Implementación de redes neuronales y análisis de rendimiento para sistemas empotrados

Sara Esteban Uribe

Resumen - Dentro de unos años las carreteras empezarán a tener vehículos sin conductor para trasladar a personas. Este acontecimiento nos lleva a utilizar redes neuronales. En este proyecto se propone realizar la implementación de diferentes redes neuronales convolucionales en un sistema embebido para aplicar en un caso práctico real. Se entrenarán las redes neuronales y posteriormente se ejecutarán en la plataforma de bajo consumo Jetson TX2. Se analizará el rendimiento medido en fotogramas procesados por segundo (fps) y se medirá el consumo en vatios de la ejecución de estos algoritmos. Los resultados demuestran que la red de mayor rendimiento en nuestro caso es la red de detección de objetos con un tiempo de 13 fps con el modo de ejecución Max-N y donde el código empleado para la ejecución en tiempo real llega casi al 100% de su rendimiento.

Palabras clave - Jetson TX2, DIGITS, consumo eléctrico, red neuronal, segmentación, clasificación, detección de objetos, dataset, rendimiento, GPU.

Abstract - In a few years, roads will begin to have vehicles without drivers to move people. This event leads us to use neural networks. In this project, we propose the implementation of different convolutional neural networks in an embedded system to be applied in a real practical case. The neural networks will be trained and later they will be executed in the low consumption platform Jetson TX2. The performance measured in processed frames per second (fps) will be analyzed and the consumption in watts of the execution of these algorithms will be measured. The results show that the network with the highest performance in our case is the object detection network with a time of 13 fps with execution mode Max-N and where the code used for real-time execution reaches almost at 100% of its performance.

Index Terms— Jetson TX2, DIGITS, electric consumption, neural network, segmentation, classification, object detection, dataset, performance, GPU.

1 INTRODUCCIÓN

Históricamente, el transporte ha sido uno de los grandes avances de la humanidad. La conducción tradicional ha dado lugar a diferentes investigaciones en el ámbito de la conducción automatizada como son la seguridad para reducir accidentes causados por errores humanos, la eficiencia y objetivos ambientales para conseguir un tráfico más fluido que ayudará a disminuir el consumo de energía y emisiones de los vehículos, la comodidad para habilitar al usuario la libertad para otras actividades cuando los sistemas automatizados están activos, y por último la inclusión social para asegurar movilidad para todos [1]. Por este motivo, desde hace años se están impulsando tecnologías de conducción inteligente.

Dentro de las causas mencionadas anteriormente, la seguridad es el principal punto de mejora, debido a que el factor humano es una de las principales causas de los accidentes de tráfico. En el año 2017 se registran 1.067 accidentes mortales en vías interurbanas, en los que han fallecido 1.200 personas y 4.837 heridas hospitalizadas, lo que supone un aumento del 3% en lo que a accidentes mortales y fallecidos se refiere y una disminución de un 6% en lo relativo a heridos hospitalizados [2]. Esto nos demuestra que los accidentes de tráfico en vías urbanas siguen estando presentes, aunque sea en mayor o menor medida. No obstante, se pretende reducir aún más este índice de accidentes incorporando nuevas tecnologías como son los coches autónomos o semiautónomos.

Una de las metodologías que está cobrando más importancia en el campo automovilístico, son las redes neuronales [3]. Las redes neuronales son modelos matemáticos para el tratamiento de la información inspirado en modelos biológicos, donde su unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona. La importancia de las redes neuronales reside en dos puntos fundamentales. Por un lado, las redes neuronales generan patrones de reconocimiento a partir del procesamiento de muestras predefinidas en forma de imágenes, manuscritos o secuencias de tiempo [3]. Por otro lado, podemos decir que la capacidad de reconocimiento del patrón puede mejorar mediante el aumento de tamaño del muestreo. Por ello se dice que las redes neuronales poseen capacidad de aprendizaje. Aun así, el tiempo de procesamiento de datos y generación de patrones de una red neuronal es dependiente de la complejidad de ésta y del tamaño de las muestras, conllevando una gran necesidad de capacidad computacional [4].

Las GPUs se ajustan bien a los cálculos de una red neuronal porque tienen muchos más recursos de cómputo y un mayor ancho de banda para la memoria [5]. Gracias a la gran capacidad computacional que proporcionan las GPUs, el entrenamiento de una red neuronal se puede ver reducido de semanas a días. Esto

hace que las GPUs sean una solución muy eficiente para la computación de alto rendimiento (HPC) [5].

El proyecto está dirigido a la implementación de diferentes tipos de redes neuronales convolucionales tales como la red neuronal de segmentación, clasificación y detección de objetos. Cada una de ellas estará orientada a un caso práctico en concreto. Se hará el análisis de rendimiento de las mencionadas redes neuronales y se definirá cuál es la que mayor eficiencia tiene para ejecutarse en tiempo real en un sistema empotrado de bajo consumo.

2 OBJETIVOS

En el presente trabajo, los siguientes objetivos están orientados a la implementación de tres tipos redes neuronales de convolución en la plataforma Jetson TX2 [6]. En primer lugar, se implementará una red neuronal de segmentación para delimitar diferentes elementos en una imagen. Seguidamente, se implementarán las redes de clasificación y de detección de objetos, en un caso práctico que tiene como objetivo observar la eficiencia de estas para localizar un objeto en específico en el espacio.

Con las mencionadas redes neuronales, se crearán métricas de rendimiento medidas en fotogramas procesados por segundo (FPS) y métricas de consumo medidas en vatios.

El objetivo global de este trabajo es saber escoger para cada situación en específico que red convolucional es la más indicada a utilizar, implementarla y saber cuál es su rendimiento en un sistema empotrado, para obtener la mayor eficiencia en la ejecución en tiempo real.

3 ESTADO DEL ARTE

En la actualidad, los algoritmos de *machine learning* se utilizan ampliamente para encontrar soluciones a los diversos desafíos que surgen en los coches autónomos. Una de las principales tareas de un algoritmo de aprendizaje automático es la representación continua del entorno y la previsión de los cambios para este. Estas tareas se clasifican en la detección, la identificación, la localización de un objeto y la predicción de su movimiento [7].

No obstante, implementar una red neuronal en sistemas empotrados es un problema ya que tienen las desventajas de requerir un gran número de operaciones de coma flotante y además tienen tiempos de ejecución prolongados que dificultan su usabilidad en tiempo real [7].

Google, por ejemplo, está realizando una inversión notable en el desarrollo de vehículos autónomos. Sus prototipos de vehículos autónomos incorporan tecnología *deep learning* que ya puede detectar peatones en varios escenarios desafiantes. Su proyecto estrella es el coche autónomo Waymo [8]. Estos sistemas de *deep learning* han logrado un rendimiento excepcional, lo que hace que la tasa de error para la visión artificial sea inferior a la de los seres humanos (el índice de referencia humano es una tasa de error del 5 por ciento). Este logro, también es debido a las nuevas arquitecturas hardware que usan múltiples GPU. Estas, están impulsando la migración de características basadas en la tecnología tradicional de procesamiento de imágenes a soluciones basadas en *deep learning* [3].

4 METODOLOGÍA

El proyecto es un desarrollo iterativo e incremental que tiene como objetivo un crecimiento progresivo de la funcionalidad del proyecto. En otros términos, el trabajo va evolucionando con cada una de las entregas planificadas hasta que se ajusta a lo esperado por el destinatario [9].

Una característica de este método es que, en cada una de las entregas, el proyecto debe mostrar una evolución con respecto a la anterior; nunca dos entregas pueden ser iguales. El responsable del proyecto debe analizar si los resultados parciales son los esperados y si apuntan al objetivo principal. En cada iteración, se pueden realizar cambios y se pueden añadir nuevas funcionalidades y capacidades. Lo cual se ajusta mucho al desarrollo que he implementado ya que este proyecto ha sido variable en el tiempo según los resultados durante el desarrollo.

5 DESARROLLO

Contamos con un conjunto de fases de desarrollo que se pueden separar en seis grandes bloques: (i) Introducción a las redes neuronales convolucionales, (ii) Implementación de una red neuronal de segmentación, (iii) Implementación de la red neuronal de clasificación, (iv) Implementación de una red neuronal de detección de objetos, (v) Caso práctico y (vi) Obtención de medidas de energía consumida y rendimiento. En el primer capítulo se explicará el concepto de red neuronal convolucional y las diferentes capas que podemos encontrar en este tipo de red. Los siguientes tres capítulos constan de una pequeña descripción con el objetivo de cada red, juntamente con la descripción de su arquitectura y sus características. El quinto se explicará en que consiste el caso práctico que se va a desarrollar y en el último capítulo se explicará el proceso para obtener datos de energía en la plataforma Jetson TX2 y cómo se puede mejorar el rendimiento.

5.1 Introducción en las redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) se utilizan principalmente para el procesamiento de imágenes. Eliminan la necesidad de una extracción manual de características, por lo que no es necesario identificar las características utilizadas para clasificar las imágenes puesto que se aprenden durante el entrenamiento. Esta extracción de características automatizada hace que los modelos aprendidos sean precisos para tareas de visión artificial, tales como la clasificación de objetos [10].

Las redes neuronales convolucionales, contienen tres tipos de capas diferentes: las de convolución (CONV), las de agrupamiento (*pooling*) y las totalmente conectadas (*fully-connected layers*, FC) [11].
La capa de convolución recibe como entrada la imagen y luego aplica sobre ella un filtro que nos devuelve un mapa de características. La capa de reducción o *pooling* va posicionada generalmente después de la capa convolucional. Esta capa consiste en la reducción de las dimensiones espaciales (ancho x alto) del volumen de entrada para la siguiente capa convolucional. La operación realizada por esta capa también se conoce como reducción de muestreo, ya que la reducción de tamaño conduce a la pérdida de información. Esta pérdida puede ser beneficiosa porque provoca una menor sobrecarga de cálculo para las próximas capas de la red. En conclusión, esta capa se queda con las características más comunes de la imagen de entrada [12].

Por último, al final de las capas convolucionales y de *pooling*, las redes normalmente usan capas *fully-connected* en la que cada píxel se considera como una neurona separada al igual que en una red neuronal regular. Esta última capa clasificadora tendrá tantas neuronas como el número de clases que se debe predecir [13].

Para el entrenamiento de las redes neuronales convolucionales es muy común utilizar modelos pre entrenados. De esta forma, tendremos un esquema de inicialización de pesos adecuado y el aprendizaje de nuestra red será mucho más rápido y fiable.

5.2 Red neuronal de segmentación

Las redes neuronales de segmentación se emplean para localizar o para descubrir los límites de objetos dentro de una imagen [14]. Las imágenes se dividen en píxeles y cada píxel es clasificado. Por cada píxel de la imagen, la red se entrena con el fin de predecir de qué clase forma parte ese píxel en particular. Esto permite que la red identifique varias clases y también que determine la ubicación de los objetos. La segmentación de imágenes normalmente genera una imagen etiquetada del mismo tamaño que la entrada cuyos píxeles están codificados por colores según sus clases [14]. Si las imágenes de entrada son de una resolución asumible, el número de elementos que habrá que entrenar será lo suficientemente moderado como para que pueda funcionar correctamente, aunque todo el proceso será significativamente más lento. Sin embargo, si la resolución aumenta los tiempos de entrenamiento y testeo se ven afectados y se vuelven enormes [15].

Es muy común dividir la red en dos partes: en la primera parte, el extractor de características, los datos pasan por varias capas convolucionales para extraer progresivamente características más complejas y abstractas. En la segunda y última parte, el clasificador consta de varias capas *fully-connected*, la primera de las cuales recibe sus entradas del extractor de características. Estas capas aprenden relaciones complejas entre las características para dotar a la red de un alto nivel de comprensión de los contenidos de la imagen [16].

El *dataset* de este tipo de redes contiene un conjunto de imágenes las cuales estarán divididas en imágenes originales e imágenes etiquetadas. Para realizar el modelo solo es necesario el *dataset* y una descripción de la red donde se detallan las capas que contiene y sus las características de cada una de ellas.

5.3 Red neuronal de clasificación

La red neuronal de clasificación es un tipo de red mucho más sencillo que la red de segmentación. La función de esta red es detectar que hay en la imagen. Dada una imagen de entrada, la red genera la probabilidad para cada clase existente.

Al igual que en el modelo de segmentación es necesario definir un conjunto de datos de entrada. Tener un *dataset* significativo es una tarea muy importante porque si el *dataset* no es el correcto el resultado del entrenamiento de la red no será el esperado. El *dataset* en el caso de la red neuronal de clasificación se separa mediante el uso de carpetas. El nombre de la carpeta es la etiqueta de la imagen o nombre de la clase.

5.4 Red neuronal de detección de objetos

El objetivo de un sistema de detección de objetos es detectar todas las instancias de objetos de una categoría conocida en una imagen.

Un buen sistema de detección de objetos tiene que ser robusto para la presencia o ausencia de objetos y ser invariante respecto al tamaño, el punto de vista y la orientación, y ser capaz de detectar objetos parcialmente

ocluidos. Las imágenes del mundo real pueden contener algunas instancias de objetos o un número muy grande; esto puede tener un efecto sobre la precisión y la eficiencia computacional de un sistema de detección de objetos. La mayoría de las redes de detección de objetos se basan en reutilizar redes de clasificación previamente entrenadas [17].

Entre las diferentes redes de detección de objetos, una de las más utilizadas es DetectNet. Esta, es una red derivada de GoogLeNet que está específicamente ajustada para la detección de objetos [17].

La imagen 1 muestra la arquitectura DetectNet utilizada durante el entrenamiento. Destaca tres procesos importantes.

1. Las capas de datos cogen las imágenes, las etiquetas de entrenamiento y una capa de transformación y aplica la estrategia de *online data augmentation* que
2. consiste en aumentar las muestras en cada *epoch* de la fase de entrenamiento sin considerar si está equilibrado o no [18].
3. La red GoogLeNet FCN, realiza la extracción de características y la predicción de clases de objetos y cuadros delimitadores.
4. Las funciones de pérdida miden simultáneamente el error en las dos tareas de predecir la cobertura del objeto y las esquinas del cuadro que delimitan el objeto [19].

Por otro lado, la imagen 2 muestra la arquitectura DetectNet durante la validación y dos procesos clave más.

1. Una función de *clustering* [20] produce el conjunto final de cuadros delimitadores predichos durante la etapa de validación.
2. Se calcula una versión simplificada de la métrica de precisión media (mAP) para medir el rendimiento del modelo frente al conjunto de datos de validación [19].

   El *dataset* para la red de detección de objetos requiere más información para el entrenamiento comparado con la red de clasificación. En este caso, se tienen las imágenes originales en una carpeta y en otra las etiquetas de estas imágenes en un fichero de texto. El contenido de este fichero es el nombre de la clase y unos parámetros, como por ejemplo las coordenadas del cuadro delimitador (bbox).

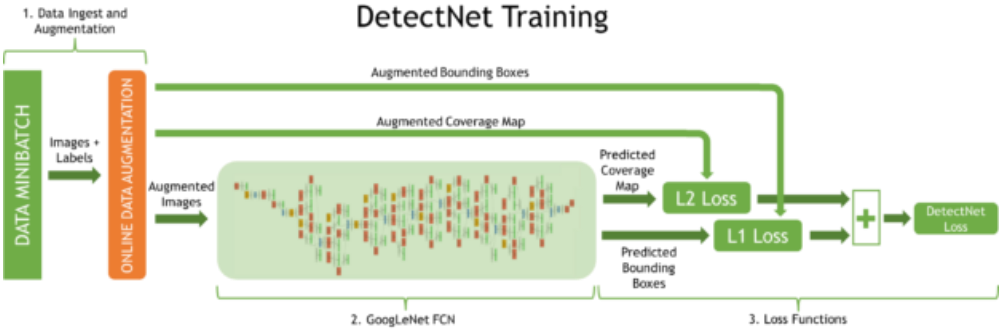


Imagen 1. Arquitectura DetectNet durante el entrenamiento [19]

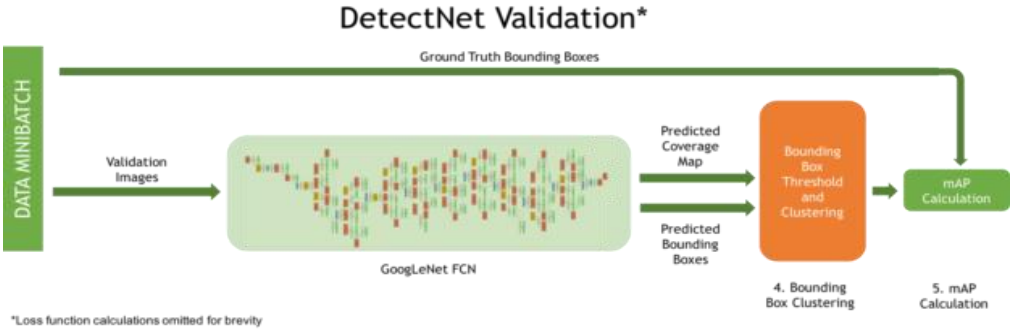


Imagen 2. Arquitectura DetectNet durante la validación [19]

5.5 Caso práctico

Se desea diseñar una red neuronal para encontrar un triángulo rojo en la imagen y decir si está en el centro, a la derecha o a la izquierda, y además decir si el triángulo es pequeño, mediano o grande.

Esta propuesta servirá para informar a un algoritmo de control que debe mover un coche, girando y avanzando o retrocediendo, para colocar el triángulo en el centro de la pantalla y con un tamaño mediano. De esta forma, podríamos provocar que un coche siguiera a un triángulo dirigido por una persona.

El triángulo, es principalmente equilátero, pero se añadirá un tipo de deformación, la cual consiste en aplanar el triángulo desplazando el vértice superior en el eje y. De este modo, consideramos el caso de sujetar el triángulo e inclinarlo hacia delante o hacia atrás. Esto se puede observar en la imagen 3. Este tipo de deformaciones se conoce como *data augmentation* [21]. Esto nos permite tener un *dataset* mucho más fiable y que si el triángulo se mueve o se deforma, la red siga identificando un triángulo y lo tenga en cuenta. A parte de aplanar el triángulo, también se varia su tamaño.

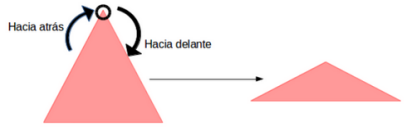


Imagen 3. Deformación aplicada al triángulo en el algoritmo para la creación del dataset.

Se deben añadir tonalidades de rojo al crear el triángulo, puesto que las cámaras generalmente no perciben el color primario rojo (255,0,0) entonces cuando se crean los triángulos debe variar su color dentro de la gama de las tonalidades de rojo.

5.6 Obtención de energía consumida y rendimiento

La plataforma Jetson TX2 consiste en una GPU con una frecuencia de 1,3 Ghz. También cuenta con un clúster de CPU que consiste en un procesador Denver 2 de doble núcleo y un ARM Cortex-A57 de cuatro núcleos (Sección A1). En la placa los núcleos de las CPU pueden estar en línea o fuera de línea, excepto el núcleo CPU0, que siempre está encendida [22].

Esta placa permite usar cinco modos diferentes de arranque. La tabla 1 muestra los modos disponibles, qué núcleos de CPU estarán activos y la frecuencia máxima de CPU y GPU que utiliza.

Tabla 1. Características de los diferentes modos disponibles para la Jetson TX2.

Mode	Name	Denver2	Freq (GHz)	ARM A57	Freq (GHz)	GPU Freq (Ghz)
0	Max-N	2	2	4	2	1,3
1	Max-Q	0		4	1,2	0,85
2	Max-P Core-All	2	1,4	4	1,4	1,12
3	Max-P ARM	0		4	2	1,12
4	Max-P Denver	2	2	0		1,12

Entre los diferentes modelos, destacamos que Max-N activa las máximas frecuencias de reloj de CPU sacrificando el consumo de energía por un mayor rendimiento. Por otro lado, Max-Q desactiva los dos núcleos de la CPU Denver y baja la frecuencia de reloj de la CPU y GPU. Este modo ofrece la mayor eficiencia de procesamiento ya que los componentes de la Jetson TX2 están configurados con su máxima eficiencia.

La plataforma Jetson TX2 posee unos monitores de potencia, los cuales informan de los valores de voltaje, corriente y potencia de la CPU, GPU, SOC, DDR RAM y WIFI como podemos ver en la imagen 4. Se puede acceder fácilmente a ellos leyendo unos ficheros que ofrece el propio sistema y que proporcionan información sobre los dispositivos. Por lo tanto, la función de estos monitores es exportar una visión de la configuración hardware del sistema. Las medidas proporcionadas por los monitores de potencia son precisas dentro del 5%.

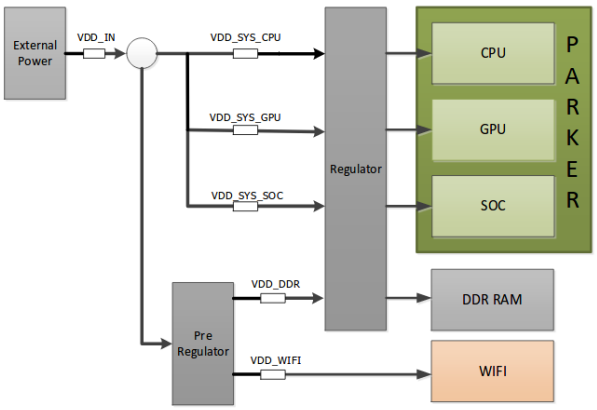


Imagen 4.Diagrama de bloques de la fuente de alimentación y los rieles que se pueden medir con los monitores de potencia incorporados en la Jetson TX2

6 RESULTADOS

En este apartado contamos con un conjunto de capítulos que se pueden separar en cinco grandes bloques: (i)Herramientas utilizadas, (ii) Red neuronal de segmentación, (iii) Red neuronal de clasificación, (iv) Red neuronal de detección de objetos y (v) Rendimiento y consumo

6.1 Herramientas utilizadas

Las redes neuronales implementadas se entrenarán en DIGITS [23]. Este es el sistema de entrenamiento de GPU NVIDIA *deep Learning* el cual se puede utilizar para entrenar rápidamente una red neuronal para tareas de clasificación de imágenes, segmentación y detección de objetos. DIGITS deja que te enfoques en diseñar y entrenar redes en lugar de programar y depurar [23]. Este software te permite implementar tu red neuronal con tres tipos de *frameworks*, Caffe, TensorFlow [24] y PyTorch [25]. En este proyecto, nos decantamos por Caffe, que es un entorno de programación dentro del ámbito del *deep learning* que soporta CNN, redes neuronales regulares y es compatible con aceleradores de GPU utilizando CUDNN [26] de Nvidia. Este *framework* puede ser usado mediante Python o C++ [27]. Cabe destacar que el software DIGITS está instalado en una máquina con una GPU GTX 1080 para acelerar la ejecución de los entrenamientos. Las ejecuciones en tiempo real se realizan sobre la plataforma Jetson TX2.

Para el proceso de optimización se hace uso de NVIDIA TensorRT [28], un optimizador de inferencia de aprendizaje profundo que maximiza el rendimiento y minimiza la latencia. TensorRT es capaz de convertir el modo de precisión FP32 a FP16 o incluso a INT8 sin una pérdida significativa de precisión, esto ofrece un mayor rendimiento y menores requisitos de memoria. No obstante, este proceso requiere mucho más que una simple conversión [28].

Con lo explicado anteriormente, el entorno de desarrollo queda reflejado en la imagen 5, donde se exponen las diferentes herramientas hardware y software que se utilizan en el proyecto.

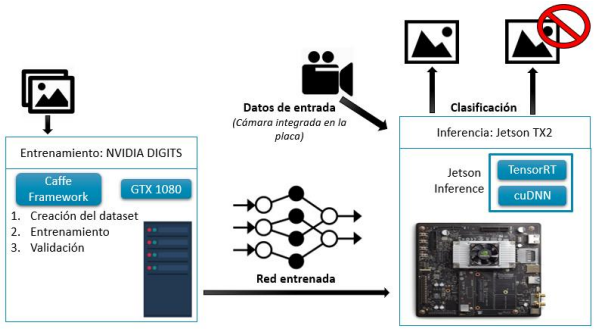


Imagen 5. Entorno de desarrollo del proyecto.

Para cambiar el consumo en la Jetson se utiliza la herramienta nvpmodel que proporciona el kit de desarrollo de la Jetson TX2. NVIDIA ofrece esta herramienta para maximizar el rendimiento y el consumo de energía en diferentes escenarios [28].

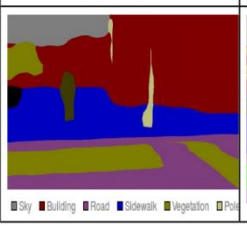
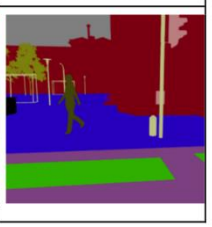
6.2 Red neuronal de segmentación

Para el *dataset* de esta red se usó el conjunto de imágenes que proporciona Synthia [29]. Se elaboró un modelo de tipo segmentación, donde su descripción se define mediante Caffe. Tanto la creación del *dataset* como la del modelo se hizo mediante DIGITS.

Para entrenar la red se utilizó un modelo pre-entrenado llamado FCN-Alexnet que incorpora DIGITS. Una vez finalizado el entrenamiento, DIGITS proporciona una curva de aprendizaje que contiene las pérdidas de entrenamiento en función del número de iteraciones. Estas son muy útiles para examinar las pérdidas y la precisión de validación. En nuestro caso, el modelo logró una precisión de validación del 92% y dejó de mejorar después de 5 iteraciones (apéndice Sección A2 imagen 17).

El resultado de hacer inferencia en una imagen desde DIGITS es el mostrado en la tabla 2 y como se puede contemplar, el resultado es bastante ajustado. La inferencia tarda 15 segundos por imagen, el cual hace referencia a la suma total de los tiempos de CPU, GPU y E/S (entrada y salida).

Tabla 2. Resultado de realizar inferencia en la plataforma DIGITS sobre la GPU GTX 1080. La ejecución es con Caffe.

Imagen de entrada	Inferencia	Ground truth
		

Una vez entrenada esta red se procede a la ejecución en la Jetson TX2. Para poner en funcionamiento la ejecución es necesario el repositorio jetson-inference (Disponible en: https://github.com/dusty-nv/jetson-inference) Este repositorio está diseñado para ser ejecutado en la Jetson TX2 y aceptar los modelos entrenados en DIGITS. A su vez, contiene seis programas en c++ para cada tipo de red neuronal, es decir, dos programas para cada tipo de red. El primer programa sirve para hacer inferencia sobre una imagen y el segundo hace inferencia en tiempo real. Estos programas vienen incorporados con TensorRT.

El resultado de la ejecución es el presentado en la imagen 6. El resultado no es tan eficiente como el observado en la GPU GTX 1080 (tabla 2). Se sugiere que el resultado puede ser causado a que dos capas (*deconvolution* y *crop*) de la descripción de la red no son compatibles con TensorRT. Entonces al prescindir de ellas da como resultado una inferencia poco precisa.

Otro punto crítico es que alcanza un rendimiento de ~0,8 fps en su ejecución en tiempo real.

Imagen 6. Resultado de realizar inferencia en la plataforma Jetson TX2. La ejecución es con Caffe y TensorRT.

Se ha comprobado que la red neuronal de segmentación tiene la desventaja de requerir una gran cantidad de operaciones de punto flotante y tener largos tiempos de ejecución que dificultan su usabilidad en tiempo real, como en nuestro caso. Además del problema del tiempo, el resultado de la inferencia tampoco es lo suficientemente ajustado como para fiarse de él.

6.3 Red neuronal de clasificación

La implementación de esta red se basará en el caso práctico que se ha explicado en la sección 5.5. Para la creación de las imágenes del *dataset* se implementó un algoritmo en Python que utiliza las imágenes del *dataset* de Synthia como fondo y se ubica un triángulo rojo en las diferentes secciones de la imagen y dependiendo de su tamaño y localización se guardará en una clase u otra.

Para saber en qué sección de la imagen está ubicado el triángulo se realizan tres divisiones iguales en la imagen y según la posición del centro del triángulo se guarda en una clase o en otra.

Para entrenar la red se usará la red GoogLeNet [30]. Las clasificaciones obtenidas de la red neuronal con diez clases no saben diferenciar entre izquierda y derecha, mientras que los tamaños los detecta de forma correcta. Esto nos lleva a pensar que le hace falta un aumento en la resolución de la imagen. Para aumentar la resolución, se crean subdivisiones a las clases izquierda, derecha y centro (imagen 7). Añadiendo estas subdivisiones a la imagen tendremos 19 clases.

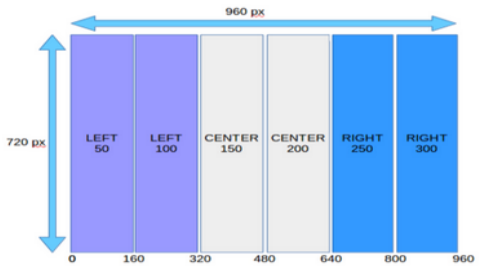


Imagen 7. Subdivisiones realizadas a las imágenes del dataset con un tamaño de 960x720 px.

Haciendo inferencia con este nuevo entrenamiento nos damos cuenta de un suceso que también pasa con el anterior entrenamiento. Cuando ponemos un triángulo en el centro nunca falla, en cambio, cuando ponemos un triángulo en los laterales sí. Pero no se equivoca de forma aleatoria, puesto que el porcentaje de duda está entre la clase opuesta a la que está analizando. En otras palabras, si se realiza inferencia con una imagen de "*left 50 small*" dudará entre *"left 50 small*" y "*right 300 small*". Esto lo podemos observar de forma más clara en la imagen 8. En consecuencia, el porcentaje más alto lo dará en función de la cantidad de imágenes que hay en el *dataset*, si el número de imágenes es igual en todas las clases, los porcentajes serian 50% a una clase y 50% a la otra.

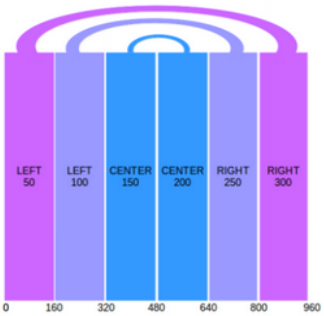


Imagen 8. Relación entre las clases cuando se realiza inferencia.

Este acontecimiento lo podemos observar en la imagen 9 que contiene el resultado de la inferencia con el modelo de 19 clases. En la sección A3 del apéndice está el resultado con el modelo de 10 clases.

Imagen 9. Resultado de realizar inferencia con el modelo de 19 clases

En cuanto a los entrenamientos (apéndice Sección A2 imágenes 15 y 16), el *accuracy* disminuye de manera que mientras más divisiones se hagan más puntos frontera hay. Si se realiza inferencia con un triángulo ocupando dos divisiones, la red no sabrá discernir cuál es la clase correcta. En el entrenamiento con 10 clases se obtiene un *accuracy* del 80% y con 19 clases tenemos un *accuracy* del 60%.

El rendimiento medido en fotogramas procesados por segundo de la red de clasificación con 19 clases es

aproximadamente 18 fps. Pese a que la red neuronal de clasificación tiene un buen tiempo de ejecución en tiempo real y diferencia los tamaños del triángulo, no es efectiva para clasificar la ubicación del objeto a encontrar.

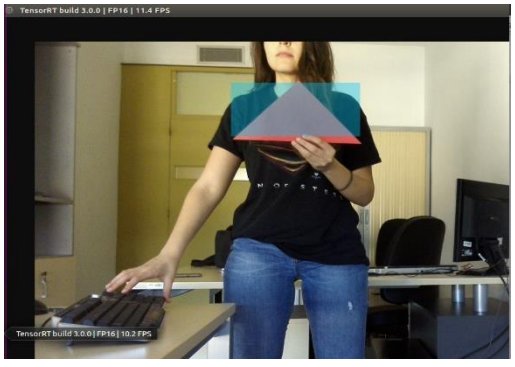
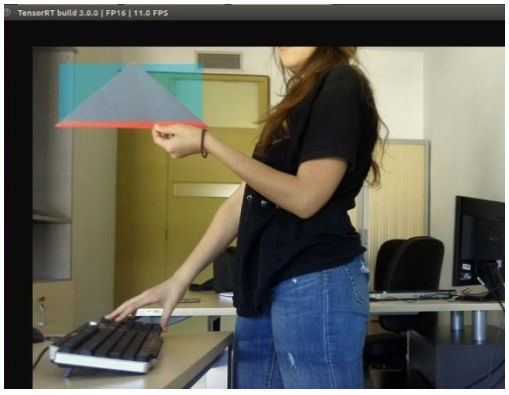
6.4 Análisis red neuronal de detección de objetos

Para la creación de este *dataset*, se reutiliza la estructura del algoritmo del modelo anterior, pero en este caso crearemos un fichero de texto (como se ha mencionado en la Sección Desarrollo 5.4) con el nombre de la clase triangulo y las coordenadas del cuadro delimitador (bbox). Todas las deformaciones realizadas en el caso de la red de clasificación se mantienen, solo se modifica la parte de la definición de las etiquetas.

Para el entrenamiento de la red, se utiliza un modelo pre entrenado de GoogLeNet. Considerando que DetectNet (Sección 5.4) se deriva de GoogLeNet, se recomienda utilizar los pesos pre-entrenados de GoogLeNet ya que esto ayudará a acelerar el entrenamiento y ayudará a que aprenda de forma significativa.

En la tabla 3 se observa lo ajustado que es el resultado en tiempo real en la plataforma Jetson TX2**.** En la sección A4 del apéndice, en la tabla 4 se distinguen los resultados de efectuar inferencia sobre una imagen en las diferentes GPUs, la GTX 1080 y la Tegra X2.

Tabla 3. Resultado de hacer inferencia en tiempo real sobre la plataforma Jetson TX2 con el modelo de detección de objetos.

Resultado ejecución en tiempo real



La red neuronal de detección de objetos alcanza un rendimiento 9 fps aproximadamente.

6.5 Rendimiento y consumo

Para medir el consumo de energía se consulta el manual que proporciona Nvidia disponible en https://developer.nvidia.com/embedded/downloads. Para hacer la medición se realiza un script que consulta unos ficheros del sistema ubicados en /sys/bus/i2c/drivers/ina3221x. Estos ficheros proporcionan información del voltaje, de la corriente y de la potencia. En la imagen 9, se observa el número de canal y de dirección que debemos consultar para obtener los datos que mencionamos anteriormente. Por ejemplo, si queremos saber la corriente de la GPU deberemos acceder a /sys/bus/i2c/drivers/ina3221x/0-0040/iio_device/in_current0_input y nos devolverá la corriente en mA.

Power Rail	<Address>	Channel	Power Rail	<Address>	Channel
VDD_GPU	0-0040	0	VDD_IN	0-0041	0
VDD_SOC	0-0040	1	VDD_CPU	0-0041	1
VDD_WIFI	0-0040	2	VDD_DDR	0-0041	2

Imagen 10. Número de canal y dirección de los diferentes power rail.

En la imagen 11 se observan las ejecuciones de las redes neuronales implementadas en este proyecto. Podemos observar que en cuanto a tiempo de ejecución la red de clasificación es la que mayor rendimiento proporciona, debido a que obtiene ~18fps y ~20fps. Sin embargo, si tenemos en cuenta el resultado de la inferencia, podemos decir que no es la ideal para aplicar en un caso práctico real. La que tiene mayor aplicación en un entorno real es la red de detección de objetos. Porque tanto en tiempo de ejecución como en resultado de inferencia es la que mayor rendimiento y aplicabilidad nos proporciona. Por esta razón, esta red sería la seleccionada para nuestro caso práctico expuesto en el trabajo

La red de detección de objetos va ~11x veces más rápida que la red de segmentación, pero la red de clasificación va ~2,2 veces más rápida que la red de detección de objetos.

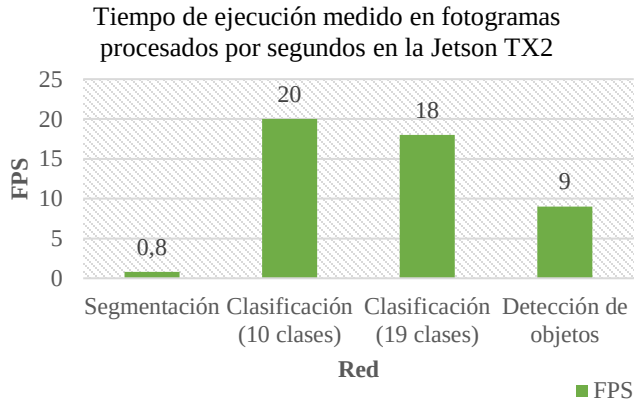


Imagen 11. Tiempo medido en fotogramas procesados por segundo de las diferentes redes neuronales empleadas en el trabajo. La ejecución es sobre la plataforma Jetson TX2.

Si analizamos más a fondo la red de detección de objetos, observamos que para una ejecución en tiempo real de 100 iteraciones ejecuta 28 funciones de GPU. Si analizamos los *kernels* que mayor porcentaje del tiempo de ejecución consumen, se observa que obtienen una utilización de la unidad de función de *half-Precision* entre medio-alta (Sección A5). El IPC (instrucciones por ciclo) de los tres *kernels* está entre 3,2 y 3,9. Este IPC indica la cantidad de instrucciones que un procesador ejecuta en un ciclo de reloj [31]. Si tenemos en cuenta que el mejor IPC es 4 podemos decir que estos *kernels* están optimizados casi al 100% de su máximo rendimiento.

Si ejecutamos la red de detección de objetos con los diferentes modos de ejecución explicados anteriormente (Sección Desarrollo 5.6) vemos que se puede llegar a mejorar el tiempo de ejecución. En la imagen 12, se puede ver que el modo MAX-N alcanza los ~13 fps. Con este resultado logramos una mejora de ~1,4x con respecto al tiempo original de 9 fps. El modo que viene por defecto es el Max-P ARM y con el de mayor eficiencia (Max-Q) obtenemos un resultado de 9,5 fps.

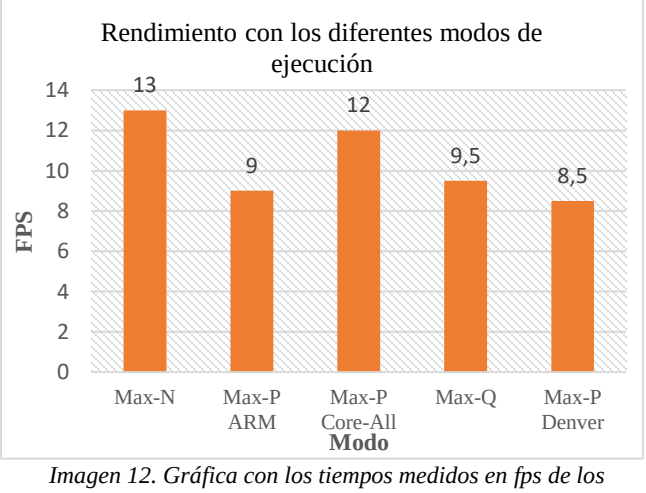


Imagen 12. Gráfica con los tiempos medidos en fps de los diferentes modos de ejecución disponibles para la Jetson TX2.

Por otro lado, si comparamos el consumo del modelo de detección de objetos, medido en vatios de los diferentes modos de ejecución. Se aprecia que el consumo de la CPU se mantiene en los cuatro modos, pero el consumo para la GPU varia (imagen 13). Para el caso práctico propuesto, el mejor modo de ejecución es Max-P Core-All ya que ofrece menor consumo de vatios que Max-N, 4,5 en total y un tiempo de ejecución aproximado a 12 fps. Esta ejecución, tendría un rendimiento similar al del ojo humano, que puede procesar entre 10 y 12 fps.

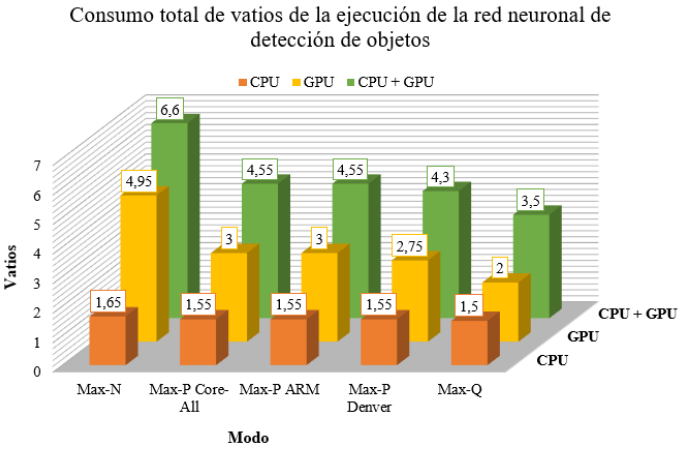


Imagen 13. Gráfica con el consumo en vatios de la CPU, GPU y la suma de estos para los diferentes modos de ejecución con ayuda de la herramienta nvpmodel.

7 CONCLUSIONES

Para el presente trabajo se marcaron los siguientes objetivos. En primer lugar, se pretendía aplicar tres tipos de redes convolucionales orientadas a la placa Jetson TX2. Nuestros resultados sugieren que las tres redes neuronales se han aplicado de forma satisfactoria, aunque se observaron diferentes eficiencias.

La red neuronal de segmentación obtiene un bajo rendimiento (0,8 fps) (imagen 11) y, por lo tanto, no es eficiente en tiempo real. La red de clasificación obtiene un incremento del rendimiento de ~25x más rápida que la red de segmentación (imagen 11) (20 fps y 18 fps), sin embargo, no localiza objetos en el espacio. Por otro lado, la red de detección de objetos tiene un rendimiento de 9fps (imagen 11) y localiza objetos en el espacio. A su vez, se obtuvieron medidas de rendimiento de todas las redes neuronales y se midió el consumo de la red de detección de objetos. Se comprobó que utilizando el modo de ejecución Max-N se puede obtener una mejora en el rendimiento en ~1,4x (13 fps) (imagen 12) pero es el modo que requiere más consumo energético (6,6 vatios) (imagen 13).

En conclusión, las tres redes se pueden implementar para diferentes usos en el ámbito de la conducción. Aun

así, cada una presenta características particulares que las hacen ser más apropiadas para según que circunstancia. En nuestro caso, teniendo en cuenta el caso práctico la detección objetos es la más apropiada.

Por otro lado, los experimentos realizados confirman la importancia de tener en cuenta las características hardware del sistema embebido que se va a utilizar. Puesto que, los resultados se vuelven más críticos cuando el desarrollo se realiza bajo una plataforma de bajo consumo.

Como trabajo futuro se desearía implementar un prototipo de coche con la Jetson TX2 y probar de implementar la red neuronal de detección de objetos en el coche y observar cómo se mueve en base a la salida de la red neuronal. También se podría mejorar el *dataset* de la red de detección de objetos aplicando más técnicas de *data augmentation* para considerar más casos como el triángulo invertido, más colores, diferentes posiciones angulares del triángulo, etc. De esta forma el *dataset* aumentaría su fiabilidad.

AGRADECIMIENTOS

Agradecer a mi tutor Juan Carlos Moure por su ayuda incondicional para orientarme, por sus recomendaciones y por mostrar interés hacia el proyecto. Agradecer a mis amigos, a mi pareja y familia por su apoyo en las etapas más difíciles de este proyecto.

BIBLIOGRAFÍA

[1] Ibañez-Guzmán, J., Laugier, C., Yoder, J. and Thrun, S. (2012). Autonomous Driving: Context and State-of-the-Art. *Handbook of Intelligent Vehicles*, pp.1271-1310.
[2] En 2017, 1.200 fallecidos. Available at: http://revista.dgt.es/es/noticias/nacional/2018/01ENERO/0103-Presentacion-balance-accidentes-2017.shtml#.Wx-kFyB9jIU. (Accessed: 8th June 2018)
[3] Falcini, F., Lami, G. and Costanza, A. (2017). Deep Learning in Automotive Software. IEEE Software, 34(3), pp.56-63.
[4] Frro.utn.edu.ar. (2018). Redes Neuronales: Conceptos Básicos y Aplicaciones. [online] Available at: https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf (Accessed 6 Mar. 2018)
[5] MIT 6. S094: Deep Learning for Self-Driving Cars. (2018). MIT 6. S094: Deep Learning for Self-Driving Cars. [online] Available at: https://selfdrivingcars.mit.edu/ (Accessed 8 Mar. 2018)
[6] Embedded Systems Developer Kits, Modules, & SDKs | NVIDIA Jetson. Available at: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/. (Accessed: 12th June 2018)
[7] The Machine Learning Algorithms Used in Self-Driving Cars. Available at: https://www.kdnuggets.com/2017/06/machine-learning-algorithms-used-self-driving-cars.html. (Accessed: 24th June 2018)
[8] Google. Waymo. Available at: https://waymo.com/. (Accessed: 12th June 2018)
[9] Características y fases del modelo incremental | OBS Business School. Available at: https://www.obs-edu.com/es/blog-project-management/metodologias-agiles/caracteristicas-y-fases-del-modelo-incremental. (Accessed: 12th June 2018)
[10] Aprendizaje profundo: Tres cosas que es necesario saber - MATLAB & Simulink. Available at: https://es.mathworks.com/discovery/deep-learning.html. (Accessed: 23rd June 2018)
[11] CS231n Convolutional Neural Networks for Visual Recognition. Available at: http://cs231n.github.io/convolutional-networks/. (Accessed: 12th June 2018)
[12] Redes neuronales convolucionales con TensorFlow. Available at: https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/. (Accessed: 23rd June 2018)
[13] Red neuronal Convolucional CNN - Diego Calvo. Available at: http://www.diegocalvo.es/red-neuronal-convolucional-cnn/. (Accessed: 23rd June 2018)
[14] García Fernando, G. A. Deep Learning en segmentación de imagen médica. (2017). basada en técnicas de aprendizaje profundo.
[15] Durán Suárez, J. Redes neuronales convolucionales en R: Reconocimiento de caracteres escritos a mano. (2017).
[16] Image Segmentation Using DIGITS 5. Available at: https://devblogs.nvidia.com/imagesegmentation-using-digits-5/. (Accessed: 21st April 2018)
[17] Deep Learning for Object Detection with DIGITS. Available at: https://devblogs.nvidia.com/deep-learning-object-detection-digits/. (Accessed: 12th June 2018)
[18] Aquino, N. M. R., Gutoski, M., Hattori, L. T. & Lopes, H. S. The Effect of Data Augmentation on the Performance of Convolutional Neural Networks. in Brazilian Society of Computational Intelligence (2017).
[19] DetectNet: Deep Neural Network for Object Detection in DIGITS. Available at: https://devblogs.nvidia.com/detectnet-deep-neural-network-object-detection-digits/. (Accessed: 12th June 2018)
[20] Cluster Analysis | NVIDIA Developer. Available at: https://developer.nvidia.com/discover/cluster-analysis. (Accessed: 12th June 2018)
[21] Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. Understanding deep learning requires rethinking generalization. (2016).
[22] NVPModel – NVIDIA Jetson TX2 Development Kit | JetsonHacks. Available at: https://www.jetsonhacks.com/2017/03/25/nvpmodel-nvidia-jetson-tx2-development-kit/. (Accessed: 12th June 2018)
[23] Luke Yeager, Julie Bernauer, Allison Gray, and Michael Houston. Digits: the deep learning gpu training system. In ICML 2015 AutoML Workshop, 2015.
[24] Get Started with Eager Execution | TensorFlow. Available at: https://www.tensorflow.org/get_started/eager. (Accessed: 23rd June 2018)
[25] PyTorch | About. Available at: https://pytorch.org/about/. (Accessed: 23rd June 2018)
[26] NVIDIA cuDNN | NVIDIA Developer. Available at: https://developer.nvidia.com/cudnn. (Accessed: 23rd June 2018)
[27] Jia, Y. et al. Caffe: Convolutional Architecture for Fast Feature Embedding. (2014).
[28] NVIDIA TensorRT | NVIDIA Developer. Available at: https://developer.nvidia.com/tensorrt. (Accessed: 10 Mar. June 2018)
[29] Ros, G., Sellart, L., Materzynska, J., Vazquez, D. and Lopez, A. (2016). The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
[30] Szegedy, C. et al. GoogleNet. J. Mach. Learn. Res. 45, 1–9 (2015).
[31] Hennessy, J. L. & Patterson, D. A. Computer Architecture A Quantitative Approach 4th Edition. Microelectronics J. 28, 704 (2006).

APÉNDICE

A1. ESPECIFICACIONES DE LA PLACA JETSON TX2

Tabla 4. Especificaciones técnicas de la placa Jetson TX2.

	Jetson TX2
GPU	NVIDIA Pascal™, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 +
	Quad ARM® A57/2 MB L2
Memory	8 GB 128 bit LPDDR4
	59.7 GB/s

Imagen 14. Placa Jetson TX2 development Kit.

A2. GRÁFICAS COMPLEMENTARIAS DEL RESULTADO DE LOS ENTRENAMIENTOS

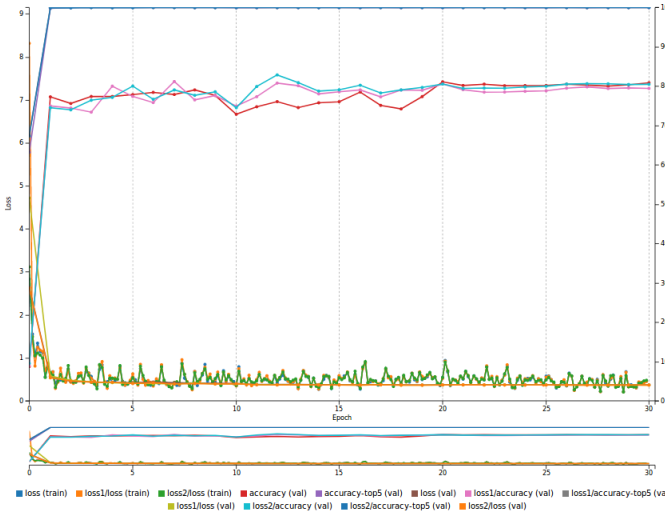


Imagen 15 Resultados de los entrenamientos de la de red clasificiación con 10 clases

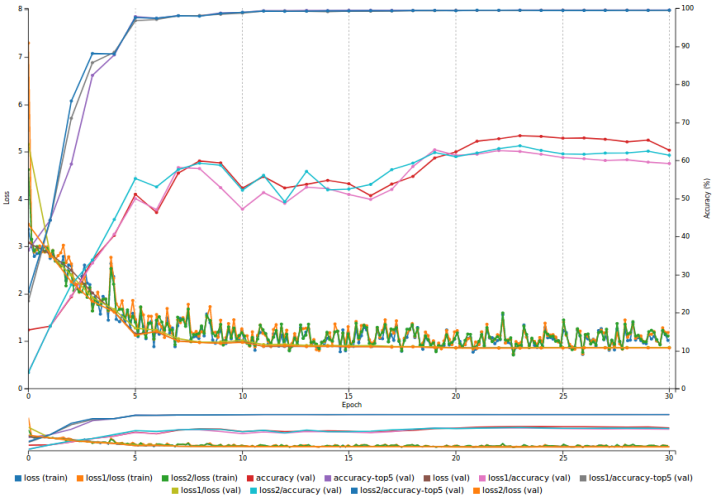


Imagen 16. Resultados de los entrenamientos de la de red clasificación con 19 clases

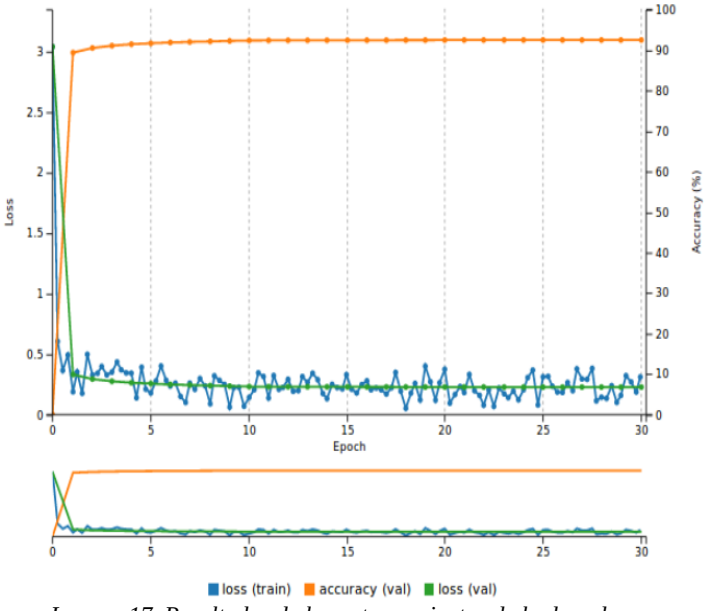


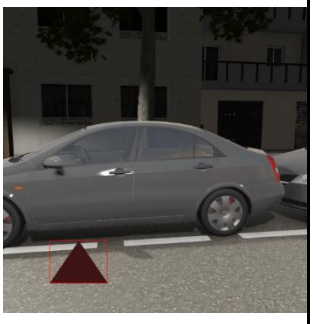
Imagen 17. Resultados de los entrenamientos de la de red segmentación

A3. RESULTADO DE HACER INFERENCIA CON LA RED DE CLASIFICACIÓN.

Imagen 18. Resultado de hacer inferència con el modelo de clasificación de 10 clases.

A4. Resultado de realizar inferencia con la red neuronal de detección de objetos

Tabla 5. Resultado de realizar inferencia sobre una imagen en la GPU 1080 y la GPU Tegra X2.

GTX 1080	
Imagen de entrada	Inferencia de la imagen
	
Tegra X2	
Imagen de entrada	Inferencia de la imagen
	

A5. Imágenes y tablas complementarias de la red neuronal de detección de objetos.

Tabla 6. Kernels que consumen más porcentaje de tiempo en la ejecución de la red de detección de objetos.

#Kernel	Nombre	Tiempo (%)
Kernel_0	trtwell_fp16x2_hcudnn_winograd_fp16x2_128x128_ldg1_ldg4_relu_tile148m_nt	18,95
Kernel_1	trt_maxwell_fp16x2_hcudnn_fp16x2_128x64_relu_medium_nn_v1	18,16
Kernel_2	trt_maxwell_fp16x2_hcudnn_fp16x2_128x64_relu_interior_nn_v1	9,77

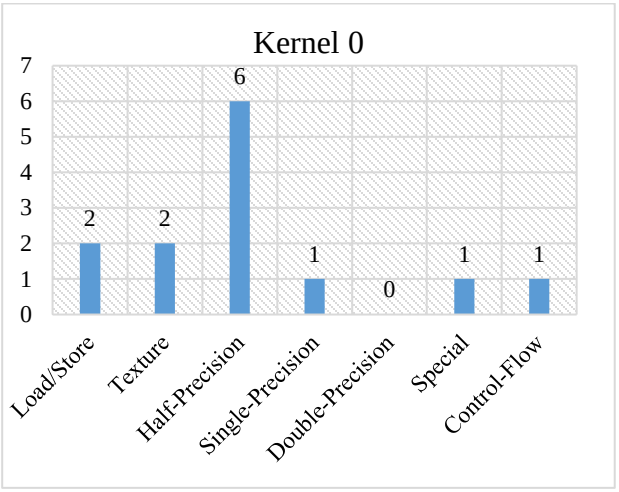


Imagen 19. Gráfica que muestra la utilización de las unidades funcionales del kernel 0

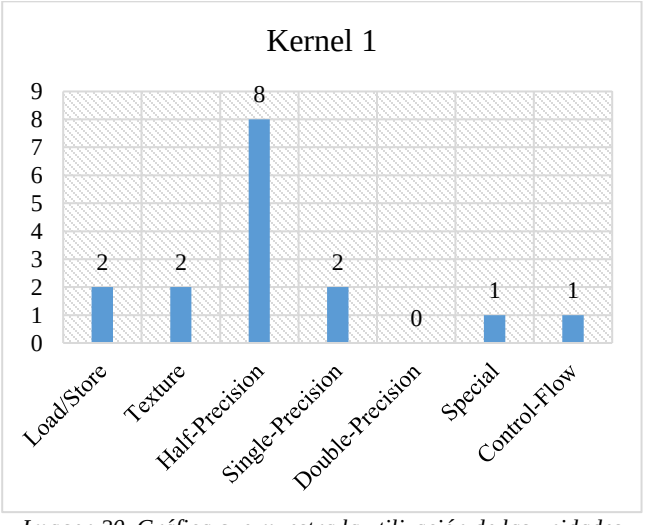


Imagen 20. Gráfica que muestra la utilización de las unidades funcionales del kernel 1

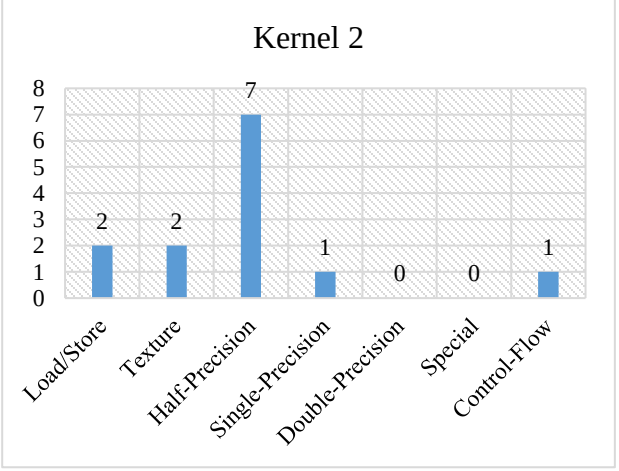


Imagen 21. Gráfica que muestra la utilización de las unidades funcionales del kernel 0

Tabla 7. Tabla con las características de la placa Jetson TX2 cuando no existe ningún algoritmo de deep learning en ejecución y cuando se ejecuta la red de detección de objetos.

Métricas	**No hay algoritmo en ejecución**	**Red de detección de objetos en ejecución**
RAM	3166/7851MB	4242/7851MB
CPU Cores	4%@2002	13%@2036
	0%@2035	0%@2034
	0%@2035	0%@2034
	5%@2006	19%@2034
	2%@2007	15%@2036
	7%@2008	16%@2034
EMC	EMC 1%@1866	EMC 39%@1866
APE	150	150
GR3D	GR3D **0%**@1300	GR3D **99%**@1300

Tabla 8. Significado de los parámetros de la tabla 11.

RAM	current / total MB
CPU	[core1%,core2%,core3%,…]@MHz
EMC	External Memory Controller, bus%@MHz
APE	audio/video processor
GR3D	GPU, processor%@MHz