ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

ACELERACIÓN EN GPU DEL PROCESAMIENTO DE VÓXELES EN TIEMPO REAL

Realizado por
Fernando Gallego Donoso
Tutorizado por
Manuel Ujaldón Martínez
Departamento
Arquitectura de Computadores

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2019

Fecha de defensa:
El Secretario del Tribunal

Resumen

En la actualidad, existen muchos sistemas que necesitan tratar una gran cantidad de datos en un tiempo muy bajo. Los sistemas de procesamiento de datos en tiempo real son uno de los más relevantes. Este sistema necesita procesar vóxeles lo más rápido posible para poder emitir una respuesta inmediata. Con el avance de la tecnología, los datos se han incrementado también, por lo que ha surgido la necesidad de nuevas técnicas para poder tratar dichos datos. En este proyecto se estudiarán las distintas redes neuronales, que destaquen por su alto rendimiento, y su paralelización en las distintas GPUs. Concretamente, el sistema que se va a procesar es una red neuronal convolucional (CNN).

Las redes neuronales seleccionadas son las que destacan por mayor rendimiento a la hora de procesar vídeo en tiempo real. Por último, se han utilizado diferentes tipos de hardware para analizar el rendimiento y la eficiencia de las técnicas desarrolladas en este trabajo, como NVIDIA GTX 960, NVIDIA Jetson TX2 y NVIDIA Jetson Nano.

Palabras clave

- CUDA.
- RED NEURONAL.
- MULTIPROCESAMIENTO.
- APRENDIZAJE PROFUNDO.
- PROCESADOR GRÁFICO.

Abstract

Currently, most systems need to process large amounts of data in a very short time. Real-time data processing systems are one of the most relevant. This system needs to process voxels as fast as possible in order to calculate, and apply, the results in real time. Driven by technological improvements, the data to be processed has increased, so, in order to process in real time the required amounts of data, new techniques are needed. In this work, a new technique will be presented, based upon the acceleration the need for new techniques has arisen to be able to process such data. The technique that will be presented in this project is the acceleration of the system through the parallelization of a graphics processor (GPU). Specifically, the system to be processed is a convolutional neural network (CNN).

The selected neural networks are those that stand out for the highest performance when it comes to processing video in real time. Finally, different hardware has been used to analyze the performance and efficiency of the techniques developed in this work, including NVIDIA GTX 960, NVIDIA Jetson TX2 and NVIDIA Jetson Nano.

Keywords

- CUDA.
- CNN.
- GPU.
- MULTIPROCESSING.
- DEEP LEARNING.

Índice general

1 Introducción

1.1. Marco del proyecto

Este Trabajo Fin de Grado surge como necesidad natural dentro de dos proyectos recientemente concedidos por la Universidad de Málaga en la convocatoria 2018 de proyectos Smart Campus, StreetQR y DIAS2P, fusionados finalmente en uno solo para amortizar parte de la dotación económica disponible.

StreetQR presenta un dispositivo que ambiciona mejorar las rotulaciones con el nombre de las calles para incorporar a los carteles un código QR que redirija al usuario a una base de datos que aporte al transeunte o turista toda la información que pueda serle de interés [39].

DIAS2P, por su parte, se basa en un sistema de reconocimiento de coches y peatones en los pasos de cebra que no disponen de semáforo, con el objetivo de alertar a los coches de la presencia de peatones en ellos para reducir el número de

atropellos.

Ambos proyectos componen un sistema de procesamiento de datos en tiempo real, por lo que tienen un cálculo computacional muy exigente. En este TFG se analizarán las necesidades de visión artificial que tiene DIAS2P, dotando al sistema de la habilidad para reconocer objetos a través del análisis de los fotogramas filmados desde una cámara integrada en el cartel informativo del paso de cebra, según prototipo descrito en el proyecto.

Para este reconocimiento, utilizaremos técnicas de aprendizaje profundo mediante la implementación de redes neuronales convolucionales (CNNs - *Convolutional Neural Networks*), y trataremos de proporcionar una respuesta en tiempo real que a la vez cumpla con los requisitos de consumo energético y miniaturización que requiere la especificación inicial del prototipo. Para ello se propone el uso de procesadores gráficos o GPUs (*Graphics Processing Units*), en los que se implementarán dichas redes, bien mediante el uso de librerías disponibles o, en su defecto, realizando una implementación más artesanal utilizando programación CUDA que permita paralelizar la ejecución sobre los miles de procesadores disponibles.

Con objeto de contrastar rendimiento, consumo energético y miniaturización, se estudiarán diversas plataformas para abordar la implementación, realizando una comparativa final de todos estos aspectos.

1.2. Hoja de ruta

A la hora de llevar a cabo nuestro trabajo debemos acomodar la consecución de objetivos ya señalados a la agenda temporal descrita en la propuesta inicial de los proyectos StreetQR y DIAS2P. Dicha agenda se estructura en cuatro fases, de duración trimestral, al final de cada una de la cuales se obtendrá un hito del proyecto, que se remitirá al Vicerrectorado de Smart-Campus.

1. Creación del grupo de trabajo y asignación de tareas. Realización de las especificaciones técnicas de cada subsistema del dispositivo, y de sus requerimientos de espacio, energía y computación. Diseño industrial del prototipo a partir de las especificaciones y requerimientos establecidos. Marco temporal: Primer trimestre de 2019. Hito: Diseño industrial del prototipo.

2. A partir del diseño industrial obtenido, estudiar los componentes a adquirir y seleccionarlos en base a sus especificaciones y precios. Adquirir dichos componentes. Ensamblar los componentes de los distintos sistemas del dispositi-

vo siguiendo el diseño industrial de partida. Ensamblar los distintos sistemas para obtener un prototipo operativo. Marco temporal: Segundo trimestre de 2019. Hito: Prototipo realizado físicamente.

3. Instalación del prototipo en su ubicación real en la calle. Puesta en marcha. Detección de problemas de funcionamiento y solución de los mismos. Obtención de los primeros datos y ajuste de los parámetros de los algoritmos para afinar su funcionamiento. Marco temporal: Tercer trimestre de 2019. Hito: Prototipo operativo.

4. Seguimiento del funcionamiento del dispositivo en situación real y continua de funcionamiento. Recopilación de datos masivos para su análisis posterior. Detección de mejoras a realizar en la siguiente fase industrial. Marco temporal: Cuarto trimestre de 2019. Hito: Informe final del prototipo, sobre su funcionamiento en el entorno real y las posibles mejoras futuras a realizar.

Este Trabajo Fin de Grado contribuye a las tareas 2 y 3, habiéndose ubicado por tanto a lo largo del segundo trimestre de 2019.

1.3. Motivación

El objetivo que impulsa este proyecto es aportar una combinación de sistema de detección de objetos y tarjeta gráfica que emita una respuesta a tiempo real. En muchas ocasiones los desarrolladores buscan o bien un sistema de detección de objetos, o bien la paralelización de un sistema en diversos dispositivos hardware.

Dado que en gran parte de este tipo de proyectos existen ciertas limitaciones. En este TFG se hará un estudio de las redes neuronales que destaquen por su rendimiento, buscando maximizar el número de imágenes por segundo, y se ejecutarán sobre una serie de procesadores gráficos con limitaciones económicas realistas, analizando su rendimiento, consumo y dimensiones.

De esta manera los desarrolladores que busquen sistemas de detección con estas limitaciones, tendrán a su disposición un listado con los resultados obtenidos. Facilitando así la elección de su sistema y el dispositivo sobre el que desplegarlo.

1.4. Objetivo

El objetivo principal de este Trabajo Fin de Grado consiste en analizar diversos sistemas hardware basados en GPUs y estudiar sobre ellos la implementación del software anteriormente descrito, con objeto de seleccionar la mejor alternativa para su integración dentro del prototipo que emita las correspondientes alertas luminosas y acústicas para minimizar la probabilidad de atropello de un peatón.

Los requerimientos a satisfacer en nuestro planteamiento inicial son los siguientes:

1. Rendimiento de la aplicación para una respuesta en tiempo real. Debe procesar las imágenes captadas por la cámara a un ritmo de 15-20 FPS (*Frames Per Second*).

2. Consumo energético. Adicionalmente, hay que tener en cuenta que el dispositivo propuesto incorpora unas placas solares para la autonomía energética que limita el consumo energético de la aplicación, por lo que un rango estimado de 5-10 W. Sería deseable, y debe ser considerado aunque esto suponga limitar el rendimiento anterior.

3. Dimensiones físicas. El prototipo se integra junto a la señal informativa del paso de peatones, por lo que debe ser lo más pequeño posible. Entre los sistemas propuestos basados en GPUs, existen en la actualidad algunos que priorizan la miniaturización, y serán nuestros predilectos para poder atender también estos requisitos.

2

Contexto y Estado del Arte

Las redes neuronales están diseñadas para simular el funcionamiento del cerebro humano. Existen diferentes tipos de redes neuronales, aunque este proyecto se centrará en las redes neuronales convolucionales *(ver figura* 2.1).

Desde el primer modelo computacional de Alan Turing en 1943, Turing Machine, hasta el día de hoy, las redes neuronales han experimentado un constante desarrollo. En 1958, Frank Rossenblat presentó el "Perceptron", una red de una única capa cuya función de activación era de tipo signo.

Los siguientes grandes avances se encuentran en 1980 y 1992 con la aparición del Neocognitron [10] y el Cresceptron [43] respectivamente.

En el nuevo milenio, fue el momento en el que el término Deep Learning comenzó a aumentar su repercusión gracias a una investigación en la que se entrenaba una red neuronal. Desde entonces, Deep Learning ha estado en continuo desarrollo, sin embargo, no será hasta 2016 cuando obtiene gran relevancia.

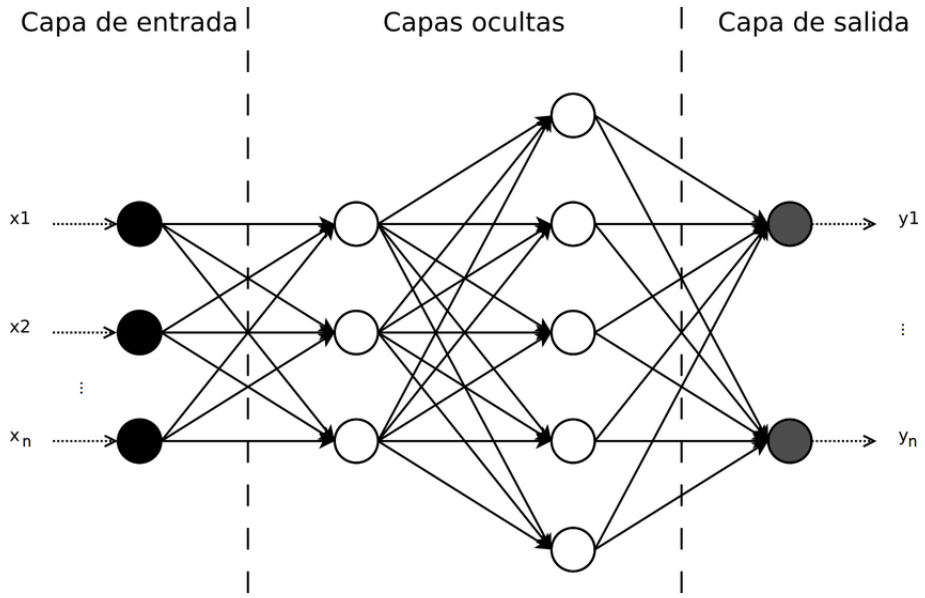


Figura 2.1: *Red neuronal [2].*

En ese año, grandes empresas como Microsoft, Amazon, Google o Facebook comenzaron a apostar por Deep Learning. Esto supuso una oleada de nuevas investigaciones.

Ya en el año 2018, Yosuke Oyama, Tal Ben-Nuny, Torsten Hoeflery y Satoshi Matsuoka desarrollaron una investigación para acelerar los marcos de aprendizaje profundo con Micro-Batching. En ella utilizaron una red neuronal AlexNet y otra ResNet e implementaron cuDNN sobre Caffe para mejorar la eficiencia y reducir los requisitos y TensorFlow para dividir las capas convolucionales en micro-batches. Gracias a esta aceleración consiguieron obtener un factor de aceleración de 1.63x de AlexNet y 1.21x en ResNet [33].

Otra investigación importante y relacionada con este proyecto, que también se realizó en 2018, es la llevada a cabo por Wencong Xiaoy, Cheng Chen, Youshan Miao, Jilong Xue, Ming Wu. En esta publicación, debido a la gran importancia que estaba obteniendo el aprendizaje profundo, los investigadores buscaron las diferentes técnicas de optimización mediante nuevos algoritmos y hardware personalizado. Como la carga de trabajo que supone para un desarrollador incorporar técnicas de optimización específicas a su necesidad es demasiado alta, propusieron una serie de patrones según optimizaciones de modelos específicos ya existentes.

Los resultados preliminares muestran que, en TensorFlow, el uso del mapeo de patrones de optimización podría reemplazar modelos multicapa a corto plazo (LSTM) y paralelizarlo implementando cudNN mejorando con un factor de acelera-

ción de 4.12x [45].

Este mismo año hemos podido encontrar investigaciones tan interesantes para este proyecto como la desarrollada por Alberto S. Garea, Dora B. Heras y Francisco Argüello en la que utilizan aprendizaje profundo, Caffe y cuDNN para la clasificación de imágenes hiperespectrales en GPU [11].

Otra investigación de 2019 que está muy relacionada con este proyecto es "Fast Algorithm of Unified Layer Performing Convolution and Average Pooling on the GPU"[41], desarrollada por Hiroki Tokura, Takahiro Nishimura, Yasuaki Ito, Koji Nakano, Akihiko Kasagi y Tsuguchika Tabaru. Este grupo presenta un nuevo filtro al que llaman "fused filter"que reduce las operaciones de punto flotante para el cálculo de la capa convolucional y lo aceleran con cuDNN en una NVIDIA V100 obteniendo un factor de aceleración de 1.8x.

Como se ha podido observar, a día de hoy, Deep Learning y redes neuronales están relacionados siempre con la paralelización en GPUs, y por lo tanto con la programación CUDA en tarjetas gráficas de NVIDIA.

En este TFG, las redes neuronales que se van a utilizar son sistemas de detección de objetos. Un sistema de detección de objeto tiene como objetivo detectar y clasificar objetos dada una imagen o un *stream* a partir de un conjunto de datos y probabilidades que permita a dicho sistema diferenciar el objeto del entorno. Este proyecto se centrará en las redes neuronales convolucionales (CNNs), que describiremos en la sección 2.1.

2.1. Redes neuronales convolucionales

Las CNNs son un tipo de red neuronal artificial en la que cada capa se encarga de una tarea diferente, por lo que el entrenamiento es mucho más rapido y facilita su paralelización. Las redes de este tipo son muy útiles para algoritmos de reconocimiento de imágenes o sistemas de detección de objetos. Su estructura está formada por varias capas convolucionales y de reducción alternadas entre sí y una capa final de conexión *(ver figura* 2.2).

En la capa convolucional se realizan los productos y sumas entre la capa de partida y los filtros dando lugar a un mapa de características. Cada una corresponde a cada posible ubicación del filtro en la imagen original (*ver en la figura* 2.3). Esto permite que cada neurona pueda extraer dicha característica en cualquier lugar de la imagen, reduciendo el número de parámetros a entrenar y las conexiones necesarias. Una vez aplicada la convolución, se pasa a la función de activación,

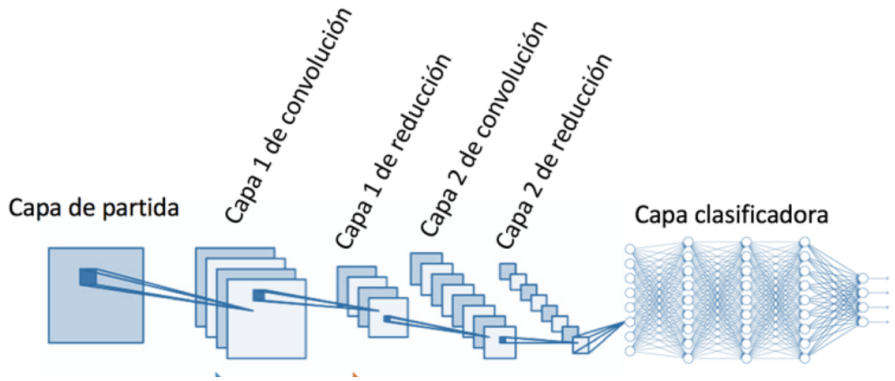


Figura 2.2: *Red neuronal Convolucional [6].*

que define la salida dado el mapa de características. En la capa de reducción, se descartan los parámetros menos comunes utilizando un promedio para cada región (*ver en la figura* 2.4). Finalmente se accede la última capa, la capa clasificadora. Una de las redes seleccionadas para el estudio en este proyecto es YOLO [35].

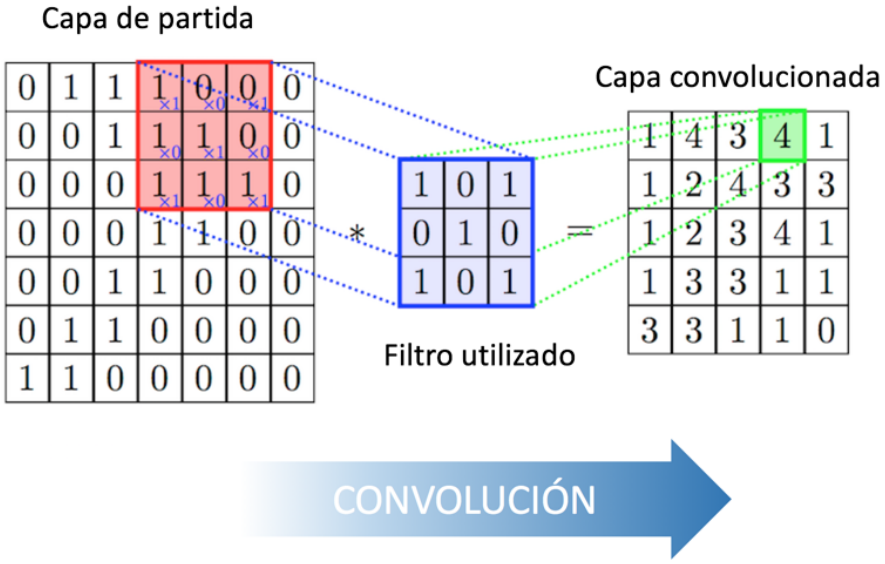


Figura 2.3: *Capa de convolución [6].*

2.1.1. YOLO

YOLO es un sistema de detección de objetos en tiempo real de última generación. Actualmente tiene tres versiones. Es uno de los sistemás más rápidos y precisos que permite cambiar la velocidad y precisión sin necesidad de entrenar de

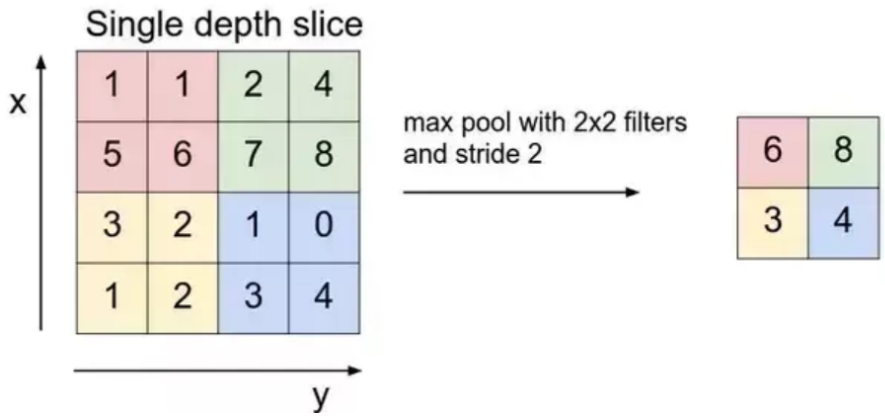


Figura 2.4: *Capa de reducción [38].*

nuevo la red (*ver en la figura* 2.5).

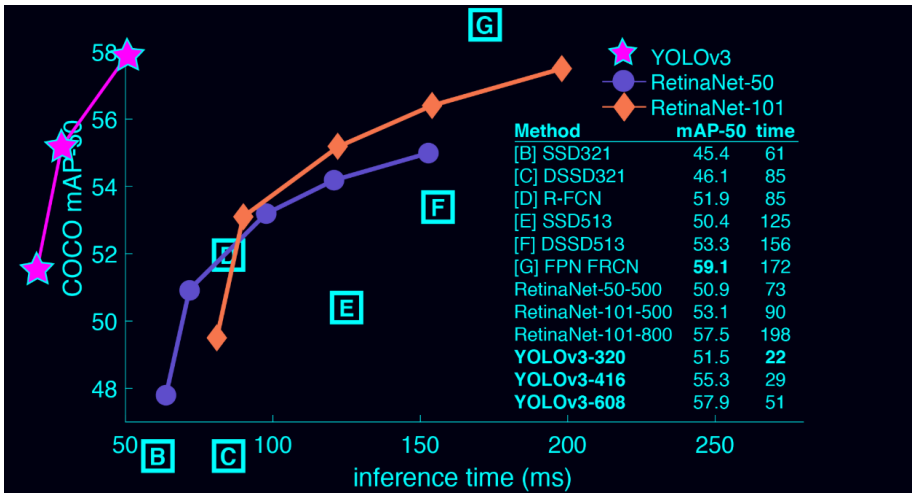


Figura 2.5: *Comparativa de la red YOLO [35].*

Es un modelo que tiene ventaja sobre los sistemas basados en clasificadores ya que examina la red completa en el momento de la prueba y basa sus predicciones en el contexto global de la imagen. Además, estas predicciones las realiza basándose en una sola evaluación, diferenciándola del resto de sistemas como R-CNN.

Según los desarrolladores de YOLO, esta red es 1000 veces más rápida que R-CNN y 100 más que Faster-RCNN. Para entornos restringidos ofrecen un modelo más pequeño, conocido como YOLO-V3-Tiny. Este tipo de redes se pueden acelerar obteniendo grandes resultados. Para ello, se paralelizará en una GPU implementando CUDA-C o CUDA-Python.

2.2. CUDA

Como se puede encontrar en la página de NVIDIA, Compute Unified Device Architecture (CUDA) es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema [21].

La paralelización de un procesador gráfico consiste en segmentar, lo máximo posible, el flujo de datos para poder enviar cada vóxel a cada unidad de procesamiento de una GPU. Un vóxel es unidad más pequeña en la que se puede descomponer un objeto tridimensional. Equivaldría a un pixel en un sistema de dos dimensiones.

Actualmente la mayoría de los campos utilizan un procesador gráfico, para poder conocer la aplicación en los distintos campos de dicha paralelización, NVIDIA lo muestra en el siguiente enlace [22].

A lo largo de la historia de NVIDIA se han presentado siete microarquitecturas. En 2006, publicaron la primera mircroarquitectura conocida como "Tesla". Es una arquitectura con 128 procesadores de multi-threads por GPU y con un ancho de banda de 76.8GB/s.

Fermi, la segunda generación, se publicó en 2010, formada por más de tres millones de transistores y con 512 cores CUDA. Estaba diseñada para la supercomputación como podemos observar en su rendimiento y su consumo era veinte veces menor que los servidores que realizaban dicha tarea con la CPU. Fue la primera arquitectura de computación de la GPU en todo el mundo*(ver figura* 2.6).

La siguiente generación, Kepler, se publicó en 2012. Esta arquitectura presentaba 1536 cores, frente a los 512 que tenía Fermi, además de un menor consumo. Permitía realizar overclock con GPU Boost. El gran avance de esta generación se produce en el tamaño de los transistores, que bajaba de 40nm a 28nm *(ver figura* 2.7).

En 2014, NVIDIA presentaba su cuarta generación, Maxwell. Aunque esta generación tenía un gran rendimiento, la característica que más destacaba era su bajo consumo. Además, incluía la superresolución dinámica que permitía renderizar imágenes 4K en pantallas de 1080p *(ver figura* 2.8).

La quinta generación, Pascal, sorprendía a los consumidores debido a su rendimiento. Pascal ofrecía 10 veces más rendimiento que la generación anterior con transistores de 16nm. Además incluía el uso de memoria 3D que provocaba que el ancho de banda fuera todavía mayor *(ver figura* 2.9).

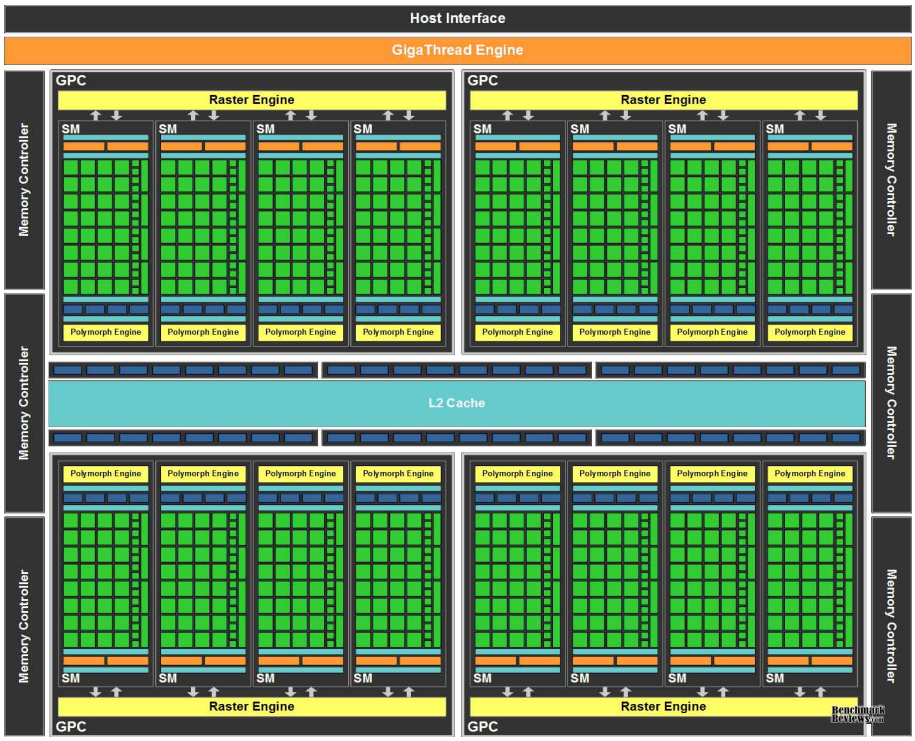


Figura 2.6: *Arquitectura Fermi [7].*

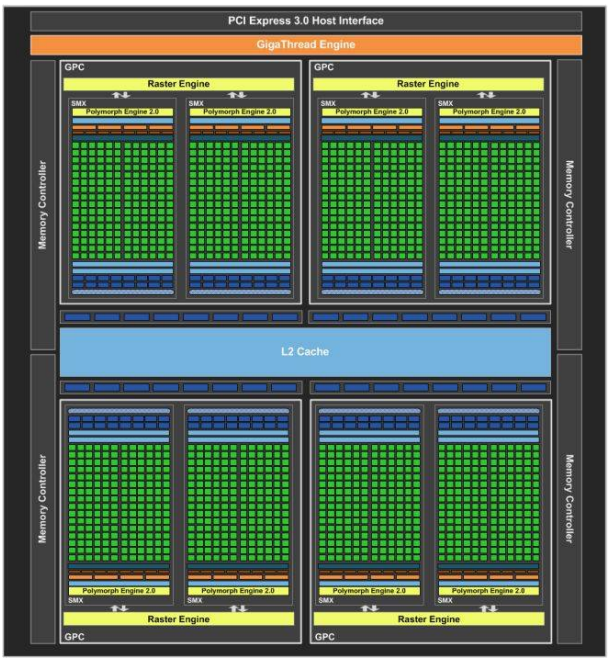


Figura 2.7: *Arquitectura Kepler [8].*

Figura 2.8: *Arquitectura Maxwell [9].*

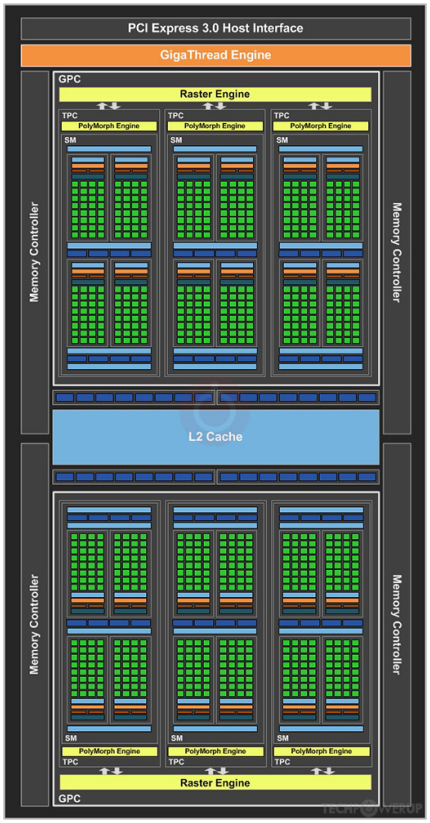


Figura 2.9: *Arquitectura Pascal [13].*

La siguiente generación de tarjetas gráficas de NVIDIA no llegaría hasta 2018. Dicha generación fue conocida como 'Volta'. Volta conseguía bajar el tamaño de sus transistores a 12 nm, lo que aumentaba considerablemente el rendimiento (110 TFLOPS). Además de 5120 cores, entre los cuales 640 Tensor para operaciones de Deep Learning. Esta última generación sufrió una modificación en la arquitectura y se conoce como Turing. La modificación que presentaba dicha generación no estaba orientada solo al rendimiento en el sector del gaming, sino que apostaba por el ray-tracing [29] y a procesadores neuronales *(ver figura* 2.10).

Figura 2.10: *Arquitectura Volta [4].*

Todas estas generaciones han sido programadas en CUDA, que ha ido evolucionando hasta llegar a la versión en la que nos encontramos a día de hoy, CUDA X. Ha sido presentada este mismo año en el GTC 2019, pero a día de hoy no se conoce todavía suficiente información.

	K40(Kepler)	**M40(Maxwell)**	**P100(Pascal)**	**V100(Volta)**
GPU	GK110	GM200	GP100	GV100
Transistores(x10^6)	7100	8000	15300	21100
Área de integración	$551nm^2$	$601nm^2$	$610mm^2$	$815mm^2$
Fabricación	28 nm	28 nm	16 nm. FinFET	12 nm.FinFET
TDP	235 W	250 W	300 W	300 W
Numero de cores fp32	2880 (15 x 192)	3072 (24 x 128)	3584 (56 x 64)	5120 (80 x 64)
Numero de cores fp64	960	96	1792	2560
Frecuencia nominal	745 MHz	948 MHz	1328 MHz	1370 MHz
Frecuencia Boost	875 MHz	1114 MHz	1480 MHz	1455 MHz
TFLOPS(fp16,fp32,fp64)	No, 5.04, 1.68	No, 6.8, 2.1	21.2, 10.6, 5.3	30, 15, 7.5
Interfaz de memoria	GDDR5 de 384 bits:		HBM2 de 4096 bits	
Memoria de vídeo	Hasta 12 GB	Hasta 24 GB	16 GB	16 GB
Caché L2	1536 KB	3072 KB	4096 KB	6144 KB
Memoria compartida/SM	48 KB	96 KB	64 KB	Hasta 96 KB
Banco de registros/SM	65536	65536	65536	65536

Tabla 2.1: *Resumen de las arquitecturas desde Kepler hasta Volta.*

Este mismo año también, NVIDIA presentó una nueva plataforma de aprendizaje profundo de alto rendimiento, TensorRT. Este sistema optimiza inferencias de aprendizaje profundo bajando la latencia y aumentando el rendimiento. Según datos obtenidos por investigadores de NVIDIA se puede obtener un factor de aceleración de hasta 40x. Basado en CUDA X-AI, proporciona optimizadores INT8 y FP16 para aprendizaje profundo como sistemas de procesamiento de video. Sin embargo, no ha podido ser utilizado en nuestro sistema ya que las arquitecturas hardware que se han utilizado no tenían habilitados dichos cores [31].

3

Hardware y Software

Dado que este proyecto consta de un sistema con un cálculo computacional elevado, se han analizado diferentes procesadores gráficos para obtener datos sobre su rendimiento y su eficiencia. Los dispositivos que han sido seleccionados para dicho análisis son: un equipo del laboratorio, NVIDIA Jetson TX2 y NVIDIA Jetson Nano.

3.1. Hardware

3.1.1. Equipo de laboratorio

Este equipo ha sido el primero con el que se ha realizado las pruebas ya que estaba equipado con una NVIDIA GTX 960, que iba a ser utilizada posteriormente en NVIDIA Jetson Nano. Esta tarjeta gráfica al ser de la misma arquitectura podía

proporcionar datos relativamente cercanos a los que NVIDIA Jetson Nano ofrecería al implementar el software. NVIDIA GTX 960 esta orientada al *gaming* de última generación y se caracteriza por su rendimiento y eficiencia con un bajo consumo. Esta GPU tiene 1024 núcleos CUDA, un ancho de banda de 112Gb/s, un consumo de 120W y la arquitectura Maxwell *(ver figura* 3.1).

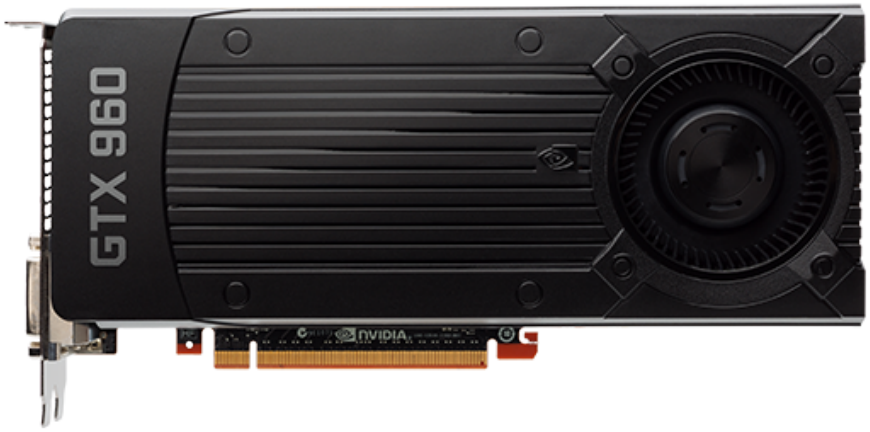


Figura 3.1: *NVIDIA GTX 960 [30].*

El equipo incluye una CPU Intel® Core™2 Quad Q8300. Procesador de 4 núcleos, con una frecuencia de reloj de 2.5 GHz, caché de 4 MB y un consumo de 95 W. Esta orientado al procesamiento de multitareas aunque no para aquellas muy exigentes debido a que es una versión algo anticuada.

3.1.2. NVIDIA Jetson TX2

NVIDIA Jetson TX2, destaca por ser un dispositivo con un consumo de energía muy bajo que ofrece un gran rendimiento computacional *(ver figura* 3.2 y 3.4). Como podemos encontrar en la página de NVIDIA: "Jetson TX2 incorpora una GPU NVIDIA Pascal de 256 núcleos, un complejo de CPU de 64 bits ARMv8 de núcleo hexadecimal y 8 GB de memoria LPDDR4 con una interfaz de 128 bits"[21]. Tiene un peso de tan solo 85 gramos y un consumo 7.5 W con una exigencia normal.

NVIDIA Jetson TX2 tiene transistores de 16 nm y dos veces más rendimiento que su antecesora NVIDIA Jetson TX1. Este dispositivo es perfecto para poder desplegar IA e implementar operaciones de aprendizaje profundo.

NVIDIA Jetson TX2 tiene múltiples motores de *streaming* multimedia que incluyen seis puertos de cámara MIPI CSI-2 *(ver figura* 3.3). Estos puertos proporcionan hasta 2.5Gb/s de ancho de banda y 1.4 gigapíxeles/s de procesamiento me-

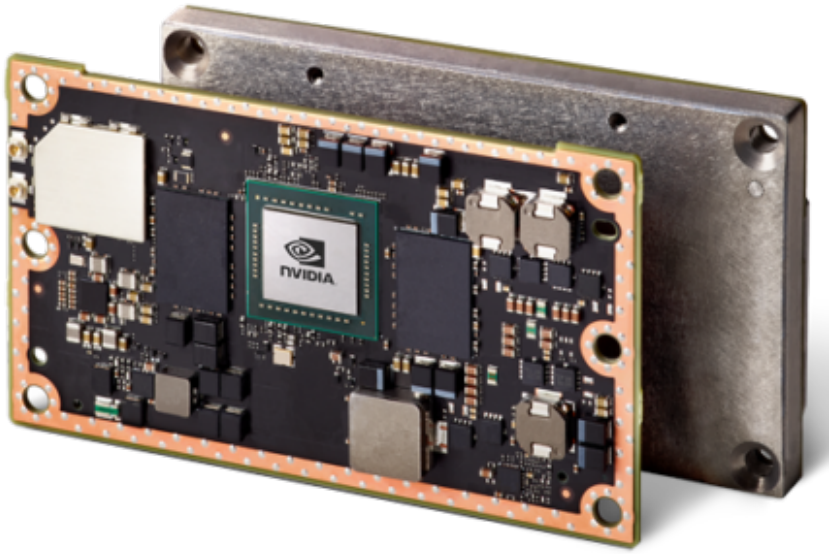


Figura 3.2: *NVIDIA Jetson TX2 [27].*

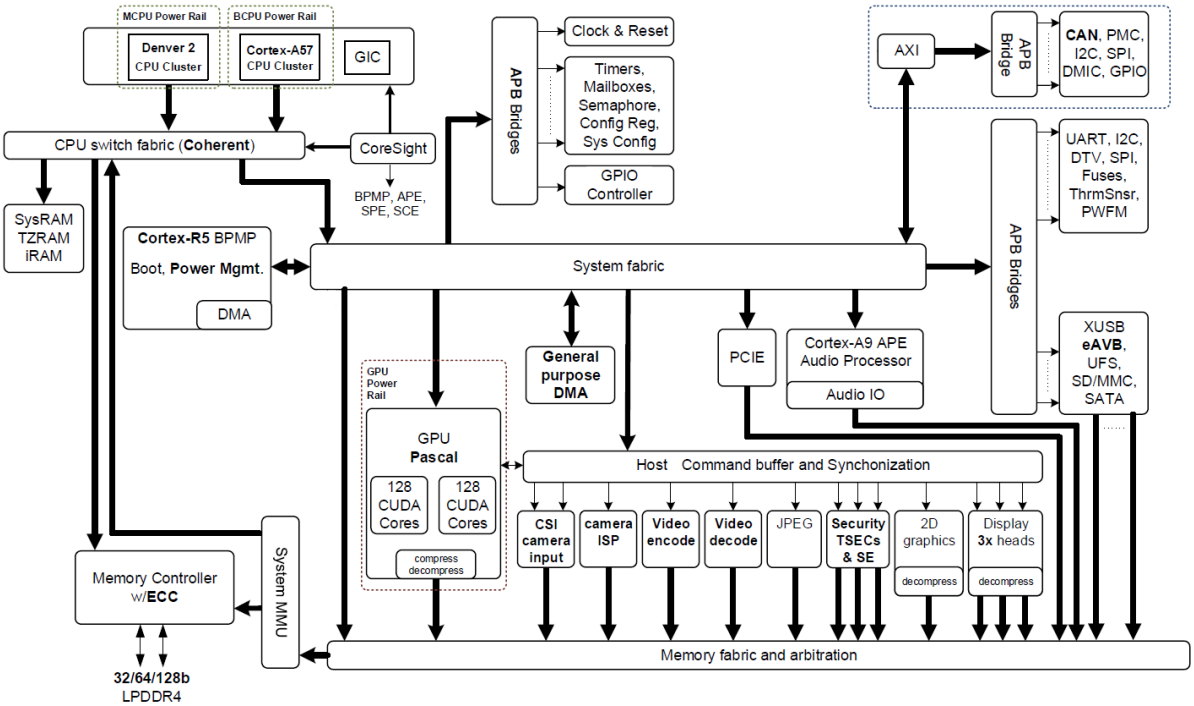


Figura 3.3: *Diagrama de bloques NVIDIA Jetson TX2 [27].*

diante procesadores duales de servicios de imagen (ISP), así como códecs de vídeo compatibles con H.265 a 4K 60 fotogramas por segundo [26].

Se ha decidido utilizar este dispositivo ya que NVIDIA Jetson TX2 esta diseñada para acelerar las arquitecturas de red neuronal profunda (DNN) más avanzadas, utilizando las bibliotecas cuDNN de NVIDIA, con soporte para redes neuronales recurrentes (RNN), redes de memoria a corto plazo (LSTM) y aprendizaje de refuerzo en línea.

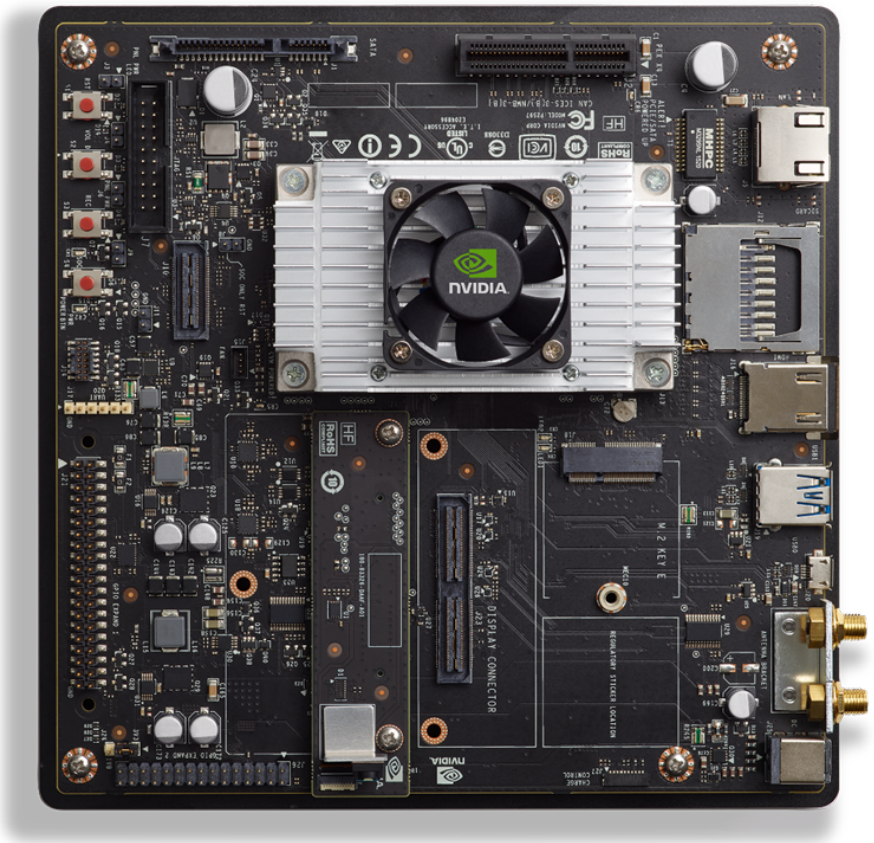


Figura 3.4: *NVIDIA Jetson TX2 50 x 87 [27].*

3.1.3. NVIDIA Jetson Nano

Gracias a Manuel Ujaldón Martínez, tutor de este proyecto, se ha podido tener acceso unos meses antes de su lanzamiento a esta nueva arquitectura *(ver figura* 3.5). NVIDIA Jetson NANO, como se puede ver en la web oficial, permite el desarrollo de nuevos sistemas de IA con un bajo consumo de energía. Si antes se hablaba sobre el bajo consumo que presentaba NVIDIA Jetson TX2, esta nueva arquitectura disminuye el consumo a 5 W. Es actualmente el dispositivo Jetson de menor tamaño. NVIDIA Jetson Nano proporciona 472 GFLOPS para ejecutar los algoritmos de la IA moderna de forma rápida [28].

Incorpora Arquitectura NVIDIA Maxwell™ con 128 núcleos CUDA como GPU, procesador ARM Cortex-A57 MPCore de cuatro núcleos, memoria LPDDR4 de 4 GB y 64 bits, 16 GB de almacenamiento Flash eMMC 5.1 y codificación y decodificación de vídeo a 4K/30 FPS y 4K/60 FPS, respectivamente.

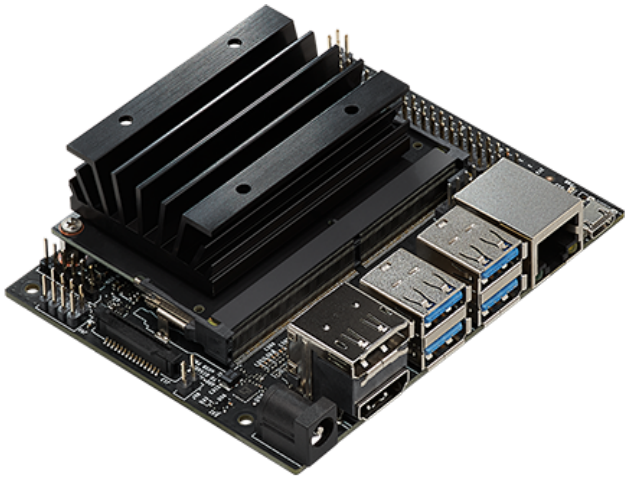


Figura 3.5: *NVIDIA Jetson Nano 70 x 45 mm [24].*

	GTX 960	JETSON NANO	JETSON TX2
Arquitectura	Maxwell	Maxwell	Pascal
Millones de transistores	2940	2940	7200
Fabricación	28 nm	28 nm	16 nm
CUDA Cores	1024	128	256
Frecuencia Nominal y Boost	1127/1178 MHz	921 MHz	854 MHz
Memoria de vídeo	2 GB	4 GB	4 GB
Caché L2	1024 KB	256 KB	2048 KB
TDP	120	5 W	7.5 W

Tabla 3.1: *Resumen de características de las GPUs utilizadas.*

3.2. Software

Para el funcionamiento correcto de estos sistemas se ha instalado el gestor de dependencias Anaconda [3] que nos permitía el uso de versiones distintas de una misma librerías para poder compatibilizar el funcionamiento de varias CNN.

Como sistema operativo se ha utilizado Linux Ubuntu 18.04LTS [42], principalmente porque toda la documentación necesaria viene referida a Ubuntu, también, tanto NVIDIA Jetson Nano como NVIDIA Jetson TX2 incluye dicho sistema. Las librerías más importantes:

- Cuda 10.0 [23].
- CuDNN 7.3 [25].
- OpenCV 3.3.1 [32].
- GPU con CC >= 3.0
- CMake >= 3.8

4

Evaluación experimental

En un principio, se tomó como referencia una red neuronal que destacase por su rendimiento, ya que este trabajo sigue la línea del proyecto DIAS2P-StreetQR. Se buscó una que permitiese un funcionamiento correcto del sistema de procesamiento de vídeo. La primera que se seleccionó fue Mask R-CNN. Posteriormente, viendo que era necesario una red neuronal que ofreciese más FPS para que la respuesta fuese en tiempo real, se utilizó YOLO.

Para medir el rendimiento de las redes se tomaron como parámetros:

- FPS (Frames Per Second): Número de veces que se actualiza la imagen por segundo
- Start-up: Tiempo de arranque o inicialización
- Delay: Retraso o latencia

4.1. Mask R-CNN

Mask R-CNN es un sistema de detección de objetos. Este sistema destaca por su gran precisión. Es una red neuronal fácil de entrenar, aunque su principal problema es sus bajos FPS, lo cual dificulta o imposibilita el uso de esta red. Sabiendo que este trabajo sigue la línea del proyecto DIAS2P-StreetQR, en el cual se trabaja con un sistema que requiere una respuesta inmediata, esta CNN no sería viable.

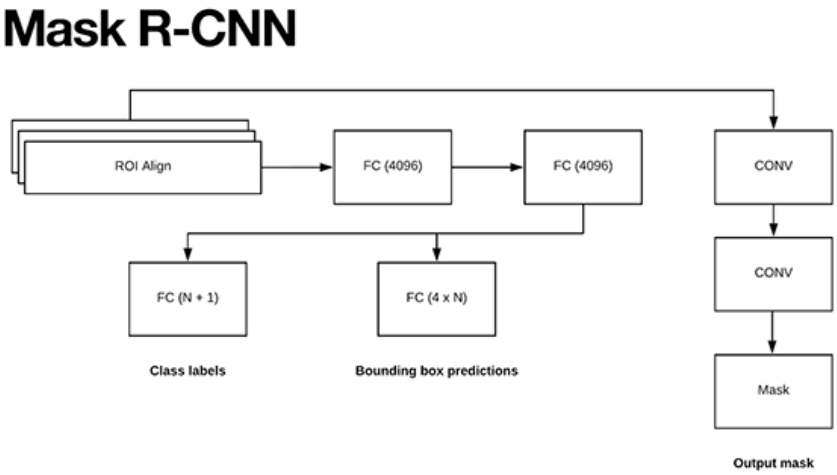
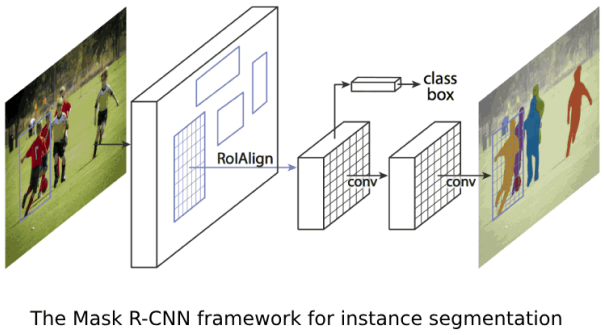


Figura 4.1: *Diagrama de capas Mask R-CNN [34].*



The Mask R-CNN framework for instance segmentation

Figura 4.2: *Procesamiento de una imagen Mask R-CNN [40].*

4.1.1. Equipo de laboratorio

Mask-RCNN funcionó con una media de 2 FPS, un *delay* que oscilaba entre 3 y 5 segundos y un tiempo de start-up que era algo superior a un segundo. Todos

estos datos se obtuvieron de su rendimiento en GPU, ya que los desarrolladores no daban opción a una ejecución secuencial en CPU. Estos resultados descartaban prácticamente el uso de esta red a esperas de que existiese una gran mejora en NVIDIA Jetson TX2 o NVIDIA Jetson Nano.

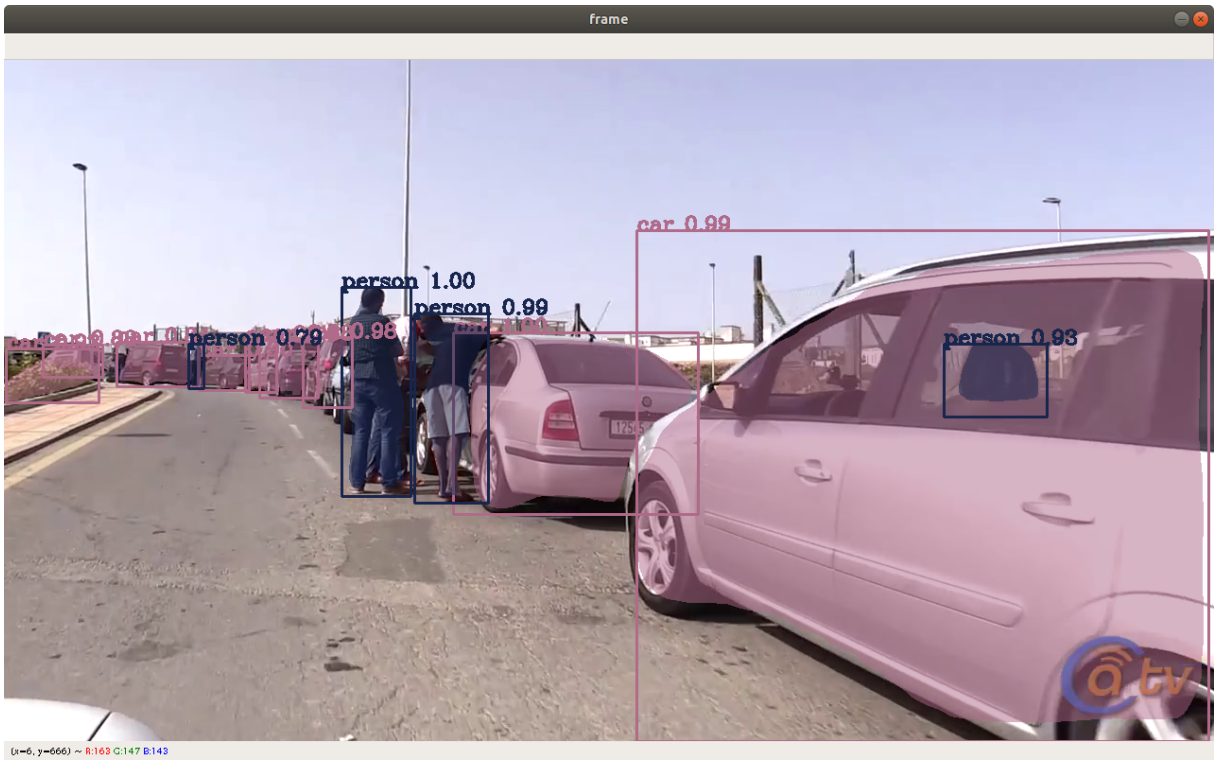


Figura 4.3: *Mask-RCNN funcionando en NVIDIA GTX 960 .*

Esta misma red se intentó instalar en NVIDIA Jetson Nano y NVIDIA Jetson TX2, sin embargo, no se obtuvo ningún resultado debido a las dependencias que existía entre las librerías. En estos dispositivos no se consiguió instalar correctamente los gestores de dependencias y constantemente daba errores de incompatibilidad de librerías. Dado que el proyecto DIAS2P-StreetQR no disponía del tiempo necesario para solucionar dicha dependencia, se decidió descartar la instalación. Además, los resultados ofrecidos por NVIDIA GTX 960, los cuales serían prácticamente idénticos en NVIDIA Jetson Nano, no mostraban ninguna luz para continuar con este sistema.

Tanto NVIDIA Jetson Nano como NVIDIA Jetson TX2, son dispositivos que traen preinstalado CUDA 10.0 y cuDNN 7.3, lo que genera grandes conflictos de dependencias para las redes neuronales menos actuales.

4.2. Yolo

Tras Mask R-CNN, se buscó una red neuronal que pudiese ofrecer más imágenes por segundo para que una vez que se determinase el tratamiento de estos datos permitiese una respuesta en tiempo real. Esto nos llevó a seleccionar YOLO. Esta CNN es un sistema de detección de objetos en tiempo real. Se caracteriza por ser el más rápido y muy preciso. Mientras que la mayoría de estos sistemas reutilizan localizadores o clasificadores para realizar la detección, YOLO aplica una sola red neuronal a la imagen, dividiendo ésta en regiones y dándole una probabilidad a cada región *(ver figura* 4.4).

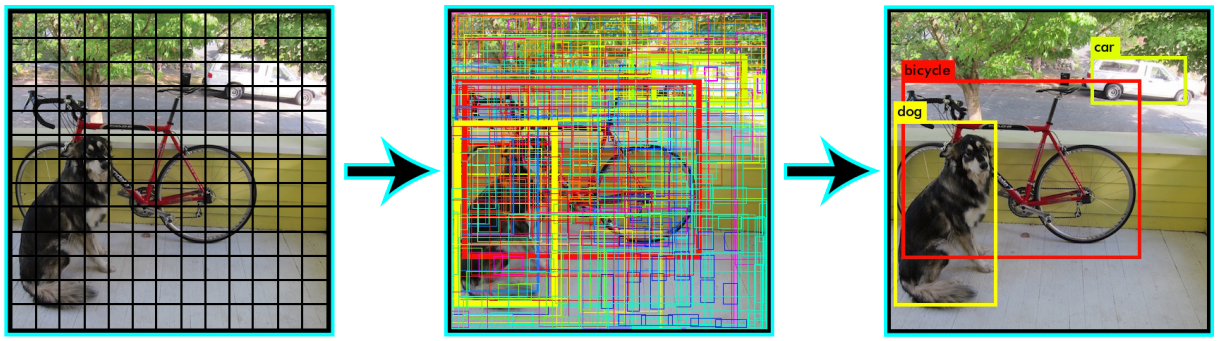


Figura 4.4: *Red neuronal YOLO [37].*

4.2.1. Yolo-V2

Cuando se comenzó a trabajar con esta red neuronal, la versión existente fue YOLO v2, ésta destacaba por ser el sistema de detección de objetos en tiempo real con mayor rendimiento hasta la salida de la nueva versión. Concretamente la versión utilizada ha sido *YOLO v2 tiny*, ya que dentro de la versión 2, ésta ofrecía mucho mayor rapidez aunque perdiese algo de precisión. En este proyecto, ya que no se disponía del tiempo para entrenar la CNN, se utilizó una red ya entrenada. YOLO v2 permitía la posibilidad de procesar los datos, obtenidos en tiempo real mediante una cámara web que proporciona imágenes con una resolución de 640 x 480 píxeles [37].

4.2.1.1. Equipo de laboratorio

Para obtener unos datos fiables respecto al *start-up*, se ejecutó el código 20 veces y se tomó como resultado la media, obteniendo 2336.21 y 435.6 milisegundos

si se ejecutaba en la GPU y CPU, respectivamente. Respecto a FPS, la ejecución de la red neuronal en CPU tenía una media de 0.5 FPS, mientras que si este mismo se ejecutaba en GPU el resultado era de 15 FPS, con picos de 20 FPS *(ver figura* 4.5).

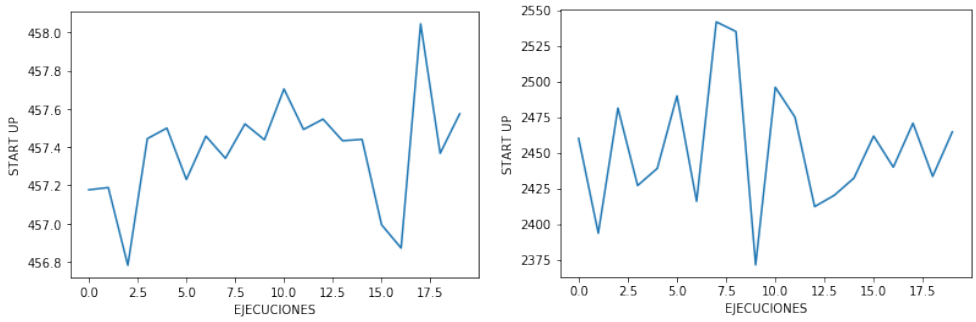


Figura 4.5: *Variación en el tiempo de inicialización de la red Yolo-v2 a lo largo de 20 ejecuciones, a la izquierda en la CPU y a la derecha en la GPU GTX 960.*

Las gráficas se han realizado mediante el entorno jupyter notebook [19], importando las librerías de matplotlib [18]. En el eje de abscisa se muestra el número de ejecuciones y en el eje de ordenada el tiempo de start-up. Con estas gráficas se quiere reflejar la varianza que ha ido tomando el sistema durante las distintas ejecuciones.

4.2.1.2. NVIDIA Jetson TX2

Con respecto a los otros dispositivos, Jetson TX2 iba a mostrar resultados considerablemente mejores en rendimiento. Nuestras expectativas se cimentaban en que presentaba una tarjeta gráfica de la siguiente generación, Pascal, aunque también un mayor consumo y tamaño que la Jetson Nano. Este dispositivo incorpora Dual-Core NVIDIA Denver 2 64-Bit y Quad-Core ARM® Cortex®-A57 MPCore como CPU y 256-core NVIDIA Pascal™ como GPU. Los tiempos de *start-up* fueron de 2668.63 ms en GPU y de 927.012 ms en CPU. Sin embargo, la gran diferencia se puede observar en el apartado de FPS donde ejecutando este código en GPU se obtenía una media de 30 FPS (0.5 FPS si se ejecutaba en CPU)*(ver figura* 4.6).

4.2.1.3. NVIDIA Jetson Nano

Las grandes sorpresas iban a encontrarse en este nuevo hardware. Pese a las dimensiones que presenta este dipositivo [24], está formada por una NVIDIA Maxwell™ -128 cores (GPU) y ARM® Cortex®-A57 MPCore de cuatro núcleos (CPU),

YOLO:

Capa	Filtros	Tamaño	Entrada	V2(BFLOPS	V3 (BFLOPS)
0 Conv	16	3 x 3 /1	416 x 416 x 3	0.150	0.150
1 Max		2 x 2 /2	416 x 416 x 16	0.003	0.003
2 Conv	32	3 x 3 /1	208 x 208 x 16	0.399	0.399
3 Max		2 x 2 /2	208 x 208 x 32	0.001	0.001
4 Conv	64	3 x 3 /1	104 x 104 x 32	0.399	0.399
5 Max		2 x 2 /2	104 x 104 x 64	0.001	0.001
6 Conv	128	3 x 3 /1	52 x 52 x 64	0.399	0.399
7 Max		2 x 2 /2	52 x 52 x 128	0.000	0.000
8 Conv	256	3 x 3 /1	26 x 26 x 128	0.399	0.399
9 Max		2 x 2 /2	26 x 26 x 256	0.000	0.000
10 Conv	512	3 x 3 /1	13 x 13 x 256	0.399	0.399
11 Max		2 x 2 /2	12 x 12 x 512	0.000	0.000
12 Conv	1024	3 x 3 /1	13 x 13 x 512	1.595	1.595
13 Conv	512/256	1 x 1 /1	13 x 13 x 1024	1.595	0.089
14 Conv	425/512	3 x 3 /1	13 x 13 x 256	0.074	0.399
15 Conv	255	1 x 1 /1	13 x 13 x 512	-	0.044
16 Yolo					0.011
17 Route	13				
18 Conv	128	1 x 1 /1	13 x 13 x 256	-	0.011
19 Upsample					
20 Route	19				
21 Conv	256	3 x 3 /1	26 x 26 x 384	-	1.196
22 Conv	255	1 x 1 /1	26 x 26 x 256	-	0.088
23 Yolo					
TOTAL				5.412	5.571

Tabla 4.1: *Desglose de la arquitectura YOLO y rendimiento obtenido en BFLOPS (Billones de operaciones de punto flotante por segundo) 5.412 y 5.571 para las versiones v2 y v3 sobre Jetson Nano. En las capas max aplica la función máximo a cada ventana de valores para reducir la matriz, por este motivo suele ser 0 las operaciones de punto flotante.*

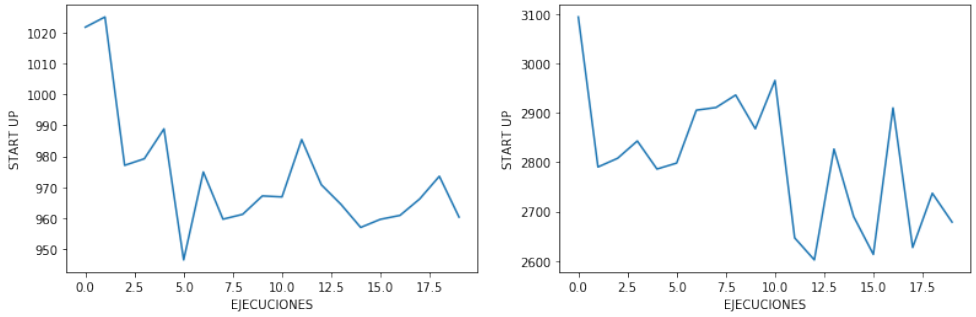


Figura 4.6: *Variación en el tiempo de inicialización de la red Yolo-v2 a lo largo de 20 ejecuciones, a la izquierda en la CPU y a la derecha en la GPU Jetson TX2.*

por lo que, a priori, se esperaban resultados algo peores a los obtenidos en la NVIDIA GTX 960 (con la que comparte arquitectura). Sin embargo, los resultados fueron prácticamente idénticos, no existía esa pérdida esperada. En *start-up* tenía un tiempo de 3855.06 y 1479.45 milisegundos en GPU y CPU, respectivamente *ver figura* 4.7).

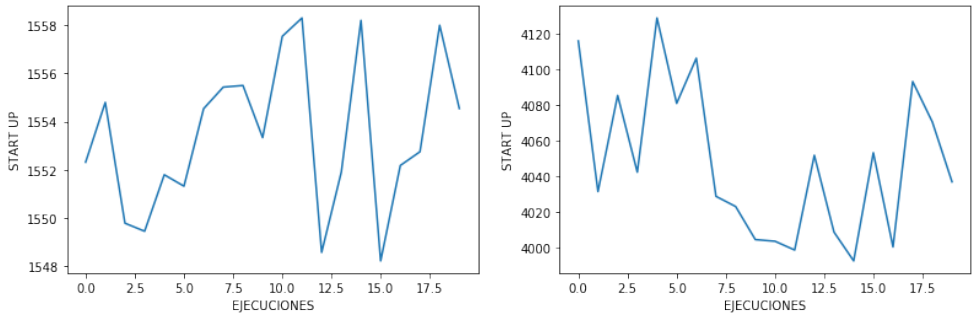


Figura 4.7: *Variación en el tiempo de inicialización de la red Yolo-v2 a lo largo de 20 ejecuciones, a la izquierda en la CPU y a la derecha en la GPU Jetson Nano.*

4.2.2. Yolo-V3

Durante la implantación de YOLO v2, los investigadores desarrollaron una nueva versión que mejora el entrenamiento y que aumenta el rendimiento, incluyendo predicciones a múltiples escalas. Al igual que con YOLO v2, la versión utilizada ha sido *YOLO v3 tiny* [35]. En un principio, se pensó que esta mejora de rendimiento podía suponer también una necesidad de prestaciones hardware superiores, lo cual no interesaba a este proyecto. No obstante, los resultados mostraron que no, que esta mejora de rendimiento se notaba considerablemente en términos

de *delay* y de *start-up* (no tanto en el apartado de FPS) *(ver tabla* 4.1).

4.2.2.1. Equipo de laboratorio

Al igual que en los casos anteriores, se tomaron 20 ejecuciones del programa para obtener el tiempo de *start-up*. En este caso, el tiempo de inicio fue de 356.37 ms. En CPU, mientras que en GPU era de 2127.28 ms. Con estos resultados ya se puede observar una leve diferencia respecto a YOLO v2 en CPU, y una diferencia más notable ejecutándose en GPU. En el apartado de los FPS no se obtuvieron mejoras considerables pero si más precisión. Se podría hablar de un FPS más de media *(ver figura* 4.10).

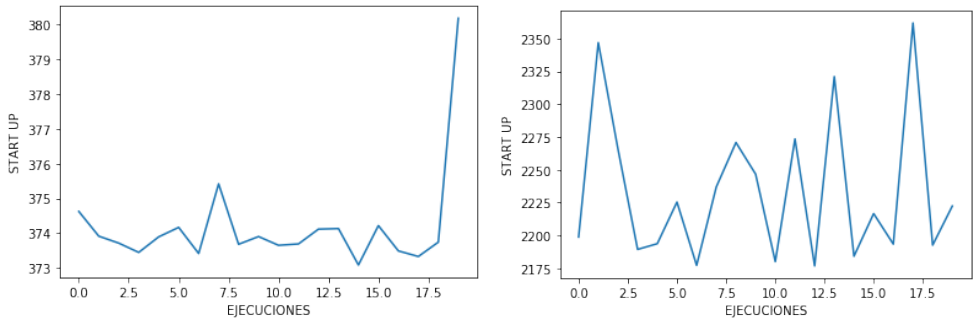


Figura 4.8: *Variación en el tiempo de inicialización de la red Yolo-v3 a lo largo de 20 ejecuciones, a la izquierda en la CPU y a la derecha en la GPU GTX 960.*

4.2.2.2. NVIDIA Jetson TX2

NVIDIA Jetson TX2, también iba a presentar esta mejora, aunque si la comparamos con los otros dispositivos, ésta no iba a ser de tal magnitud. Los tiempos de *start-up* fueron de 2658.37 y 798.49 ms en GPU y CPU, respectivamente. En CPU conseguía una aceleración de 128.52 ms y en GPU de 10ms, esta última prácticamente nula. En términos de FPS se obtuvieron 0.5 en CPU y 25-35 *frames* en GPU, aunque de nuevo el delay era algo menor *(ver figura* 4.9).

4.2.2.3. NVIDIA Jetson Nano

Tras las ejecuciones mencionadas, se obtuvo que el tiempo de *start-up* disminuía respecto a YOLO V2 de 3855 a 3519.16 ms en GPU, y de 1479.45 a 1193.89 en CPU. En FPS, de nuevo se volvía a ver esa diferencia mínima de 1 *frame*. Destacar

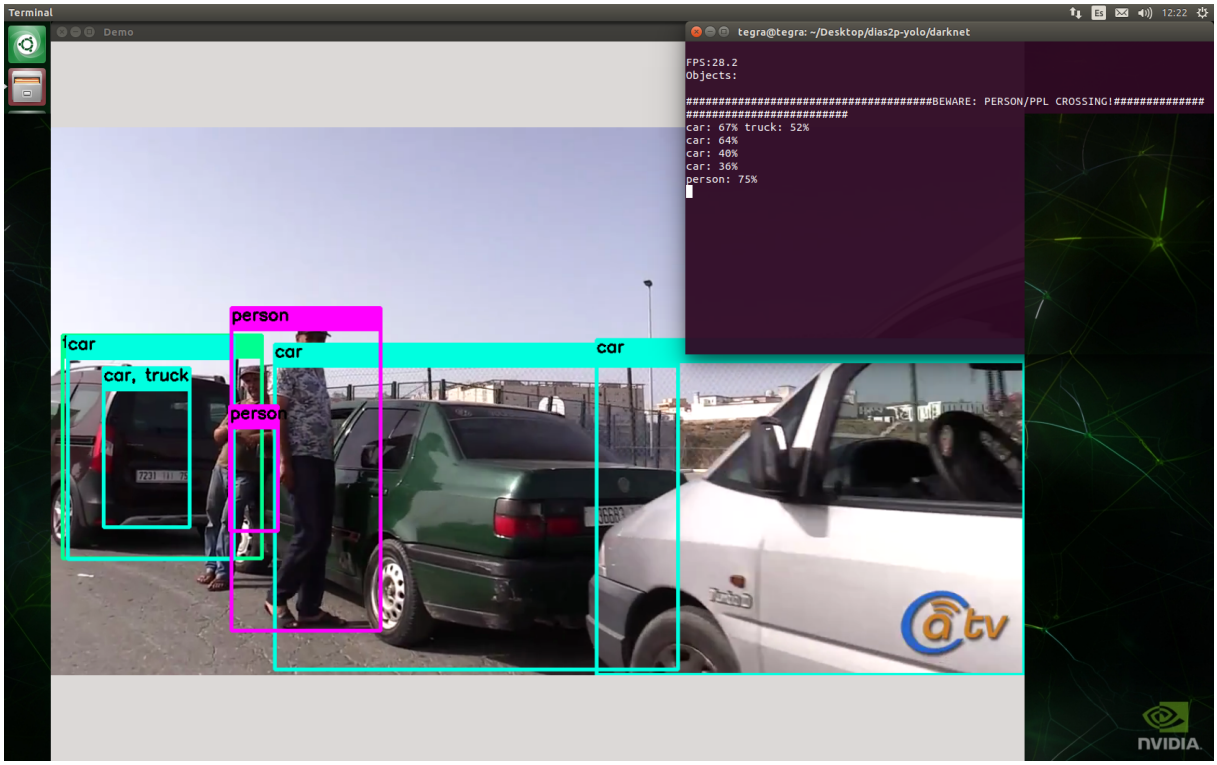


Figura 4.9: *YOLO-V3-Tiny funcionando en NVIDIA Jetson TX2.*

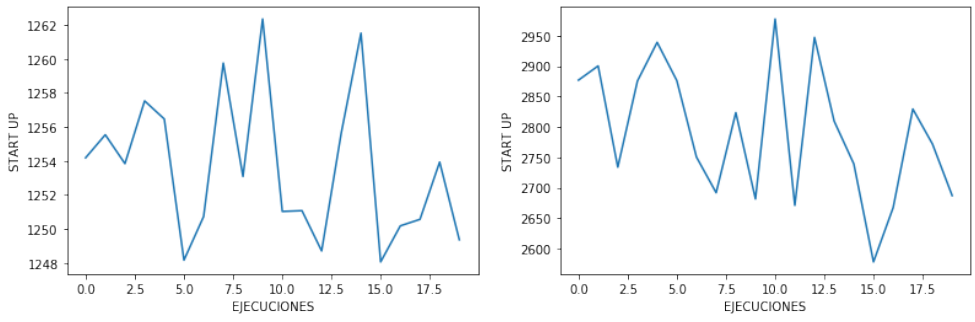


Figura 4.10: *Variación en el tiempo de inicialización de la red Yolo-v3 a lo largo de 20 ejecuciones, a la izquierda en la CPU y a la derecha en la GPU Jetson TX2.*

también que el *delay* desaparecía prácticamente. Con estos resultados obtenidos se encontraba una combinación de red neuronal, optimizada en GPU, y hardware con bajo consumo que permitía una respuesta en tiempo real *(ver figura* 4.11).

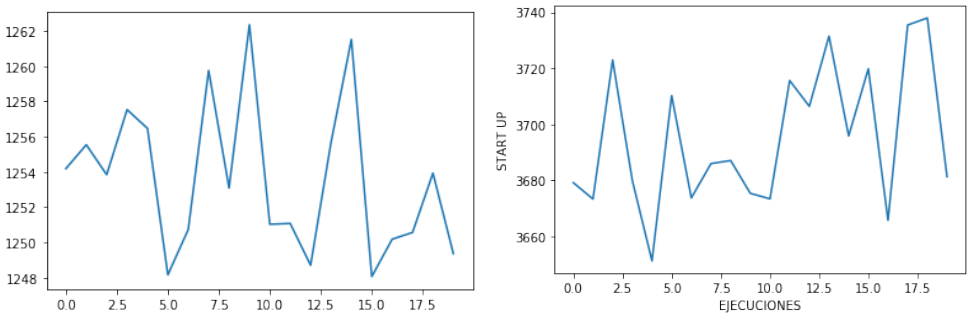


Figura 4.11: *Variación en el tiempo de inicialización de la red Yolo-v3 a lo largo de 20 ejecuciones, a la izquierda en la CPU y a la derecha en la GPU Jetson Nano..*

5

Discusión y evaluación de resultados

Para el análisis de los resultados obtenidos se han clasificado los diferentes sistemas según dos parámetros. El primero se basará en el rendimiento, destacando aquellos que han ofrecido más FPS con un menor *start-up* y *delay*. Para el segundo, se tomará como referencia el tamaño y el consumo de los dispositivos.

Estudiando los datos obtenidos (ver en la tabla 5.1), YOLO ofrece mejores resultados que Mask-RCNN. Una de las causas en las que se basa esta diferencia es que Mask-RCNN utiliza principalmente librerías de python, un lenguaje de alto nivel, mientras que YOLO está implementado en su mayor parte en C.

Además, Mask-RCNN es una red neuronal más precisa que YOLO, mientras que YOLO sacrifica precisión con objetivo de proporcionar más FPS. Dentro de las

distintas versiones de YOLO, la versión 3, es la que ofrece mejores resultados. YOLO v3 tiny, incorpora nuevas optimizaciones respecto a YOLO v2 que permiten que el sistema reduzca a nulo el *delay* y *start-up*.

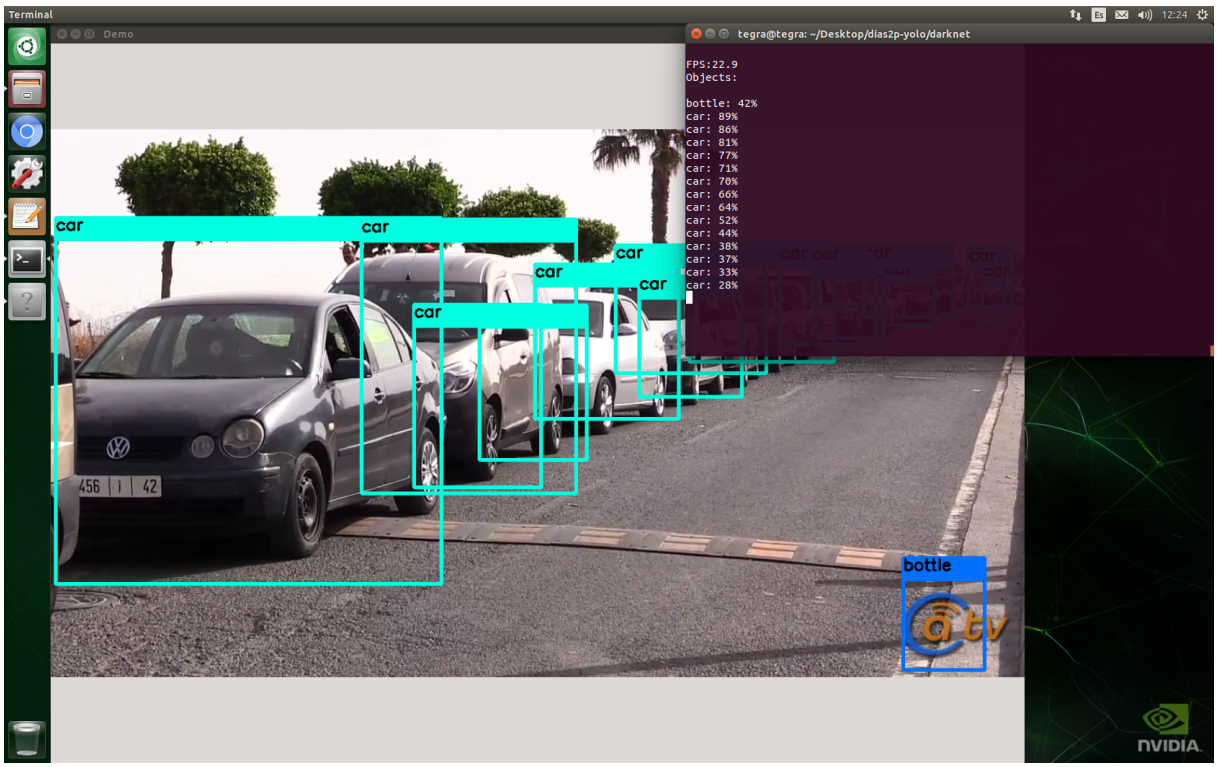


Figura 5.1: *YOLO V2 funcionando en NVIDIA Jetson TX2.*

5.1. Según el rendimiento

Siguiendo estos parámetros se ha obtenido que el hardware que mostraba mejor resultado era NVIDIA Jetson TX2, seguido de NVIDIA Jetson Nano y NVIDIA GTX 960. Esta mejora es debido a la arquitectura de la tarjeta gráfica que presenta NVIDIA Jetson TX2, Pascal, la cual es la siguiente generación de arquitecturas a la que incorpora NVIDIA Jetson Nano y NVIDIA GTX 960, Maxwell.

Según podemos ver en la *tabla* 5.1, Jetson TX2 ofrece 30 FPS, por lo que sería un sistema a tiempo real (superior a los 20 FPS), con un *start-up* de 2658.37 milisegundos y con un delay que se puede despreciar si utilizamos YOLO V3.

Además, como suele ocurrir en la paralelización en CUDA, si comparamos los tiempos de *start-up* de las diferentes redes ejecutándose en CPU y en GPU,

se puede observar que la CPU siempre es más rápida, aunque luego a la hora de procesar el *stream* los resultados no son viables *(ver tablas* 5.1 *y* 5.2).

También se ha podido observar que YOLO v3 en NVIDIA Jetson TX2 consigue mantener los *frames* cuando la red detecta muchos objetos *(ver figura* 5.1), sin embargo, YOLO v2 decae hasta 10 FPS. Esta diferencia no se aprecia si se ejecuta en NVIDIA Jetson Nano o NVIDIA GTX 960.

	GTX 960	**JETSON NANO**	**JETSON TX2**
Arquitectura	Maxwell	Maxwell	Pascal
YOLO V3:			
start-up	2127.28 ms	3519.16 ms	2658.37 ms
FPS	15 frames	15 frames	30 frames
Delay	0.5 seg	0.5 seg	Nulo
YOLO V2:			
start-up	2336.21 ms	3855.06 ms	2668.63 ms
FPS	15 frames	15 frames	30 frames
Delay	1 seg	1 seg	0.5seg
Mask-RCNN:			
start-up	2 ms	no disponible	no disponible
FPS	2 frames	no disponible	no disponible
Delay	4 seg	no disponible	no disponible

Tabla 5.1: *Resumen de los resultados obtenidos para cada uno de los 3 parámetros de rendimiento seleccionados sobre cada una de las plataformas GPU elegidas para la ejecución.*

5.2. Según el consumo y las dimensiones

El nacimiento de las GPUs como aceleradores hardware estuvo basado en el rendimiento, articulado éste a su vez en torno a dos grandes vertientes:

- Altas prestaciones en aplicaciones *large scale* constituyeron la primera ola, donde *Big Data* y *Deep Learning* son claros exponentes de vanguardia que mantienen su vigencia actual.

- Respuesta ágil en aplicaciones de tiempo real formaron la segunda ola, pudiendo mencionar aquí áreas tan extensas como la robótica y los servidores Web.

	GTX 960	JETSON NANO	JETSON TX2
Arquitectura	Maxwell	Maxwell	Pascal
YOLO V3:			
start-up	356.37 ms	1193.89 ms	798.49 ms
FPS	0.5 frames	0.5 frames	0.5 frames
Delay	4 seg	4 seg	3seg
YOLO V2:			
start-up	435.6 ms	1479.45 ms	927.012 ms
FPS	0.5 frames	0.5 frames	0.5 frames
Delay	4 seg	4 seg	3seg

Tabla 5.2: *Resumen de los resultados obtenidos para cada uno de los 3 parámetros de rendimiento seleccionados sobre cada una de las plataformas CPU elegidas para la ejecución.*

Una tercera ola que está complementando a las dos anteriores en tiempos más recientes está compuesta por aplicaciones que requieren consumo y dimensiones reducidos, y en la que podemos citar fenómenos como las *smart cities* y el internet de las cosas (IoT). Los proyectos DIAS2P y StreetQR en los que se enmarca nuestra labor pertenecen claramente a estas dos categorías, y por lo tanto, debemos completar nuestro análisis con un estudio del consumo y las dimensiones que exige cada una de las implementaciones propuestas.

Tanto DIAS2P (control peatonal en pasos de cebra) como StreetQR (información proporcionada desde las placas rotuladoras de las calles) asumen como axioma de trabajo que el dispositivo de cómputo va a estar integrado en la señalización vial, lo que concede un pequeño margen para que nuestros dispositivos sean viables en términos de consumo y dimensiones.

Hasta aquí se han mostrado comparativas en rendimiento con CPUs y GPUs procedentes de la informática de consumo más popular, pero más que nada para tener una referencia de las aceleraciones logradas. La energía y el habitáculo que demandan quedan muy por encima de lo que se exige en estos proyectos, y por lo tanto, quedan fuera de la ecuación final de nuestro trabajo, donde sólo avanzan la Jetson TX2 y la Jetson Nano. Nuestro último análisis está dedicado a confrontarlas en estos dos aspectos, aún manteniendo la GTX 960 como referencia base.

La tabla 5.3 resume consumo y dimensiones para estas tres plataformas. Tomando como referencia el rendimiento de Yolo-v3, GTX960 y Jetson Nano empatan en 15 FPS, mientras que Jetson TX2 llega al doble, 30 FPS. Esta mejora se consigue sacrificando mucho consumo, un 50 % respecto a Jetson Nano, y sobre todo las dimensiones físicas, que aumentan casi un orden de magnitud (es 9.17 veces más

grande).

Para las especificaciones que se exigen en nuestros proyectos, consumo y dimensiones son magnitudes prioritarias, pero el rendimiento no lo es tanto a partir de un umbral crítico que consideramos mínimamente superado con los 15 FPS proporcionados por la Jetson Nano, que se convierte así en la plataforma predilecta de cara a formar parte de un producto comercial patentado como resultado de los proyectos mencionados. Además, es el hardware más económico de todos, si bien se ha conseguido con un descuento promocional de pre-lanzamiento en el congreso GTX'19.

Jetson TX2 también reduce su coste hasta los 299 dólares si se adquiere como plataforma educacional, pero aún así sigue multiplicando por 3 el precio inicial de la Jetson Nano, por 1.5 el consumo máximo y por 9.17 el espacio ocupado. A duras penas hubiera logrado validar las especificaciones requeridas por nuestros proyectos, donde hubiera sido necesario reformular los prototipos de trabajo de no ser porque la Jetson Nano apareció justo en el momento oportuno para que pudiera ser planteada como una alternativa válida.

	GTX 960	**Jetson TX2**	**Jetson Nano**
Anchura x Longitud	139 x 267 mm.	170 x 170 mm.	45 x 70 mm.
Área de la placa base	371.13 cm^2	289 cm^2	31.50 cm^2
Consumo	120 W.	7.5 W.	5 W.
Coste	199 dólares	599 dólares	99 dólares

Tabla 5.3: *Comparativa de dimensiones físicas y consumo energético para cada cada una de las plataformas hardware consideradas a lo largo de nuestra implementación. Se ha tomado como referencia su coste a fecha de lanzamiento en el mercado minorista.*

6

Resumen y conclusiones

Este Trabajo Fin de Grado tuvo su origen en las necesidades de implementación en tiempo real y espacio reducido de una aplicación que permitiera detectar objetos con el menor consumo energético posible. En un estudio preliminar se decidió orientar el diseño hacia el entorno de las GPUs por su contrastado éxito en estas tres vertientes (rendimiento, espacio y consumo) y por la experiencia previa del equipo investigador. Una vez analizados los resultados, consumo y espacio han resultado mejor de lo esperado por la irrupción del modelo Jetson Nano, mientras que el rendimiento ha quedado ligeramente por debajo de la cota de 20 FPS que se planteó inicialmente, y que asumimos como el peaje a pagar por las otras dos grandes mejoras.

La alternativa hubiera sido la GPU Jetson TX2, que ha terminado demostrando su buen rendimiento, aunque con concesiones apreciables en consumo y dimensiones. En general, los objetivos del proyecto se cumplen con un notable alto para la

implementación en Jetson Nano y con un aprobado justo para la Jetson TX2.

El marco temporal de ejecución del proyecto ha limitado implementaciones más sofisticadas, tanto en la eventual selección de una plataforma basada en FPGAs como en otras redes neuronales que exigieran un mayor esfuerzo de entrenamiento y optimización. Ezequiel López Rubio, colaborador en estos proyectos y experto en la materia, trazó la hoja de ruta que hemos seguido para probar inicialmente Mask-RCNN debido a su alta precisión, y posteriormente Yolo para relajar los requerimientos computacionales y aspirar a la respuesta en tiempo real requerida, manteniendo los resultados operativos dentro de unos márgenes convincentes.

Nuestra intención inicial era paralelizar la red neuronal seleccionada utilizando CUDA y las librerías de Deep Learning que proporciona NVIDIA, pero estudiando resultados recientes publicados al respecto encontramos implementaciones ya realizadas e incluso mejoradas en versiones sucesivas, que han sido las que finalmente hemos analizado durante nuestro trabajo. Cabe mencionar que en esta etapa preliminar completé varios cursos de formación a través del DLI (Deep Learning Institute) de NVIDIA para familiarizarme con Python y Deep Learning en el contexto de las GPUs, además de fortalecer mis habilidades con CUDA, descontando un escenario más exigente para implementar la red que el que finalmente se ha plasmado en este documento.

No obstante, el camino no ha estado exento de dificultades y desafíos. La red seleccionada, Yolo, fue bastante complicada de instalar en las plataformas hardware utilizadas, ya que en general se trata de un universo muy dinámico en el que las versiones de cada librería y entorno evolucionan rápidamente, conformando un ecosistema frágil que suele dar numerosos errores por dependencias entre los módulos a integrar conjuntamente. Para solventar estas dificultades se hizo uso del gestor de dependencias Anaconda (descrito en el Apéndice), lo que facilitó la instalación final y la hizo más robusta.

Mientras se trabajaba en el correcto funcionamiento de la versión 2 de Yolo, apareció la versión 3 y de nuevo Anaconda ayudó a la instalación, que contando ya con la experiencia previa, conseguimos llevar a buen puerto.

En resumen, pienso que, la línea de investigación de mi proyecto, me ha brindado la oportunidad de descubrir el áreas de las redes neuronales y sus aplicaciones, así como el uso de numerosas herramientas que desconocía. Todo ello me ha aportado unos conocimientos que espero desarrollar a lo largo de mi carrera profesional.

6.1. Futuras mejoras

En 2019, NVIDIA ha presentado TensorRT. TensorRT es una plataforma de aprendizaje profundo de alto rendimiento que incluye optimizadores para reducir el delay. Para la implementación de éste, es necesario una tarjeta gráfica de última generación, por ejemplo, NVIDIA Jetson Xavier, aunque este es muy superior (cercano a los 3.000 dólares), por lo que dependería del coste de los diseños del proyecto la posibilidad de incorporarlos al producto comercial final.

Ésta podría ser una nueva mejora del sistema, implantar *YOLO v3 tiny* en una GPU que permitiese dicha optimización. De esta manera, se podría estudiar de nuevo estos sistemas de detección de objetos y analizar su rendimiento.

Otra mejora podría ser el estudio de nuevos sistemas de detección de objetos que destaquen por su alto rendimiento, para intentar encontrar algún sistema nuevo que ofrezca mejores resultados que YOLO.

El mundo de las redes neuronales y las GPUs están en continua expansión y es de esperar que los resultados mostrados en este trabajo queden pronto mejorados con nuevas versiones de unas y otras, aunque nuestro interés principal era mostrar la viabilidad de la implementación elegida a fecha de hoy para aportarla como punto de partida inicial. Confiamos en que desarrollos futuros permitan mejorar los principales aspectos desde dentro de esta línea de trabajo basada en GPUs, tanto desde el punto de vista del coste como de la eficiencia energética y la miniaturización del producto final.

7

Bibliografía

[1] AlexeyAB (Accessed on 2019-06). Darknet. https://github.com/AlexeyAB/darknet#requirements.

[2] Alvarado, M. (Accessed on 2019-05). Researchgate. https://www.researchgate.net/figure/Red-neuronal-artificial-de-cuatro-capas_fig1_323985249.

[3] Anaconda (Accessed on 2019-05). Anaconda distribution. https://www.anaconda.com/distribution/.

[4] Arstechnica (Accessed on 2019-05). Nvidia tesla v100: First volta gpu is one of the largest silicon chips ever. https://arstechnica.com/gadgets/2017/05/nvidia-tesla-v100-gpu-details/.

[5] Briega, R. E. L. (Accessed on 2019-06). Redes neuronales convolucio-

nales con tensorflow. https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/.

[6] Calvo, D. (Accessed on 2019-06). Red neuronal convolucional cnn. http://www.diegocalvo.es/red-neuronal-convolucional/.

[7] de Málaga, U. (Accessed on 2019-05). Unidad de métodos numéricos. http://catalogoinfraestructuras.uma.es/metodos-numericos.php.

[8] FayerWayer (Accessed on 2019-05a). Arquitectura kepler: Lo que debes saber de la nueva nvidia geforce gtx 680 y sus derivados. https://www.fayerwayer.com/2012/03/arquitectura-kepler-para-escritorio-nvidia-geforce-gtx-680/.

[9] FayerWayer (Accessed on 2019-05b). Conoce al nuevo gpu nvidia maxwell gm204. https://www.fayerwayer.com/2014/09/conoce-al-nuevo-gpu-nvidia-maxwell-gm204/.

[10] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

[11] Garea, A. S., Heras, D. B., and Argüello, F. (2019). Caffe cnn-based classification of hyperspectral images on gpu. *The Journal of Supercomputing*, 75(3):1065–1077.

[12] Guerrero, J. (2011). Autovalores y autovectores en clase matriz con plantillas (template): C/c++. https://joseguerreroa.wordpress.com/2011/10/04/autovalores-y-autovectores-en-clase-matriz-con-plantillas-template-cc/. Accessed on 2017-06.

[13] HDTecnología (Accessed on 2019-05). Nvidia muestra documentos de la arquitectura pascal. https://www.hd-tecnologia.com/nvidia-muestra-documentos-la-arquitectura-pascal/.

[14] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (Accessed on 2019-05). Mask R-CNN. *CoRR*, abs/1703.06870.

[15] Hess, C. P., Mukherjee, P., Han, E. T., Xu, D., and Vigneron, D. B. (2006). Q-ball reconstruction of multimodal fiber orientations using the spherical harmonic basis. *Magnetic Resonance in Medicine*, 56(1):104–117.

[16] Intel (Accessed on 2019-05). Procesador intel® core™2 quad q8300. https://ark.intel.com/content/www/es/es/ark/products/39107/intel-core-2-quad-processor-q8300-4m-cache-2-50-ghz-1333-mhz-fsb.html.

[17] Malcolm, J. G., Shenton, M. E., and Rathi, Y. (2010). Filtered multitensor tractography. *IEEE transactions on medical imaging*, 29(9):1664–1675.

[18] Matplotlib (Accessed on 2019-06). Matplotlib. https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.xlabel.html.

[19] Notebook, J. (Accessed on 2019-06). The jupyter notebook. https://jupyter.org/.

[20] NVIDIA (2018). Cuda toolkit documentation. http://docs.nvidia.com/cuda. Accessed on 2017-06.

[21] NVIDIA (Accessed on 2019-04a). Nvidia. https://www.nvidia.com/es-es/.

[22] NVIDIA (Accessed on 2019-04b). Nvidia applications. https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/catalog/.

[23] NVIDIA (Accessed on 2019-05a). Cuda toolkit 10.0 archive. https://developer.nvidia.com/cuda-10.0-download-archive.

[24] NVIDIA (Accessed on 2019-05b). Jetson nano. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/.

[25] NVIDIA (Accessed on 2019-05c). Nvidia cudnn. https://developer.nvidia.com/cudnn.

[26] NVIDIA (Accessed on 2019-05d). Nvidia jetson tx2. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.

[27] NVIDIA (Accessed on 2019-05e). Nvidia jetson tx2 delivers twice the intelligence to the edge. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.

[28] NVIDIA (Accessed on 2019-05f). Nvidia nano jetson. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/.

[29] NVIDIA (Accessed on 2019-05g). Nvidia raytracing. https://developer.nvidia.com/rtx/raytracing.

[30] NVIDIA (Accessed on 2019-05h). Tarjeta gráfica geforce gtx 960. https://www.nvidia.es/object/geforce-gtx-960-es.html.

[31] NVIDIA (Accessed on 2019-06). Nvidia tensorrt. https://developer.nvidia.com/tensorrt.

[32] OpenCV (Accessed on 2019-05). Opencv. https://opencv.org/.

[33] Oyama, Y., Ben-Nun, T., Hoefler, T., and Matsuoka, S. (2018). {\mu}-cudnn: Accelerating deep learning frameworks with micro-batching. *arXiv preprint arXiv:1804.04806*.

[34] pyimagesearch (Accessed on 2019-06). Mask r-cnn with opencv. https://www.pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/.

[35] Redmon, J. and Farhadi, A. (2018a). Yolov3: An incremental improvement. *arXiv*.

[36] Redmon, J. and Farhadi, A. (2018b). Yolov3: An incremental improvement. *arXiv*.

[37] Redmon, J. C. (Accessed on 2019-05). Yolo: Real-time object detection. https://pjreddie.com/darknet/yolov2/.

[38] Ricco, J. (Accessed on 2019-06). What is max pooling in convolutional neural networks? https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks.

[39] SmartCampus (Accessed on 2019-06). Dias2p + streetqr - i plan de smart-campus. http://eventos.uma.es/29978/detail/dias2p-streetqr-i-plan-de-smart-campus.html.

[40] TechLeer (Accessed on 2019-06). Mask r-cnn: Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://www.techleer.com/articles/528-mask-r-cnn-mask-r-cnn-for-object-detection-and-instance-segmentation-on-ke

[41] Tokura, H., Nishimura, T., Ito, Y., Nakano, K., Kasagi, A., and Tabaru, T. (2019). Fast algorithm of unified layer performing convolution and average pooling on the gpu. *Bulletin of Networking, Computing, Systems, and Software*, 8(1):45–49.

[42] UBUNTU (Accessed on 2019-04). Ubuntu 18.04.2 lts (bionic beaver). http://releases.ubuntu.com/18.04/.

[43] Weng, J., Ahuja, N., and Huang, T. S. (1992). Cresceptron: a self-organizing neural network which grows adaptively. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 576–581. IEEE.

[44] Wuttke, J. and Juelich, F. (2015). Levenberg-marquardt least-squares minimization. http://apps.jcns.fz-juelich.de/man/lmmin.html. Accessed on 2017-06.

[45] Xiao, W., Chen, C., Miao, Y., Xue, J., and Wu, M. (http://sites.nlsde.buaa.edu.cn/ /xiaowencong/resxiao_sosp17aisys.pdf). Optimization mapping for deep learning.

[46] Zhan, L., Leow, A. D., Zhu, S., Chiang, M.-C., Barysheva, M., Toga, A. W., McMahon, K. L., De Zubicaray, G. I., Wright, M. J., and Thompson, P. M. (2009). Analyzing multi-fiber reconstruction in high angular resolution diffusion imaging using the tensor distribution function. In *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*, pages 1402–1405. IEEE.

A

Manual de usuario

A.1. Instalación de librerías

Para la instalación de las librerías, se pueden utilizar los gestores de dependencias. En este TFG, se utilizó para la instalación en el equipo del laboratorio, ya que tanto NVIDIA Jetson Nano como NVIDIA Jetson TX2 no eran compatibles con este gestor. Para aquellos dispositivos que no sean compatibles con Anaconda, los paquetes se instalaran mediante el comando 'pip' o directamente.

```
pip install --upgrade library
sudo apt-get install library
```

A.1.1. Anaconda

Primero, se tendrá que descargar el instalador de su página oficial. Tras instalarlo, se creará un entorno y podremos instalar las librerías que se necesiten. Existe la posibilidad de importar un entorno ya existente.

```
conda env create -f environment.yml
conda activate enviroment
conda install library
conda deactivate
```

Para salir de este entorno sólo tendremos que ejecutar el siguiente comando:

```
conda deactivate
```

A.2. Gráficas en python

Para la obtención de las gráficas es necesaria la instalación de 'jupyter notebook'. Esta instalación se puede hacer en el entorno creado anteriormente.

```
conda install jupyter notebook
```

Una vez instalado, ejecutamos el comando jupyter notebook en el entorno:

```
jupyter notebook
```

Esto abrirá una nueva pestaña en el navegador con jupyter. Crearemos un nuevo fichero seleccionando como tipo de fichero el entorno que hemos creado. *(Ver en la figura A.1.*

Cuando tengamos ya el fichero creado, escribimos el siguiente código en python.

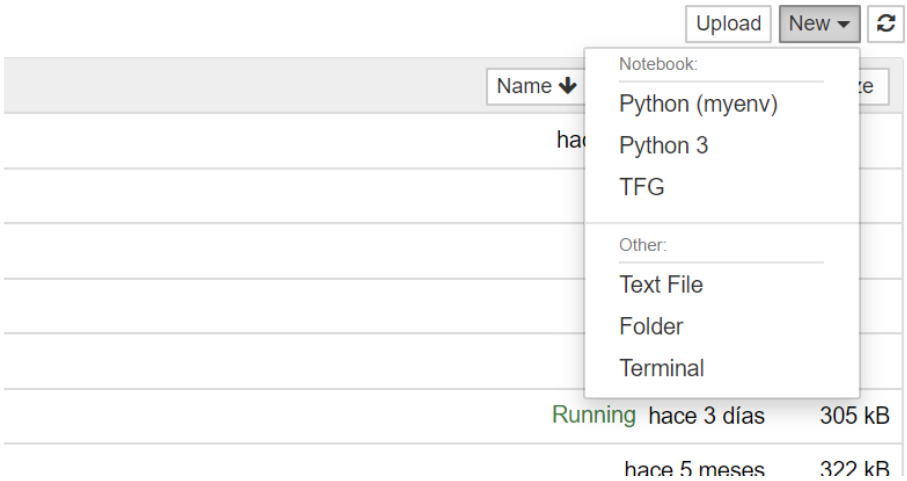


Figura A.1: *Seleccionamos el entorno creado.*

```
import numpy as np
import matplotlib.pyplot as plt
f = open("Datos/GTX 960/YOLOV2-CPU.txt")
array_yolov2_cpu = []
t=0.0;
contador=1;
linea = f.readline()
while(contador<=20):
  x=float(linea)
  array_yolov2_cpu.append(x)
  t+=x;
  contador+=1

  linea= f.readline()
f.close()
media_yolov2_cpu = t/contador
print('%f', media_yolov2_cpu)
plt.xlabel('EJECUCIONES')
plt.ylabel('START UP')
plt.plot(array_yolov2_cpu)
```

A.3. Instalación de YOLO

Una vez instalada las librerías necesarias, clonamos el repositorio de YOLO y compilamos:

```
git clone https://github.com/pjreddie/darknet
cd darknet
make
```

Si por falta de tiempo es necesario la red pre-entrenada, descargamos los pesos de la siguiente forma:

```
wget https://pjreddie.com/media/files/yolov3.weights
```

Finalmente, ejecutamos la red en la terminal:

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```