vision*.

He, K., Gkioxari, G., Dollár, P. & Girshick, R. (2020). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(2), 386-397. doi:10.1109/TPAMI.2018.2844175. arXiv: 1703.06870

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Vol. 2016-Decem, pp. 770-778). IEEE Computer Society. doi:10.1109/CVPR.2016.90. arXiv: 1512.03385

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv. doi:10.48550/ARXIV.1704.04861

Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv. doi:10.48550/ARXIV.1502.03167

Kim, S.-h. & Hwang, Y. (2021). A Survey on Deep Learning Based Methods and Datasets for Monocular 3D Object Detection. *Electronics*, *10*(4). doi:10.3390/electronics10040517

Kingma, D. P. & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv. doi:10.48550/ARXIV.1412.6980

LeCun, Y. [Y.], Boser, B., Denker, J. S., Henderson, D., Howard, R., Hubbard, W. & Jackel, L. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*, 541-551.

LeCun, Y. [Yann], Bengio, Y. & Hinton, G. E. (2015). Deep learning. *Nat*, *521*(7553), 436-444.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. & Belongie, S. (2016). Feature Pyramid Networks for Object Detection. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, *2019-Novem*, 1702-1707. arXiv: 1612.03144. Recuperado desde http://arxiv.org/abs/1612.03144

Liu, S. & Liu, Z. (2017). Multi-Channel CNN-based Object Detection for Enhanced Situation Awareness. arXiv. doi:10.48550/ARXIV.1712.00075

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. & Berg, A. C. (2016a). SSD: Single Shot MultiBox Detector. En B. Leibe, J. Matas, N. Sebe & M. Welling (Eds.), *Computer Vision – ECCV 2016* (pp. 21-37). Cham: Springer International Publishing.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. & Berg, A. C. (2016b). SSD: Single Shot MultiBox Detector. En *Computer Vision – ECCV 2016* (pp. 21-37). Springer International Publishing. doi:10.1007/978-3-319-46448-0_2

Liu, Y., Cao, S., Lasang, P. & Shen, S. (2019). Modular Lightweight Network for Road Object Detection Using a Feature Fusion Approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 1-13. doi:10.1109/tsmc.2019.2945053

Liu, Y., Cao, S., Lasang, P. & Shen, S. (2021). Modular Lightweight Network for Road Object Detection Using a Feature Fusion Approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *51*(8), 4716-4728. doi:10.1109/TSMC.2019.2945053

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. Recuperado desde https://www.tensorflow.org/

MATLAB. (2021). Convolutional neural network by matlab. 05.11.2021. Recuperado desde https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html

NVIDIA. (2019). Jetson TX2 Module Specifications. 05.10.2021. Recuperado desde https://developer.nvidia.com/embedded/jetson-tx2

Paszke, A., Chaurasia, A., Kim, S. & Culurciello, E. (2016). ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation, 1-10. arXiv: 1606.02147. Recuperado desde http://arxiv.org/abs/1606.02147

PulkitS. (2018). Object detection guide. 14.02.2022. Recuperado desde https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 779-788. doi:10.1109/CVPR.2016.91. arXiv: arXiv:1506.02640v5

Ren, S., He, K., Girshick, R. & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137-1149. doi:10.1109/TPAMI.2016.2577031. arXiv: 1506.01497

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv. doi:10.48550/ARXIV.1505.04597

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386-408. doi:10.1037/h0042519

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4510-4520. doi:10.1109/CVPR.2018.00474. arXiv: 1801.04381

Santana, A. M. & Medeiros, A. A. D. (2009). Simultaneous Localization and Mapping (SLAM) of a Mobile Robot Based on Fusion of Odometry and Visual Data Using Extended Kalman Filter. En *Contemporary Robotics* (Cap. 8). Rijeka: IntechOpen. doi:10.5772/7801

Shen, X. (2019). A survey of Object Classification and Detection based on 2D/3D data. arXiv. doi:10.48550/ARXIV.1905.12683

slicer, K. (2021). kitti depthmap slicer. Recuperado desde https://github.com/soulslicer/kitti_depthmap

Soviany, P. & Ionescu, R. T. (2018). Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction. arXiv. doi:10.48550/ARXIV.1803.08707

Spek, A., Dharmasiri, T. & Drummond, T. (2018). CReaM: Condensed Real-time Models for Depth Prediction using Convolutional Neural Networks. arXiv: 1807.08931. Recuperado desde http://arxiv.org/abs/1807.08931

TensorFlow Datasets. (2021). https://www.tensorflow.org/datasets. 07.2021. Recuperado desde https://www.tensorflow.org/datasets

Wofk, D., Ma, F., Yang, T.-J., Karaman, S. & Sze, V. (2019). FastDepth: Fast Monocular Depth Estimation on Embedded Systems. arXiv: 1903.03273. Recuperado desde http://arxiv.org/abs/1903.03273

Wu, C.-E., Chan, Y.-M., Chen, C.-H., Chen, W.-C. & Chen, C.-S. (2019). IMMVP: An Efficient Daytime and Nighttime On-Road Object Detector. arXiv: 1910.06573. Recuperado desde http://arxiv.org/abs/1910.06573

Wu, H. H., Zhou, Z., Feng, M., Yan, Y., Xu, H. & Qian, L. (2019). Real-time single object detection on the UAV. En *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019* (pp. 1013-1022). Institute of Electrical y Electronics Engineers Inc.

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., ... Adam, H. (2018). NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. arXiv. doi:10.48550/ARXIV.1804.03230