UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Evaluación de la plataforma Nvidia Jetson Nano para aplicaciones de visión artificial

Autor: Natalia Jiménez Varela
Tutor: Eduardo Cermeño Mediavilla
Ponente: Juan Alberto Sigüenza

junio 2021

Natalia Jiménez Varela
Evaluación de la plataforma Nvidia Jetson Nano para aplicaciones de visión artificial

Natalia Jiménez Varela
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres

Donde haya un árbol que plantar, plántalo.
Allá donde haya un error que enmendar, enmienda.
Donde haya un esfuerzo que todos esquivan, hazlo tú.

Gabriela Mistral

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres por su apoyo incondicional durante los años que he estado estudiando, ya que sin ellos no sé si habría llegado a estar escribiendo estas líneas.

En segundo lugar, me gustaría dar las gracias a todos mis compañeros de la universidad. En especial, a mi compañero y amigo, Gonzalo Fuentes, por cada práctica que hemos hecho juntos, que por mucha dificultad que hubiese podido tener, siempre hemos sacado el lado positivo y hemos logrado superar todas nuestras expectativas.

También quiero destacar el apoyo de mi tutor Eduardo Cermeño, por la ayuda que he recibido elaborando este trabajo.

Resumen

Nvidia ofrece una plataforma de bajo coste y de bajo consumo, denominada Nvidia Jetson Nano. En este trabajo se han analizado las distintas posibilidades de este dispositivo, con el objetivo de utilizarlo como plataforma general de desarrollo de aplicaciones de visión aritificial, o Computer Vision. Para ello, se han analizado algunos proyectos ya existentes en los que se ha utilizado esta plataforma y se han seleccionado una serie de herramientas y librerías que permitan dar un alto grado de flexibilidad a los desarrolladores a la hora de diseñar e implementar sus soluciones, poniendo especial atención a aquellas librerías dedicadas al tratamiento de imágenes.

Por otro lado, se han realizado pruebas de rendimiento, cuyo objetivo ha sido conocer las limitaciones que ofrece la Jetson Nano, al tratar con datos tan exigentes como son las imágenes y los vídeos. Los resultados han sido, por lo general, positivos, ya que se han podido instalar todas las herramientas que se han considerado necesarias. Sin embargo, ha requerido de cierto esfuerzo, al encontrar numerosos conflictos entre los elementos a instalar y querer instalar la versión más adecuada para esta plataforma.

Los experimentos que se han realizado, han demostrado que la Nvidia Jetson Nano permite trabajar, tanto como con imágenes, como con vídeos, de forma fluida, aunque con ciertas limitaciones en el uso de redes neuronales profundad de uso habitual. En muchas ocasiones, se ha utilizado además la aceleración hardware de la que dispone el dispositivo, reduciendo considerablemente el tiempo de ejecución de algunas aplicaciones, gracias al uso de CUDA y, también, de TensorRT, tecnologías específicas de Nvidia; obteniendo así un mejor rendimiento.

Palabras clave

NVIDIA Jetson Nano, Computer Vision, deep learning, CPU, GPU, Python, C++, OpenCV, Qt, FPS

Abstract

Nvidia offers a low-cost, low-power platform called the Nvidia Jetson Nano. In this work, the different possibilities of this device have been analyzed, with the aim of using it as a general platform for the development of artificial vision applications, or Computer Vision. For this, some existing projects, in which this platform has been used, have been analyzed and a series of tools and libraries have been selected that allow developers to give a high degree of flexibility when designing and implementing their solutions, putting special attention to those libraries dedicated to the treatment of images.

On the other hand, performance tests have been carried out. The goal has been to know the limitations offered by the Jetson Nano, when dealing with demanding data, as images and videos. The results have been, in general, positive, since all the tools that have been considered necessary have been installed. However, it has required some effort, finding numerous conflicts between the elements to install and wanting to install the most suitable version for this platform.

The experiments that have been carried out have shown that the Nvidia Jetson Nano allows you to work, with images and with videos, in a fluid way, although with certain limitations in the use of commonly used deep neural networks. On many occasions, the hardware acceleration available to the device has also been used, considerably reducing the execution time of some applications, thanks to the use of CUDA and, also, TensorRT, specific technologies from Nvidia; obtaining a better performance.

Keywords

NVIDIA Jetson Nano, Computer Vision, deep learning, CPU, GPU, Python, C++, OpenCV, Qt, FPS

Índice

Listas

Lista de algoritmos

Lista de códigos

Lista de cuadros

Lista de ecuaciones

Lista de figuras

Lista de tablas

Lista de cuadros

1

Introducción

1.1. Motivación y objetivos

En los últimos años ha crecido el desarrollo de aplicaciones relativas a **Computer Vision**, o en español, visión artificial, ya que es utilizado en situaciones cotidianas, [1] como es el caso de un parking, en el cual se emplea para el reconocimiento de matrículas. Sin embargo, no es el único campo en el que se emplea este tipo de tecnología, ya que, por ejemplo, en la medicina, ha mejorado el diagnóstico de enfermedades como el cáncer. También se ha utilizado en aplicaciones de videovigilancia, con el fin de reducir la incidencia delictiva.

Una de las aplicaciones de Computer Vision más común es [2] el análisis de una imagen para obtener información, ya sea para detectar objetos, marcas o, clasificar dicha imagen.

Los ingresos del mercado de inteligencia artificial en todo el mundo va en aumento año tras año, como se puede observar en la figura 1.1.

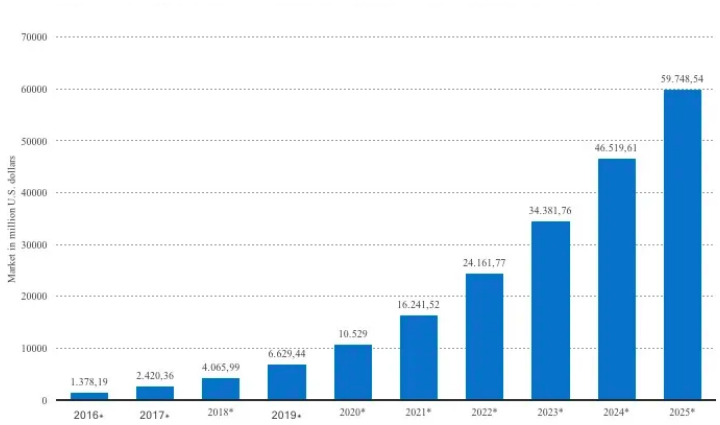

Figura 1.1: Ingresos del mercado de la IA entre 2016 y 2025 [3]

Dividiendo la IA en distintas ramas, en función de los objetivos, se puede observar en la figura 1.2 que la parte correspondiente al análisis y clasificación de imágenes es la rama que más ingresos genera.

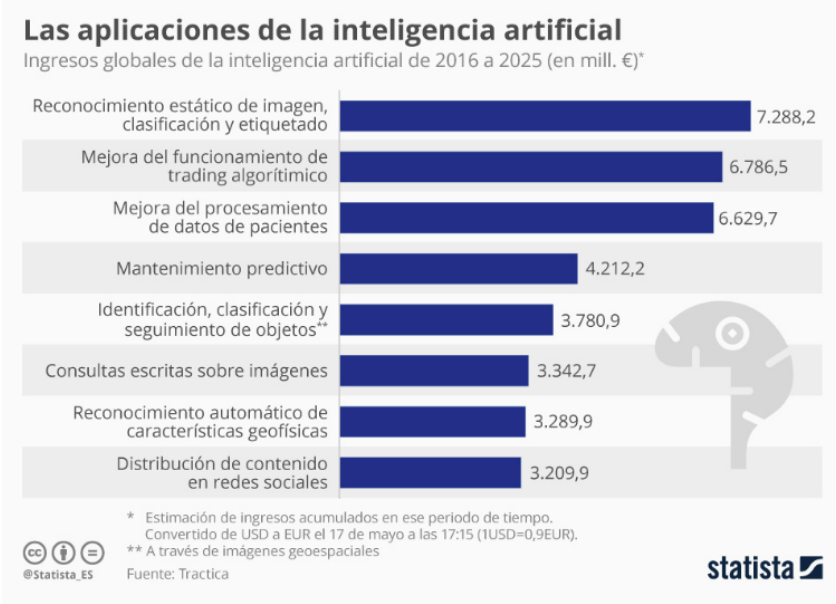

Figura 1.2: Ingresos del mercado de la IA separando por ramas entre 2016 y 2025 [4]

Esto se debe a grandes avances, como ha sido la utilización de **redes neuronales profundas**, lo que implica el uso de deep learning y de la **aceleración hardware**, haciendo que se reduzca considerablemente el tiempo de ejecución de algunas aplicaciones.

NVIDIA es una compañía especializada en el desarrollo de GPUs (unidades de procesamiento gráfico). Además NVIDIA ha creado plataformas de bajo coste con el objetivo de que los desarrolladores se puedan iniciar en el campo de Computer Vision, como es el caso de la plataforma NVIDIA Jetson Nano.

La **NVIDIA Jetson Nano** es una [5] plataforma con la que comenzar a interesarse por la IA y por la robótica. [6] Es una computadora pequeña, a través de la cual se pueden ejecutar distintas redes neuronales en paralelo, con el objetivo de clasificar imágenes o detectar objetos. Su bajo coste la hace muy atractiva, ya que no supera los 100 dólares. Además, la Jetson Nano cuenta con GPU, lo que permite la ejecución de aplicaciones con dicha aceleración, provocando que sea de gran interés el rendimiento que pueda tener.

La plataforma NVIDIA Jetson Nano parece interesante para aplicaciones de visión artificial, y la idea de este trabajo es experimentar con ella para evaluar sus posibilidades como plataforma general de Computer Vision.

Por tanto, los objetivos son los siguientes:

O-1.– Saber si es fácil utilizar esta plataforma, teniendo en cuenta el estado del arte que actualmente hay disponible.

O-2.– Incorporar librerías y frameworks de Computer Vision, así como comprobar su correcta instalación y funcionamiento, haciendo un análisis de las mismas.

O-3.– Incorporar otras librerías y frameworks para comprobar su compatibilidad con tecnologías de Computer Vision.

O-4.– De forma más global, y relacionándolo con los objetivos anteriores, saber si se puede utilizar la NVIDIA Jetson Nano para crear soluciones de Computer Vision.

O-5.– A raíz del anterior objetivo, en caso de que la respuesta a dicha pregunta sea afirmativa, saber si la NVIDIA Jetson Nano es lo suficientemente potente, midiendo su rendimiento.

1.2. Estructura de la memoria

En este documento se pueden distinguir los siguientes apartados:

- **Estado del arte:** Apartado en el que se analizarán los trabajos desarrollados en la plataforma NVIDIA Jetson Nano.
- **Diseño y desarrollo:** Teniendo en cuenta el punto anterior, correspondiente al estado del arte, se analizarán las posibilidades y dificultades para desplegar diferentes herramientas de desarrollo de aplicaciones de Computer Vision.
- **Pruebas y resultados:** En esta sección se analizarán los resultados obtenidos de una serie de pruebas de rendimiento, para valorar el mismo.
- **Conclusiones:** En esta parte del documento se recogerán las conclusiones del trabajo realizado.
- **Bibliografía:** Donde se incluirán las referencias que hayan sido utilizadas, así como el glosario, los acrónimos y sus respectivas definiciones.
- **Apéndices:** Apartado en el que será incluida información adicional, con el objetivo de que el lector pueda consultarlo.

2

Estado del arte

La NVIDIA Jetson Nano es ya una herramienta muy popular para el desarrollo de proyectos de Computer Vision [7]. A continuación, se analizan algunos proyectos que utilizan este dispositivo.

2.1. NVIDIA JetBot

NVIDIA JetBot [8] es un proyecto de robótica de código abierto. Se trata de un robot de deep learning fácil de configurar y de utilizar y cuyo conjunto no supera los 250 dólares, incluyendo la Jetson Nano y una serie de componentes adicionales, como es el chasis, las ruedas, etc; aunque es compatible con múltiples accesorios. Todo este conjunto de componentes se puede observar en la figura 2.1.

Figura 2.1: Prototipo NVIDIA JetBot [8]

También forman parte de este proyecto, [8] una serie de tutoriales sencillos, tratándose de tutoriales de inteligencia artificial sobre evitar colisiones o seguir objetos. Este robot admite librerías de inteligencia artificial, como es TensorFlow o PyTorch, haciendo uso también de Python, C++ y CMake.

Tiene una serie de tutoriales básicos como los siguientes [8]:

- Movimientos básicos: Posibilidades de moverse hacia adelante, hacia detrás, a ambos lados y pararse.
- Teleoperación: Posibilidad de mover el robot y ver todo lo que captura la cámara del mismo.
- Evitar colisiones: Basándose en un conjunto de datos de clasificación de imágenes, detectará dos escenarios.

Uno de esos escenarios es 'free', en el cual podrá seguir su camino, y 'blocked', en el cual se detendrá por posible colisión.

- Seguir un camino o carretera: Al igual que el ejemplo anterior, se basa en un conjunto de datos de clasificación de imágenes, gracias al cual se detectan coordenadas y seguir el camino correcto.
- Seguir un objeto: Se trata de un ejemplo de uso basado en un modelo pre-entrenado, capaz de detectar objetos básicos, como pueden ser una persona o un perro; y a su vez, evitará las colisiones correspondientes.

Este proyecto es destacable ya que usa un gran número de accesorios y se comprueba su compatibilidad con los mismos. Además, como se ha mencionado, puede realizar acciones básicas de movimiento a modo de iniciación a la parte de robótica, además de servir como herramienta para Computer Vision. Una de las ventajas de este pequeño robot es su precio económico, ya que, como se ha mencionado anteriormente, [8] la compra de todos los componentes no supera los 250 dólares.

2.2. Estimación de pose humana en tiempo real

Es un proyecto en el que se estima la pose humana con aceleración de NVIDIA TensorRT. [9] Cuenta con modelos pre-entrenados para la estimación de diferentes poses humanas en tiempo real. Esto se consigue detectando puntos claves, como pueden ser los hombros, codos, manos, rodillas, pies, etc; aunque también cuenta con una parte de entrenamiento para detectar puntos claves que no formen parte de la pose.

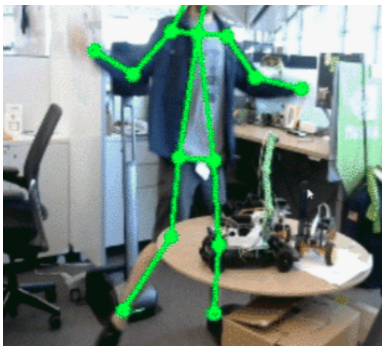

Figura 2.2: Proyecto de estimación de pose humana en tiempo real [9]

Está desarrollado fundamentalmente en Python y C++, haciendo uso de librerías como PyTorch. Uno de los modelos llega a alcanzar los 22 FPS [9].

Este proyecto es destacable, ya que se puede ampliar para distintas partes del cuerpo humano en concreto. El mismo creador de este, ha desarrollado un proyecto similar para estimar la posición de la mano [10]. Esto es destacable porque, a parte de la funcionalidad básica, una de las aplicaciones reseñables de dicho proyecto es la actuación de la mano a modo cursor en el propio dispositivo, por ejemplo utilizándolo para realizar compras en internet; o también dibujar.

2.3. Reconocimiento de matrícula en tiempo real

Se trata de un proyecto en el que se puede detectar la placa de una matrícula y reconocer la misma, [11] acelerando dicho proceso con TensorRT. El conjunto de datos en el que se ha basado este proyecto, fue recopilado en Vietnam; el cual se debe entrenar con la GPU en Google Colab o en el dispositivo.

Figura 2.3: Proyecto de reconocimiento de matrículas en tiempo real [11]

Está desarrollado completamente en Python, en el que se utilizan librerías como TensorFlow, Keras o Scipy.

Llega a alcanzar los 40 FPS [11], algo muy destacable, ya que supera la media del resto de proyectos y se podría observar la aceleración hardware de la GPU. Puede ser útil para el control de matrículas en cámaras de seguridad, ya que, por un bajo coste (alrededor de 100 euros), obtiene un alto rendimiento.

2.4. Identificador de recipiente de comida

Es un proyecto con el que se identifican recipientes o frascos de comida y [12] hace que la Jetson Nano dicte en alto el nombre del producto en cuestión, con el objetivo de que las personas ciegas o que tengan alguna discapacidad visual puedan identificar correctamente la lata o frasco que sea escogido. Se acelera dicho proceso con el uso de TensorRT.

Se deberá entrenar el modelo de clasificación de imágenes con los datos recopilados [12]. Está desarrollado completamente en Python. Cabe destacar la utilización de la librería pyttsx3, con el fin de que Jetson Nano lea en voz alta el nombre correspondiente.

Este proyecto es bastante interesante, ya que es un proyecto en el cual los resultados no se muestran por pantalla mostrando la probabilidad, sino que hacen uso de una librería para ello, como se ha mencionado anteriormente, y es por una buena causa.

2.5. Cámara detectora de mascarillas

Es un proyecto de cámara inteligente que mide el uso de la mascarilla de la gente en tiempo real. [13] En este, se detecta e identifica a las personas que se pueden ver mediante la cámara y, una vez que esto se produce, determina si están usando mascarilla. Carga estadísticas en la nube, guarda ciertos fragmentos del vídeo en el disco local (por ejemplo, en el caso de que haya muchas personas en la imagen y la mayoría no lleva mascarilla) y puede transmitir vídeos en directo a través del protocolo RTSP.

Figura 2.4: Cámara de reconocimiento de mascarillas [13]

Está desarrollado en su mayoría en Python, aunque también hay partes en C++. Dependiendo de la red que sea escogida, [13] puede alcanzar 35 FPS o bien, por lo contrario, tan solo 2.5 FPS.

Es destacable ya que se puede ver una comparativa de los FPS a los que puede trabajar, dependiendo de la red neuronal escogida. Es interesante saber que en función de las características de estas redes, por ejemplo el tamaño, puede beneficiar o no al tiempo de respuesta.

2.6. YolactEdge

Se trata de un proyecto de segmentación de objetos que se ejecuta en tiempo real, y su respectiva identificación [14]. Se ha optimizado el proceso con la herramienta TensorRT, teniendo especial cuidado con la precisión. Detecta todo tipo de objetos, ya sean personas, vehículos u objetos.

Ha sido desarrollado completamente en Python, haciendo uso de librerías como PyTorch, Matplotlib y OpenCV.

Se ha llegado a ejecutar a 30 FPS aproximadamente en Jetson AGX Xavier [14], un producto mucho más potente que Jetson Nano. Sería interesante contar con los resultados de ejecutarlo en la NVIDIA Jetson Nano, pero no se dispone de los mismos en la documentación del proyecto. Este proyecto, además, es reseñable, ya que puede detectar distintos tipos de objetos, no únicamente se

(a) Reconocimiento de objetos de cocina [14]

(b) Reconocimiento de objetos de coches [14]

Figura 2.5: Proyecto YolactEdge. Este proyecto puede reconocer todo tipo de objetos, como se puede observar en las figuras 2.5(a) y 2.5(b)

focaliza en objetos de un determinado tema.

A diferencia de los proyectos vistos anteriormente, en este trabajo se estudiarán las distintas posibilidades como una plataforma de aplicaciones de Computer Vision, teniendo una visión general de dicha herramienta, probando diferentes librerías relacionadas con el campo y comprobando su rendimiento.

3

Diseño y desarrollo

A continuación, se explica el diseño y el desarrollo para utilizar la NVIDIA Jetson Nano como herramienta de soporte de aplicaciones de Computer Vision y deep learning.

3.1. Diseño

En este caso, se analizaron los componentes necesarios para dicho tipo de aplicaciones.

3.1.1. Análisis de componentes

Para crear un proyecto, se analizaron los componentes que debían formar parte del entorno de desarrollo de una aplicación de Computer Vision y deep learning en una herramienta como es la NVIDIA Jetson Nano.

- Editores de texto
- Lenguajes de programación
- Compiladores
- Herramientas de código
- Herramientas que faciliten el proceso de instalación
- Herramientas NVIDIA
- Librerías: Para crear una aplicación completa, se instalan herramientas adecuadas para el tipo de aplicación que se desarrolla en el dispositivo, atendiendo a los siguientes tipos.
    - Cálculo matemático: Con el objetivo del manejo de imágenes, ya que facilitan ciertas operaciones numéricas.
    - Aprendizaje automático: Se necesitan para analizar imágenes e identificar ciertos modelos, de tal manera que se realice una predicción de lo que se está obteniendo.
    - Deep learning: Computer Vision es un área en el que se está aumentando el uso de deep learning, y este tipo de librerías se ejecutan en las GPUs, como de las que dispone la NVIDIA Jetson Nano. Se trata de un tipo de aprendizaje automático, pero mucho más complejo, funcionando de manera semejante al cerebro humano.

  - Computer Vision y procesado de imágenes: Toda aplicación de Computer Vision debe contar con librerías destinadas a ese mismo propósito, así como librerías que faciliten el procesado y visionado de imágenes.
  - Front-end: De tal manera que facilite la codificación que se encarga de la visión del usuario de la aplicación que se haya desarrollado.
- Frameworks
- Gestores de bases de datos
- Servidor web

3.1.2. Elección de componentes

Dentro de los componentes seleccionados, fue necesario elegir las herramientas que más se adecuen a los proyectos que se pueden realizar en la plataforma Jetson Nano.

Editores de texto

Como editores de texto disponibles con el JetPack de NVIDIA están vi y gedit. Sin embargo, usando este último en dicha plataforma, tenía un tiempo de respuesta demasiado lento. Vi se trata de un editor de texto por terminal con bastante potencia. Un punto a favor de gedit es que facilita la codificación de manera visual, y este es el motivo por el que se decidió instalar nano, ya que combina la programación por terminal con una interfaz más intuitiva.

Lenguajes de programación

Como se ha podido observar en el apartado del estado del arte, la mayoría de los proyectos de Computer Vision y deep learning se han desarrollado en Python y en C++. Además, se nota un claro descenso en el interés por C++, y, por lo contrario, un aumento en Python; aunque la media de interés es bastante pareja entre ambos lenguajes, como se puede observar en la siguiente gráfica, obtenida de Google Trends.

Python es un lenguaje con varias versiones. Se decidió instalar la versión de Python 3, debido a que el fin de vida de Python 2 se marcó en enero del año 2020 [16].

Se ha decidido instalar Cython 3 [17] debido a que se trata de un lenguaje de programación, con el que se puede acelerar la ejecución de código, generando código C. En este caso, se puede acelerar bastante el proceso de recorrer una imagen píxel a píxel.

Compiladores

Se decidió instalar los compiladores necesarios para la compilación de ciertas herramientas, así como para los lenguajes de programación elegidos. En este caso, el primer elegido fue GFortran, tam-

Figura 3.1: Comparación de búsquedas de Python y C++ [15]

bién denominado GNU Fortran. Es un compilador que forma parte de GCC (GNU Compiler Colleciton). Se instala debido a que es requerido por otras librerías como Keras o Scipy [18], de las que se hablarán en el apartado 3.1.2. Otro compilador que se ha decidido instalar es Protocol Buffer, cuyo objetivo es serializar datos estructurados y servirá para acelerar ciertos procesos. Por último, se decidió instalar el metapaquete build-essential [19], para compilar cualquier otro paquete software procedente de un fichero, es decir, de varias librerías que se instalaron en la NVIDIA Jetson Nano, como es el caso de OpenCV. Este paquete también es necesario para compilar código fuente de C++, uno de los lenguajes de programación elegidos.

Herramientas de código

Se pueden utilizar ciertas herramientas que nos faciliten el desarrollo del proyecto. Una plataforma muy popular es GitHub [20], ya que facilita compartir el código con otros usuarios, así como su almacenamiento. Hay más de 65 millones de desarrolladores que lo utilizan, y, con ello, más de 200 millones de repositorios. También se decidió instalar CMake [21], tratándose de una herramienta destinada a la generación o automatización de código. El caso más reseñable es la compilación de OpenCv, librería de la que se hablará en la sección 3.1.2. Por último, con el objetivo de facilitar el manejo de Python 3, se instalaron dos herramientas: virtualenv y virtualenvwrapper [22].

Herramientas que faciliten el proceso de instalación

Con el fin de facilitar procesos de instalación, se ha decidido instalar pip, para gestionar paquetes escritos en Python.

Herramientas NVIDIA

Existen unas herramientas de NVIDIA que pueden facilitar el trabajo y el desarrollo de proyectos.

Jetson-Stats [23] es un paquete que sirve para monitorear y controlar cualquier plataforma NVIDIA Jetson.

- **jtop:** Es una herramienta con la que se puede monitorear y controlar en tiempo real el estado de cualquier sistema NVIDIA Jetson. Se ejecuta en una terminal mediante el comando jtop. La interfaz es como se muestra en la figura 3.2. Como se puede observar en la figura 3.2, en la parte inferior dispone de 6 páginas diferentes. La primera página es la que muestra toda la información sobre su estado: estado de la CPU, de la memoria, de la GPU, del disco, del ventilador, así como jetson_clocks y nvpmodel; coincidiendo así con la interfaz de inicio, como se muestra en la figura 3.2. El segundo apartado es GPU, donde se muestra todo su historial en tiempo real. El punto número 3 es la CPU, donde también se muestra el historial correspondiente en tiempo real. La cuarta página es MEM (memoria). Donde se muestra un gráfico de la memoria en tiempo real, así como un monitor de intercambio. Con la opción c se puede borrar la memoria caché, con la opción s se puede activar o desactivar intercambio y con los símbolos + y - se puede aumentar o disminuir el tamaño de dicho intercambio. Esta es la pantalla correspondiente a la figura 3.3. Por último, se encuentra el apartado INFO (información). En esta pantalla se muestra toda la información correspondiente al jetpack, hardware, librerías, y sus correspondientes versiones, etc. Es bastante similar a la salida correspondiente a las variables Jetson. El quinto apartado es CTRL (control), donde se puede activar o desactivar jetson_clocks, aumentar o disminuir nvpmodel, y aumentar o reducir la velocidad de los ventiladores. Es lo correspondiente a la figura 3.4.

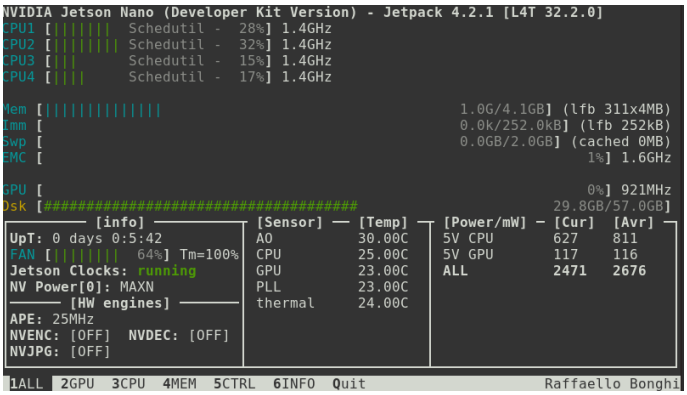

Figura 3.2: Pantalla principal del comando jtop

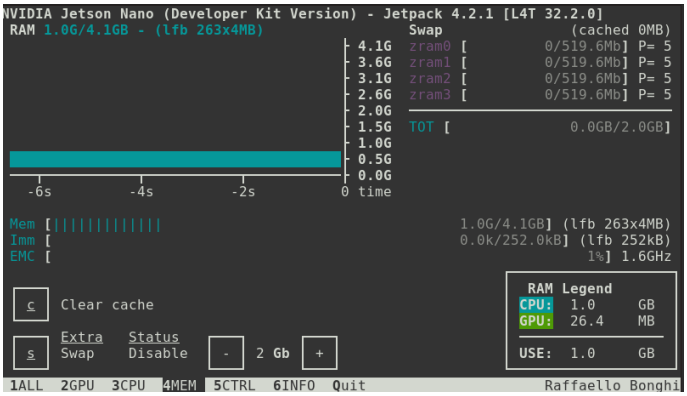

Figura 3.3: Pantalla correspondiente a memoria del comando jtop

- **jetson_config:** Con esta herramienta se pueden configurar distintas opciones, como se explica en la tabla 3.1.
- **jetson_release:** Se trata de una herramienta a través de la cual se puede obtener la información general del sistema, como son las versiones de Jetpack, librerías, etc. La salida es la correspondiente a la figura 3.5.

Figura 3.4: Pantalla correspondiente a control del comando jtop

Apartado	Descripción
Health	Opción con la que se puede comprobar el estado de jetson-stats.
Desktop	Permite o no el inicio desde el escritorio.
Wifi	Opción con la que se puede mejorar el rendimiento inalámbrico.
Update	Actualizar esta herramienta de configuración.
About	Se obtiene información sobre la propia herramienta.

Tabla 3.1: Tabla sobre la funcionalidad de Jetson Config

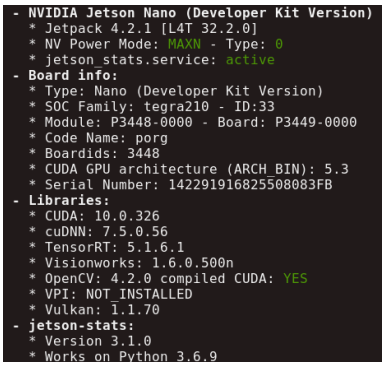

Figura 3.5: Salida del comando jetson_release

- **jetson_swap:** Es un administrador cuyo fin es encender y/o apagar un archivo de intercambio en la Jetson Nano. Cuenta con distintas opciones, tal y como se detalla en su opción de ayuda. Esto se puede observar en la figura 3.6.

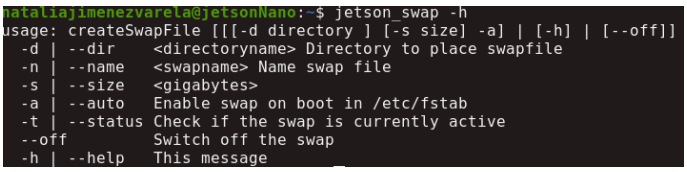

Figura 3.6: Posibilidades del comando jetson_swap

- **jetson variables:** Una vez que sea instalado el paquete jetson-stats, habrá una lista de variables de entorno para saber las versiones disponibles, como se observa en la figura 3.7.

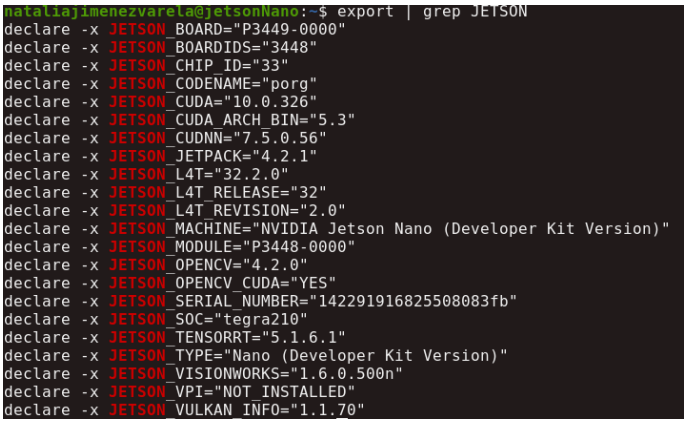

Figura 3.7: Variables relacionadas con Jetson

CUDA [24] es el modelo de programación paralela de NVIDIA. La aceleración de GPU se ha vuelto potente en los últimos años, ya que se ha producido un rápido crecimiento. El motivo del diseño de los aceleradores gráficos fue el procesamiento de imágenes, una área importante de Computer Vision, aunque con el tiempo se ha acabado utilizando para cálculos paralelos masivos. El desarrollo de esta aceleración, con el apoyo de de NVIDIA, tuvo inicio en 2010.

NVIDIA TensorRT [25] es kit de desarrollo software para inferencia de deep learning con un rendimiento alto, ya que incluye un optimizador para ello. Se basa en CUDA, descrito anteriormente, y le permite optimizar múltiples recursos.

Las aplicaciones que hacen uso de este kit, son 40 veces más rápidas que las ejecutadas con CPU durante la fase de inferencia, ya que se pueden optimizar los modelos de redes neuronales entrenados.

Librerías

Como se ha explicado anteriormente, se ha decidido instalar librerías con diversos fines.

Por un lado se encuentran las **librerías de cálculo matemático**, entre las que se han escogido NumPy y SciPy.

- **NumPy** [26] es una librería de Python, con la que se pueden manejar arrays y matrices, siendo estas de gran tamaño y multidimensionales y, además, para operar con dichas matrices, posee una serie de funciones matemáticas de alto nivel.
  Esta librería tiene una estructura de datos bastante conocida [27], n-d array, que es utilizada por otras librerías para representar datos más complejos, como pueden ser imágenes, lo que es de interés para el ámbito de Computer Vision.
- **SciPy** [28] se trata de una librería que proporciona una eficiencia superior en rutinas numéricas, así como en la integración numérica, álgebra lineal y/o estadística. Esto puede acelerar ciertos procesos de Computer Vision.

Las **librerías de aprendizaje automático** que se han escogido entre otras, son TensorFlow, Scikit-Learn y PyTorch.

- **TensorFlow** [29] es la librería de aprendizaje automático más famosa del mundo, perteneciendo a Google. Se trata de una librería de código abierto, e incorpora diferentes APIs para construir redes neuronales. Se puede ejecutar tanto en CPU, como en GPU.
- **Scikit-Learn** [30] se trata de un paquete sencillo y eficiente para el análisis de los datos, y su posterior predicción. Es un código abierto, construido sobre las librerías NumPy, SciPy y Matplotlib.
- **PyTorch** [31] es una herramienta de aprendizaje automático de código abierto. Se basa en el uso de tensores [32], una estructura de datos similar a los n-d arrays de NumPy, con la diferencia de que estos tensores pueden ejecutarse en GPU.

La **librería de deep learning** más conocida es Keras.

- **Keras** [33] es una API de deep learning en Python, que se ejecuta sobre TensorFlow. El objetivo de esta API es permitir al desarrollador una mayor rapidez.

Hay diversas **librerías dedicadas a Computer Vision y al procesado de imágenes**.

- **Matplotlib** [34] es una librería de código abierto completa destinada a crear visualizaciones (estáticas, dinámicas e interactivas) en Python. Permite mostrar matrices NumPy como imágenes y múltiples funciones más. También es una librería que ayuda al cálculo matemático, gracias a las gráficas.
- **pillow**. [35] También llamada PIL. Se trata de una librería de procesado de imágenes para Python. Está diseñada para un rápido acceso a los datos en unos píxeles básicos, teniendo una representación interna eficiente.
- **imutils** [36] es una librería que contiene una serie de funciones para hacer funciones básicas durante el procesado de imágenes, como puede ser la rotación, el cambio de tamaño o la visualización de imágenes de la librería Matplotlib.
- **Dependencias de OpenCV**. Se deben instalar todas las dependencias de OpenCV en la NVIDIA Jetson Nano, con el objetivo de compilar dicha librería con paralelismo. Entre estas dependencias se encuentra build-essential, paquete de compiladores del que se ha hablado con anterioridad. Por otro lado, se pueden instalar algunos códecs y bibliotecas de imágenes, así como bibliotecas GUI. También se instalará V4L (Video4Linux), tratándose de una API de captura y manejo de vídeo para Linux, con el objetivo de trabajar con cámaras web USB. Asimismo, se instalará una biblioteca para cámaras FireWire.
- **OpenCV** [37] (Open Source Computer Vision Library) es una librería software libre de Computer Vision y aprendizaje automático, desarrollada inicialmente por Intel. [38] Se creó en 1999 con el objetivo de facilitar una herramien-

ta común para las aplicaciones de computer vision, siendo así una librería multiplataforma. OpenCV facilita que las empresas y/o usuarios utilicen y modifiquen el código, ya que es una herramienta con licencia BSD (Berkeley Software Distribution). Es una librería robusta y eficiente.

Este producto tiene una gran cantidad de algoritmos optimizados, pudiendo así ser utilizados para múltiples funcionalidades [39] , tales como: detectar rostros y objetos, tanto en imágenes como en vídeos; cambiar de color un vídeo, extraer modelos 3D, unir imágenes para obtener una mayor resolución, reconocimiento de líneas y puntos, reconocimiento de patrones (teniendo en cuenta sus características), etc. A gran escala, es empleado para detectar intrusos en cámaras de videovigilancia, ayudar a detectar posibles accidentes en piscinas o captar las etiquetas de los productos en fábricas.

OpenCV es una librería bastante popular [37], ya que posee más de 47 mil usuarios en su comunidad y su número de descargas superan los 18 millones. Además, grandes empresas hacen uso de esta herramienta, como Intel, Google, Yahoo, IBM, etc.

Como se ha mencionado anteriormente, esta librería es multiplataforma, ya que es compatible con los sistemas operativos más utilizados [40]: Windows, Linux, Android y MacOS; y tiene varias interfaces: C++, Python, Java y MATLAB; aunque está escrita de forma nativa en C++.

En los últimos años, se han desarrollado versiones que mejoran el uso de CUDA, por lo que se instalará OpenCV con soporte CUDA.

Respecto a las **librerías de front-end**, se ha escogido React.

- **React** [41] es una librería de JavaScript, cuyo fin es crear interfaces de usuario complejas de una forma sencilla. Es una herramienta basada en componentes, y el sistema se encarga de actualizar y renderizar los componentes eficientemente cuando cambien los datos. Se ha escogido esta tecnología ya que, junto a Angular y Vue, son las herramientas más utilizadas [42]; y entre estas tres, la más buscada en Google ha sido React, como se puede observar en la figura 3.8.

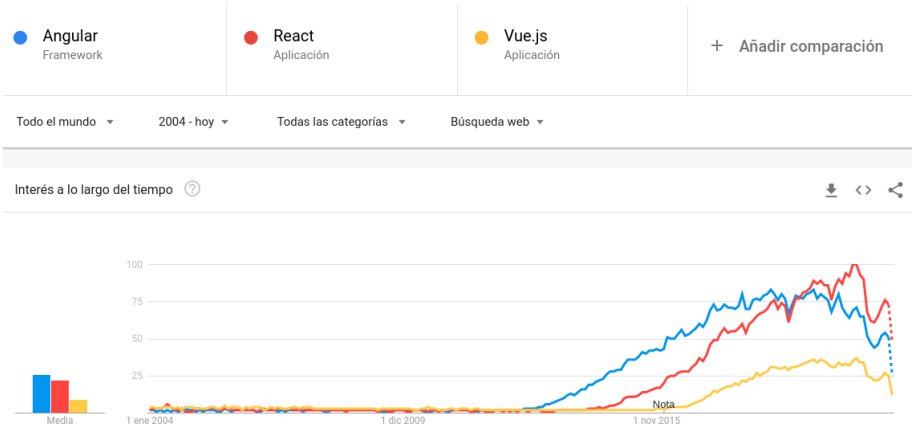

Figura 3.8: Comparación de búsquedas de Angular, React y Vue [15]

Frameworks

El objetivo principal es escoger los frameworks más adecuados para los lenguajes de programación elegidos.

- **FastAPI** [43] Es un framework web de alto rendimiento, con el objetivo de crear APIs en Python, con versiones igual o mayor a la 3.6, siendo así un framework robusto, intuitivo y fácil de usar y aprender.

- **Flask** [44] es un microframework web en Python y tiene como objetivo ser rápido y sencillo. Es una herramienta adecuada para desarrolladores de aprendizaje automático, ya que se crea fácil y rápidamente una aplicación web y una API.
- **Qt** [45] es un framework gracias al cual se pueden desarrollar aplicaciones multiplataforma. Es una herramienta multiplataforma, pudiendo utilizarse en Windows, OS X, Linux, etc. No se trata de un lenguaje de programación: es un framework escrito en C++ y, de hecho, es uno de los más populares para dicho lenguaje, siendo más adecuado para un sistema embebido. Se trata de un framework de código abierto, y las librerías que lo utilizan pueden ser compiladas con cualquier compilador compatible con C++, como es el caso de GCC.

  El desarrollo de esta plataforma tuvo como inicio 1990 [46], y la encargada de que esto se produjese, fue una empresa de Noruega, llamada Trolltech. Sin embargo, en 2008 Nokia adquirió esta empresa y, en 2012, Digia, una empresa de desarrollo finlandesa, adquirió esta herramienta de Nokia.

  Qt cuenta con Qt Creator, su propio entorno de desarrollo (IDE), y este producto también es multiplataforma. Ofrece múltiples ayudas al desarrollador, como es la finalización de código interesante, o el resaltado de sintaxis. No es necesario que se desarrolle en Qt Creator el proyecto, sino que se puede utilizar cualquier otro IDE. [47] Qt Creator fue uno de los IDEs empleados para C++ más utilizados en 2018, como se observa en la figura 3.9.

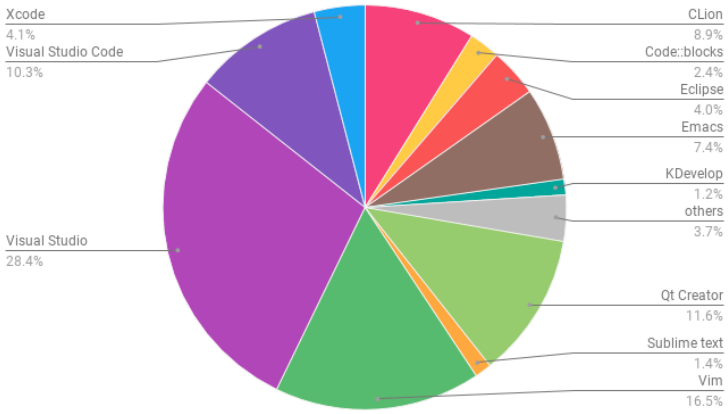

Figura 3.9: Comparación de IDEs más utilizados para C++ en 2018 [47]

- **FFMPEG** [48] es uno de los frameworks multimedia más populares, a través del cual se puede decodificar, codificar, transcodificar, etc. Es una herramienta que permite todos los formatos, y es un sistema multiplataforma, aunque inicialmente fue desarrollado en Linux. Fue lanzado en el año 2000.

  En el caso del comando ffmpeg, [49] es un conversor de vídeo y audio rápido, teniendo múltiples opciones de entrada, especificando con la opción -i, y una salida, especificada con una URL. Esto se ve reflejado en el esquema proporcionado en la figura 3.10.

Gestores de bases de datos

Se ha decidido instalar gestores de bases de datos tanto relacionales como no relacionales.

- **Relacionales:** [50] Dentro de este tipo se ha decidido instalar MySQL y SQLite. Dentro de estos gestores, el más reciente es SQLite, cuyo año de lanzamiento es el 2000. Es interesante poder contar con el más antiguo y el más reciente.

  Además de tener en cuenta lo anterior, se han analizado las ventajas y las desventajas de utilizar cada uno. En particular, SQLite está orientado a aplicaciones pequeñas, que no manejen mucha cantidad de datos. En cambio, MySQL es una buena alternativa a SQLite, ya que ofrece una mayor seguridad y un mejor manejo de una gran cantidad de datos de la aplicación correspondiente.

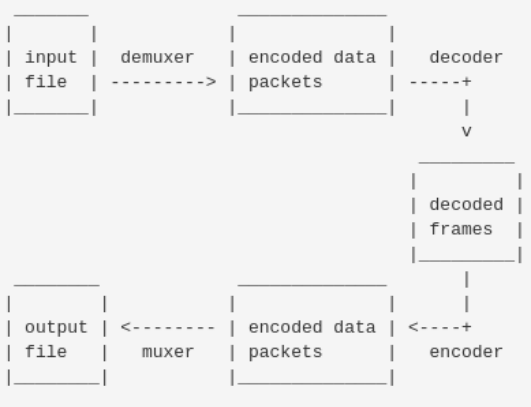

Figura 3.10: Esquema de funcionamiento de FFMPEG [49]

- **No relacionales:** [51] Frente a los anteriores, se encuentra MongoDB. La ventaja de estos es que manejan los datos en ficheros, sin necesidad de estar organizados mediante tablas; gracias a lo cual se puede no tener un esquema fijo. Ante fallos, tiene una recuperación de los datos bastante rápida.

Servidor web

Los servidores web más conocidos actualmente son Apache y Nginx [52]. Sin embargo, este último ha crecido en los últimos años, por lo que he decidido instalar el mismo.

Nginx [53] es un software de código abierto, que se puede usar como servidor web o como un proxy inverso. Este software comenzó como servidor web (HTTP), para dar un alto rendimiento y una alta estabilidad. Sin embargo, esta herramienta fue evolucionando hasta ser usada también como un servidor proxy para correo electrónico (IMAP, POP3, SMTP). También puede ser utilizado como un proxy inverso y equilibrador de carga a su vez para distintos servidores HTTP, TCP y UDP.

3.2. Desarrollo

Con el objetivo de que la NVIDIA Jetson Nano fuese una plataforma para aplicaciones de Computer Vision y deep learning se instalaron todas aquellas tecnologías mencionadas en el apartado anterior. Para ello, se han seguido una serie de pasos, descritos a continuación. Como línea inicial, se ha seguido un tutorial subido a la página de pyimagesearch (véase [54]).

3.2.1. JetPack

Fue necesaria una tarjeta **microSD** para tener la imagen del sistema operativo, con el objetivo de que funcionase la Jetson Nano.

En algunos foros de NVIDIA [55] comentan que es recomendable una tarjeta de 32 o 64GB. En mi

caso, se decidió optar por una de 32GB, ya que, en un primer momento, se pensó que era suficiente. Sin embargo, se observaron problemas con el espacio disponible, por lo que, finalmente, se optó por utilizar una tarjeta de 64GB. Este problema está comentado en el anexo A, en el apartado A.8 más detalladamente. Esto ha sido la causa por la que se recomienda que el tamaño de la microSD sea de **64GB**.

Esta tarjeta fue empleada para flashear el **JetPack de NVIDIA**, proporcionado en la web oficial de esta empresa [56]. Este JetPack es una imagen de un sistema operativo basado en Ubuntu. La versión de JetPack que se ha escogido ha sido la versión 4.2.1, ya que, podría haber problemas de compatibilidad de las versiones más recientes respecto a distintas librerías. Este JetPack, como ya he mencionado anteriormente, es un sistema operativo basado en Ubuntu, cuya versión es **Ubuntu 18.04.5 LTS**.

Para flashear dicha tarjeta, fue necesario contar con un programa que nos ayude con esa imagen de disco. Para ello, se ha utilizado el programa **balenaEtcher** [57], cuya descarga es gratuita desde la página web de dicho producto. Este programa, en función del tamaño de la tarjeta microSD que se emplee, el tiempo que tarde puede variar: cuanto mayor sea el tamaño, más tiempo llevará dicho proceso.

Se ha decidido utilizar el ordenador portátil disponible como soporte para la Jetson Nano, tanto de carga, como de visualización, por lo que ya se podría empezar a utilizar esta herramienta.

3.2.2. Configuración e instalación de dependencias y librerías

Con el objetivo de que se configurase la NVIDIA Jetson Nano, se utilizó la herramienta screen en el ordenador portátil disponible, lo que permitió acceder ya al sistema operativo con una configuración inicial básica.

Los primeros pasos básicos a realizar fueron utilizar la máxima potencia disponible, así como borrar programas que no son necesarios para el desarrollo de aplicaciones de Computer Vision y deep learning; y una actualización de los paquetes software.

Una vez que se completaron los anteriores pasos básicos, se necesitó instalar las dependencias software y de OpenCV porque, en ambos casos, facilitan la codificación y se podrían utilizar en distintas aplicaciones. A continuación, se preparó el entorno virtual de Python 3, ya que era útil para el manejo e instalación de librerías.

Se iba a necesitar de igual forma el compilador Protobuf, con el objetivo de que la librería TensorFlow fuese más rápida, de tal manera que al compilar software, el desarrollador tuviese el beneficio de optimizar la librería o el compilador correspondiente.

Por otro lado, se instaló Jetson Stats, y todas las librerías mencionadas en el apartado 3.1.2, tales

como NumPy, TensorFlow, etc. Requirió especial atención la instalación de OpenCV, ya que es la librería más conocida para el desarrollo de aplicaciones de Computer Vision, y escogiendo la versión más adecuada para contar con la aceleración hardware; así como QT, llevando a cabo la compilación cruzada para su instalación. Además, también se instalaron distintas versiones de FFMPEG, contando con una versión oficial de dicha librería, y un "parche" encontrado en Github (véase [58]), el cual permite la aceleración hardware tanto codificando como decodificando.

Por último, también se configuró la Jetson Nano para poder utilizar TensorRT, aunque ya se tenía instalada esta herramienta desde el JetPack.

Los detalles de instalación de cada uno de los componentes anteriores se incluyen en el anexo A.

4

Pruebas y rendimiento

Con el objetivo de comprobar el correcto funcionamiento de las librerías instaladas, comentadas en los anteriores apartados, y su respectivo rendimiento, se han realizado una serie de pruebas.

Para comenzar se han realizado unas pruebas básicas de ciertas clases de la librería OpenCV en Python, utilizando a su vez otras librerías instaladas y haciendo uso de la documentación oficial de OpenCV (véase [59]). Por otro lado, se ha realizado una serie de transformaciones morfológicas de imágenes gracias también a OpenCV. Además, se han realizado pruebas más complejas, completándose así con cálculos matemáticos de mayor complejidad, así como la implicación de más librerías.

Por otro lado, se han realizado pruebas de rendimiento utilizando FFMPEG, TensorRT junto con librerías y, además, el uso de inferencias neuronales con QT.

4.1. Pruebas

Las primeras pruebas se han realizado con el objetivo de probar cierta funcionalidad de las principales librerías, como es el caso de OpenCV. Todas las pruebas realizadas hacen uso de ésta. Se ha probado la funcionalidad en Python 3, haciendo uso del entorno virtual creado para ello.

4.1.1. Pruebas básicas con OpenCV y otras librerías

Se han realizado una serie de pruebas de funcionalidad básica de OpenCV, aunque también involucra la participación de otras librerías, como es el caso de NumPy o Matplotlib. Para ello, se han utilizado los tutoriales correspondientes de la documentación oficial de OpenCV (véase [60]).

Las pruebas básicas realizadas son las siguientes:

- **Ver vídeo retransmitido en directo por protocolo RTSP**: Para ello, se han tenido que crear 2 hilos, uno para recibir el vídeo y otro para mostrar el mismo, y el manejo también de colas para la comunicación entre ambos hilos. Se ha hecho uso de clases de OpenCV como VideoCapture, y de métodos como read y imshow.
- **Guardar en local y ver a la vez vídeo en directo**: Similar al punto anterior, pero además, guardando dicho vídeo en local, mediante las clases VideoWriter_fourcc y VideoWriter, y con su método correspondiente, write.

- **Cambiar la resolución de un vídeo**: Teniendo un vídeo en local, y gracias a las clases mencionadas anteriormente, podemos cambiar la resolución de un vídeo mediante una llamada al método resize, cambiando la dimensión de cada fotograma.
- **Ver un vídeo en blanco y negro, teniendo el original en color**: Mediante cvtColor, la enumeración COLOR_BGR2GRAY y el método threshold, que, a través de un umbral, puede hacer que quede más o menos contraste en la escala de grises, podemos cambiar el color original de un vídeo, y mostrarlo en blanco y negro.
- **Poner bordes a una imagen**: Una vez que se lee dicha imagen, mediante el método copyMakeBorder se puede incluir borde a la misma. Se ha utilizado la librería matplotlib para realizar la comparación entre algunos tipos de bordes distintos entre sí.
- **Rotar una imagen**: Gracias a los métodos getRotationMatrix2D y warpAffine, se consigue la rotación de una imagen, calculando la matriz de rotación y aplicando dicho argumento a la transformación afín, para que la rotación sea efectiva.
- **Calcular histogramas**: Son gráficos que representan la distribución de intensidad de la imagen correspondiente. Se pueden realizar de diferentes maneras, mediante Matplotlib u OpenCV. El método que se encarga de esto en la librería Matplotlib es hist, y en OpenCV es calcHist.
- **Transformación afín**: En este caso, es una transformación en la que todas las líneas paralelas de la imagen original, seguirán siéndolo en la transformada. Se utiliza la librería NumPy para inicializar las matrices y el método de OpenCV getAffineTransform, que se encarga de calcular la propia transformación afín.
- **Binarización de Otsu**: Es similar a la idea de umbral del que se ha hecho mención en el ejemplo de ver vídeo en blanco y negro. En este caso, se calcula el valor del umbral a partir del histograma correspondiente a la imagen, siendo esta imagen una imagen bimodal (tiene que tener únicamente 2 picos en su histograma). En el caso de que no se trate de una imagen bimodal, la binarización no será tan precisa.
- **Degradados de imagen**: OpenCV proporciona distintos tipos de filtro de degradados, o también llamados filtros de paso alto, como pueden ser Sobel o Laplacian. Laplacian calcula el laplaciano de una imagen, y dentro de dicha función, se llama a su vez al método Sobel, calculando las derivadas de una imagen.
- **Detección de bordes o puntos en la imagen**: El nombre que recibe este algoritmo es Canny Edge detection, diseñado en 1986 por F. Canny con este fin. Se realiza mediante la llamada al método Canny.
- **Buscar líneas**: Hough Line Transform tiene como objetivo detectar cualquier línea, siempre que se pueda representar dicha línea en forma matemática, pudiendo llegar a detectarla en caso de que estuviese rota o un poco distorsionada. Para ello, se pueden utilizar dos métodos: HoughLines o HoughLinesP. La diferencia entre ambos es que HoughLines utiliza la transformada estándar de Hough, mientras que HoughLinesP utiliza la transformada probabilística. También se hace uso de la librería NumPy.
- **Buscar círculos**: Hough Circle Transform tiene como fin detectar cualquier círculo. Se calcula mediante el método HoughCircles. Como en el caso anterior, también se hace uso de NumPy.
- **Extracción de primer plano o quitar fondo**: Mediante el algoritmo GrabCut, método que se realiza mediante cortes de gráficos iterados y el uso de un rectángulo creado por el usuario.
- **Detector de esquinas Harris**: Se trata de un detector de esquinas creado en 1988, combinando distintos tipos de detectores, basándose en una idea matemática. Se puede poner en funcionamiento haciendo uso del método cornerHarris.
- **Detector de esquinas Shi-Tomasi**: Al igual que el caso anterior, es un detector de esquinas. Se realiza mediante el método goodFeaturesToTrack, encontrando las esquinas más fuertes en la imagen en función del algoritmo Harris Corner Detector. Dicho número de esquinas viene determinado por un argumento en la función.
- **SIFT**: Transformación de características de escala invariable. Se trata de un detector de esquinas pero tiene en cuenta la escala, es decir, un punto clave en una escala menor, puede ser 3 puntos claves en una escala mayor, porque se amplía esa imagen. Se aplica este algoritmo mediante llamadas a SIFT, detect y drawKeypoints.

- **SURF**: Funciones robustas aceleradas. Es la versión acelerada de SIFT. Se invoca mediante la llamada de SURF, y también se hace uso de los métodos detect y drawKeypoints.
- **FAST**: Se trata de otro detector de esquinas. Sin embargo, los anteriores algoritmos mencionados no son lo suficientemente rápidos como para trabajar en tiempo real. En cambio, este sí, mediante la llamada a FastFeatureDetector y los métodos mencionados en los casos anteriores.
- **ORB**: Una versión mejorada de FAST, ya que mejora FAST a nivel de algoritmo y, además, utiliza un descriptor BRIEF con modificaciones también para mejorar su rendimiento. Se hace uso de ello con el método ORB.
- **Reducir el número de colores de la imagen**: Hay situaciones en las que se quiere decir memoria y una de las opciones es reducir el número de colores, y esto se consigue mediante la agrupación de K-medias. Se hace uso de NumPy y del método kmeans.
- **Encontrar objetos por coincidencia y homografía**: Mediante los métodos findHomography y perspectiveTransform esto se consigue. El primero de ellos se encarga de encontrar los puntos clave, con lo que se obtiene la transformación en perspectiva del objeto correspondiente. Después se utiliza perspectiveTransform para encontrarlo, necesitando, al menos, 4 puntos. Para su representación se ha hecho uso de la librería Matplotlib. El resultado de aplicar este algoritmo se muestra en la figura 4.1.

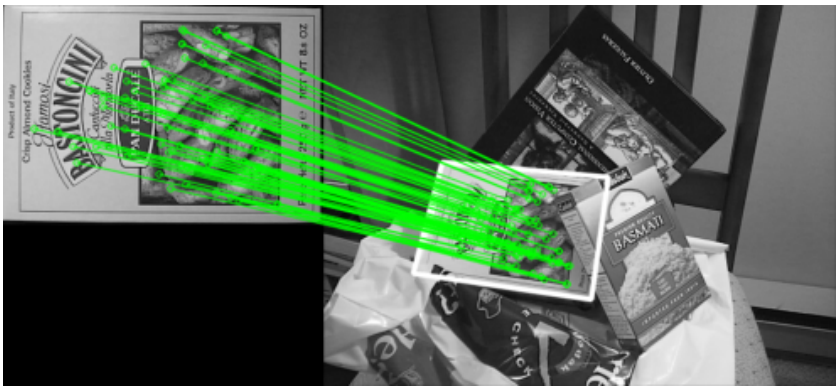

Figura 4.1: Encontrar objetos por coincidencia y homografía

4.1.2. Transformaciones morfológicas con OpenCV

Las transformaciones morfológicas son aquellas operaciones básicas, y se pueden hacer mediante la librería OpenCV. Este tipo de operaciones se aplican a imágenes binarias, basándose así en la forma de la propia imagen.

Los dos operadores morfológicos básicos son [61]:

- **Erosión**: Mediante el método “erode”, hace que un píxel tenga valor 1 si todos los píxeles que están dentro de la ventana del kernel elegido como argumento son 1. Si esto no ocurre, tendrá valor 0.
- **Dilatación**: Mediante el método “dilate”, realiza la operación opuesta a la erosión. Hace que un píxel tenga valor 1 si al menos un píxel de dentro de la ventana del kernel es 1, y valor 0 de lo contrario.

Ambos operadores morfológicos son interesantes [61], ya que se puede eliminar el ruido de una imagen aplicando el proceso de erosión seguido de dilatación, idea que se consigue con el operador apertura.

Además de estos dos operadores, tienen variantes. Entre los cuales se encuentran los siguientes [61]:

- **Apertura**: Mediante el método “”morphologyEx” y la enumeración de OpenCV “MORPH_OPEN”, hace que al proceso de erosión le sigue el proceso de dilatación.
- **Cierre**: Mediante el método “”morphologyEx” y la enumeración de OpenCV “MORPH_CLOSE”, se consigue el proceso contrario al de apertura, es decir, al proceso de dilatación le sigue el proceso de erosión.
- **Gradiente morfológico**: Mediante el método “”morphologyEx” y la enumeración de OpenCV “MORPH_GRADIENT”, se consigue la diferencia entre el proceso de dilatación y el proceso de erosión, con lo que se obtiene el contorno del objeto.

En la figura 4.2 se pueden comprobar las diferencias entre distintos procesos. Para realizar esta comparación también han sido necesarias las librerías NumPy, para el manejo de matrices, y Matplotlib para su representación, además de OpenCV.

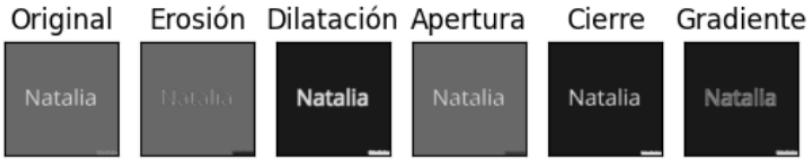

Figura 4.2: Distintas transformaciones morfológicas

4.1.3. Pruebas complejas

Con el objetivo de probar más librerías y la compatibilidad entre ellas, se han realizado pruebas más complejas.

- **Detector de imágenes y vídeos borrosos mediante la transformada rápida de Fourier**: [62] Se ha hecho uso de la transformada rápida de Fourier (FFT) para la detección de imágenes y/o vídeos borrosos o desenfocados, teniendo en cuenta un umbral a partir del cual se considera borroso o no. Se han utilizado diversas librerías como OpenCV, imutils, NumPy, argparse o Matplotlib.
- **Detector de movimiento en vídeo**: [63] Además de esta funcionalidad, esta prueba incluye la retransmisión de dicho vídeo a una página HTML, mediante la librería Flask. Además de esta librería, se han utilizado OpenCV, imutils, NumPy, threading, para el uso de hilos; datetime y time, para el manejo de tiempos. Dentro de OpenCV, además del manejo del vídeo, también se han realizado operaciones morfológicas con el objetivo de reducir el ruido en el mismo.
- **Detector de coincidencia de patrones**: [64] Se utiliza el método matchTemplate de la librería OpenCV. También se ha hecho uso de imutils, NumPy y argparse.
- **Dibujar con OpenCV**: [65] Mediante dicha librería se pueden dibujar rectángulos, círculos y líneas, con los correspondientes métodos: rectangle, circle y line. También se han utilizado las librerías NumPy y argparse.

4.2. Rendimiento

Con el fin de ver el rendimiento de la NVIDIA Jetson Nano, se ha decidido hacer una serie de pruebas en las que se medirán los FPS o el tiempo de ejecución, así como, en los respectivos apartados, ver la aceleración de la GPU mediante la herramienta JetsonStats.

4.2.1. Rendimiento de FFMPEG

En este caso, interesa ver tanto los FPS en las partes de codificación y decodificación, así como la aceleración de la GPU.

Como se ha explicado en el apartado 3.2.2, se dispone tanto de la versión oficial como de un parche no oficial [58] con el que se acelera con la GPU los procesos de codificación y decodificación.

Lo primero a destacar es que la versión oficial [48] contaba con una posible aceleración de la GPU para el proceso de decodificación, como se ha explicado en el apartado A.9 del anexo A. Sin embargo, al medir el rendimiento en el mismo proceso respecto a la versión original no se ha notado ningún cambio en los FPS ni en el uso de la GPU, que seguía con un porcentaje de uso del 0 % en cualquiera de los casos.

Ni la parte de codificación ni la decodificación en la versión oficial de FFMPEG cuenta con aceleración hardware, idea que se ve reflejada con el uso del comando jtop, de JetsonStats, como se ha reflejado un ejemplo de uso del 31 % de GPU en la figura 4.3. Otra idea a destacar es que en el proceso de codificación los FPS descienden conforme aumenta el número del fotograma. Sin embargo, en las tablas 4.1 y 4.2, se muestra el valor inicial de FPS, es decir, el número máximo. Además, en el proceso de decodificación los FPS descienden rápidamente.

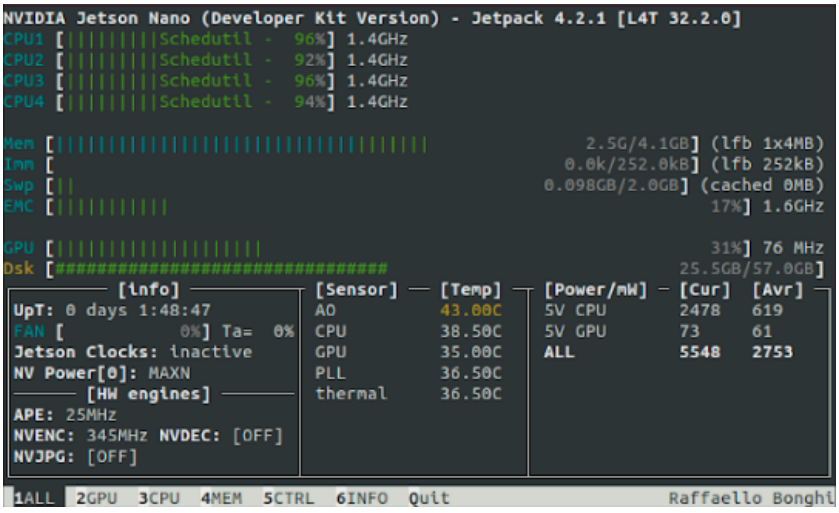

Figura 4.3: Aceleración de la GPU mediante el comando jtop

En la versión no oficial [58], en el proceso de codificación los FPS se mantienen constantes a lo largo del mismo. Por lo contrario, en la parte de decodificación sí que disminuye dicho número bastante rápido.

Se ha decidido crear dos tablas para comparar los valores entre ambos. Debido a que en el proceso de decodificación el valor de FPS en ambos casos disminuye, se ha decidido medir en esos casos la velocidad del programa al final del proceso correspondiente.

La tabla 4.1 expone los valores extraídos de ambos procesos con un vídeo de larga duración.

Versión	Proceso	Valor FPS inicial	Tipo FPS	Speed
Oficial	Codificación	28	Descendente	-
No oficial	Codificación	48	Constante (entre 46 y 49)	-
Oficial	Decodificación	156	Descendente	0.84x
No oficial	Decodificación	173	Descendente	1.35x

Tabla 4.1: Tabla del rendimiento de FFMPEG de un vídeo de larga duración

La tabla 4.2 expone los valores extraídos de ambos procesos con un vídeo de corta duración.

Versión	Proceso	Valor FPS inicial	Tipo FPS	Speed
Oficial	Codificación	56	Descendente	-
No oficial	Codificación	104	Constante (entre 101 y 104)	-
Oficial	Decodificación	0	Descendente	62.6x
No oficial	Decodificación	0	Descendente	65.5x

Tabla 4.2: Tabla del rendimiento de FFMPEG de un vídeo de corta duración

Como se puede observar, en el caso del proceso de codificación el valor de FPS es aproximadamente la mitad con un mismo vídeo en el caso de la versión no oficial, debiéndose a la aceleración hardware que ello provoca. En el proceso de decodificación no hay una diferencia tan notable, aunque el valor de la velocidad lo supera a la versión no oficial, por lo que en ambos casos tiene mejor rendimiento dicha versión.

En la versión no oficial el porcentaje de uso de la GPU máximo alcanzado es del 54 %, mientras que, en la oficial, como se ha mencionado anteriormente, es de un 0

4.2.2. Rendimiento con TensorRT

Otra herramienta interesante es TensorRT. Para probarla, se ha decidido utilizar diferentes redes neuronales para la detección de objetos, como GoogLeNet, MTCNN y SSD con diferentes modelos. Todas estas pruebas se han realizado en el entorno virtual de Python, comprobando una serie de pruebas creadas para comprobar la aceleración (véase [66]).

GoogLeNet

Se trata de una red neuronal [67], siendo una variante de otra red neuronal creada por desarrolladores de Google. Tiene una tasa de error baja [68], siendo de un 6.67 % en la parte de clasificación. La combinación de esta red neuronal con TensorRT, hace que se lleguen a alcanzar los 20 FPS en reconocimiento de objetos en vídeos en directo.

MTCNN

MTCNN (Multi-Task Cascaded Convolutional Neural Networks) [69] es una red neuronal que se encarga de detectar rostros, a través de puntos claves de los mismos, como son los ojos, la nariz o los extremos de la boca. Tiene una tasa de error del 11,28 % [70]. La combinación del uso de esta red neuronal junto con TensorRT, hace que se lleguen a alcanzar los 5 FPS en una imagen con múltiples rostros, y en un vídeo con un solo rostro se alcanzan los 15 FPS, aproximadamente.

SSD

Se trata de una red neuronal para la detección de objetos [71]. En este caso, se han utilizado dos modelos de datos.

Por un lado, se encuentra COCO [72], tratándose de un conjunto de datos para detectar objetos, diferenciando 80 clases de los mismos. Se ha probado con dos versiones diferentes, con la primera se llegan a alcanzar 23 FPS, y en el caso de la segunda versión se alcanzan 27 FPS. Por otro, se encuentra Egohands, tratándose de un conjunto de datos para detectar manos. Se ha probado con dos versiones diferentes. Con una de ellas se alcanzan los 29 FPS, mientras que con la segunda se alcanzan los 33 FPS.

4.2.3. Rendimiento de inferencias neuronales

Se ha realizado una serie de pruebas de rendimiento de inferencias neuronales utilizando Qt, el framework más popular de C++. Se han hecho mediciones de tiempos de propagación (forward time) teniendo en cuenta diferentes redes neuronales, así como distintos desarrollos. Las redes neuronales que se han utilizado son YOLOv3 y Mobilenet SSD 300. Para esto, la librería OpenCV cuenta con una clase llamada dnn para realizar todo este tipo de operaciones. Además, cuenta con los métodos setPreferableBackend y setPreferableTarget, a través de los cuales se puede escoger el modo de ejecución, es decir, se puede escoger entre ejecutarlo en CPU, con CUDA (lo que implica el uso de GPU) , CUDA con coma flotante 16, o Vulkan [73], siendo esta una API de bajo nivel con la que también se utiliza la GPU.

YOLOv3

YOLOv3 es una red neuronal pesada implementada en el framework de Darknet [74], y a través de ella se pueden detectar objetos en imágenes. Se ha probado la identificación de objetos de una sola imagen, midiendo su tiempo de inicialización y su tiempo de propagación. Se han realizado 50 mediciones, y de cada medición también se ha tenido en cuenta el modo de ejecución, es decir, si se ha ejecutado mediante CPU, CUDA, CUDA con coma flotante (CUDAFP16) o Vulkan (Vkcom). Se han agrupado los datos resultantes de dichas pruebas en una gráfica con el historial correspondiente, haciendo uso de la librería Matplotlib para su representación. La figura 4.4 representa el historial de dichas mediciones.

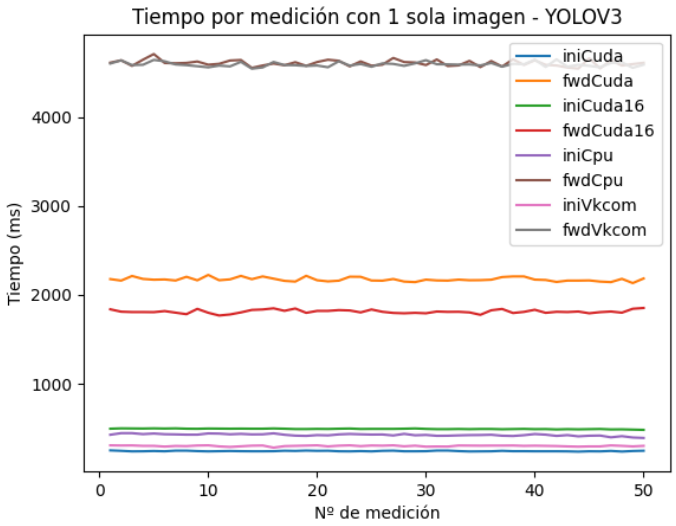

Figura 4.4: Gráfica de tiempos con la red YOLOv3

El mejor resultado del tiempo de propagación (forward) es el que corresponde a su ejecución con CUDA con coma flotante (fwdCuda16 en la leyenda de la gráfica de la figura 4.4), lo que hace indicar que se ha producido aceleración hardware. Por otro lado, se han obtenido las medias de dichos resultados, siendo los representados en la tabla 4.3.

Comparando estos resultados, se observa que en el caso de una red neuronal pesada, se gana tiempo utilizando CUDA, especialmente si se ejecuta con coma flotante 16, reduciendo a menos de la mitad su tiempo de propagación, ya que 4,6 segundos serían demasiados para poder trabajar en tiempo real (dicho resultado es el correspondiente al de ejecución en CPU).

Mobilenet SSD 300

En este caso, es una red neuronal con menos peso y, por tanto, debe ser más rápida que YOLOv3. Se ha realizado el mismo tipo de pruebas, obteniendo el historial correspondiente a la figura 4.5.

Tiempo medido	Modo de ejecución	Media de tiempo (ms)
Inicialización	CUDA	242.12
Inicialización	CUDA FP16	491.9
Inicialización	CPU	423.9
Inicialización	Vulkan	300.14
Forward	CUDA	2174.18
Forward	CUDA FP16	1812.48
Forward	CPU	4608.38
Forward	Vulkan	4593.86

Tabla 4.3: Tabla de tiempos medios de la ejecución con YOLOv3

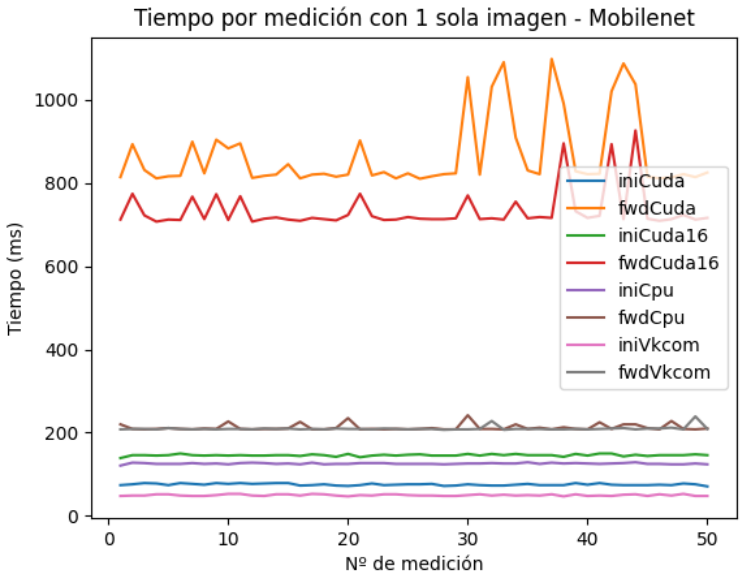

Figura 4.5: Gráfica de tiempos con la red Mobilenet SSD 300

Las medias de dichos resultados son las correspondientes a las representadas en la tabla 4.4.

Tiempo medido	Modo de ejecución	Media de tiempo (ms)
Inicialización	CUDA	75.54
Inicialización	CUDA FP16	145.94
Inicialización	CPU	125.74
Inicialización	Vulkan	49.92
Forward	CUDA	867.52
Forward	CUDA FP16	733.56
Forward	CPU	212.52
Forward	Vulkan	209.86

Tabla 4.4: Tabla de tiempos medios de la ejecución con Mobilenet SSD 300

En el caso de esta red, al tratarse de una red neuronal más ligera, no se ve tan clara la ganancia de ejecutarlo con aceleración hardware compleja, ya que se puede observar, tanto en la tabla de medias de resultados (tabla 4.4) como en la gráfica que los modos de ejecución más rápidos (figura 4.5) son los correspondientes a Vulkan y a CPU, siendo ligeramente mejor el de Vulkan.

5

Conclusiones

La plataforma NVIDIA Jetson Nano ha demostrado ser un dispositivo de gran valor para el desarrollo de aplicaciones de visión artificial. Se ha conseguido instalar una gran variedad de herramientas, como Jetson-Stats, OpenCV o Qt, lo que ofrece una gran versatilidad en el desarrollo de aplicaciones. Sin embargo, la instalación de software no es trivial. A pesar de contar con el Jetson-Pack (firmware), la rápida evolución de las herramientas y diferentes librerías hace imprescindible instalar nuevos componentes que generan conflictos de versiones entre las librerías. Por este motivo, no se puede valorar esta plataforma como demasiado amigable para usuario no expertos, y, en cualquier caso, desplegar el conjunto total de herramientas implica un cierto tiempo.

Desde el punto de vista de rendimiento, cabe destacar que el uso de la GPU nos ha permitido codificar como decodificar vídeos con FFMPEG, lo que ha mejorado el tiempo de ejecución respecto a la versión oficial. Esto convierte a la Jetson Nano en un dispositivo interesante para un tipo de aplicaciones que utilice FFMPEG.

Por otro lado, una de las principales herramientas en Computer Vision son las redes neuronales. La NVIDIA Jetson Nano se defiende correctamente cuando se usa GoogleNet o Mobilnet SSD 300, llegando a alcanzar 33 FPS, junto con el uso de TensorRT, lo que es suficiente para aplicaciones de detección de objetos utilizando dicha dicha red neuronal. Sin embargo con redes neruonales más pesadas, como puede ser YOLOv3, el rendimiento disminuye considerablemente, pudiendo ser insuficiente para dicho tipo de aplicaciones, ya que para la detección de objetos de una única imagen lleva un tiempo de ejecución elevado.

NVIDIA ofrece otro tipo de plataformas que pueden ayudar a solventar los problemas encontrados con las redes neuronales profundas. Sin embargo, no ofrecen un coste y un consumo tan reducidos como la Jetson Nano. La evaluación de estas plataformas sería una ampliación interesante del trabajo descrito en este documento.

6

Glosario, acrónimos y definiciones

API Mediante un conjunto de reglas y protocolos, permite la comunicación entre dos aplicaciones de software, permitiendo así el desarrollo e integración de las mismas.

BSD Distribución de software Berkeley. Se trata de un tipo de licencia permisiva de cara a estos sistemas, siendo estos un tipo del sistema operativo Unix.

CPU Unidad central de procesamiento, encargada de tratar las instrucciones del sistema operativo.

CUDA Modelo de NVIDIA de programación en paralelo, mediante la aceleración hardware.

DVFS Dynamic Voltage and Frequency Scaling. Escala dinámica de voltaje y frecuencia, permitiendo que utilicen la mínima energía posible.

FPS Fotogramas por segundo.

GCC Conjunto de compiladores creados por GNU (sistema operativo de tipo Unix), tratándose así de software libre.

GPU Unidad de procesamiento gráfico, a través de la cual se puede conseguir la aceleración hardware de ciertos procesos.

GUI Interfaz gráfica de usuario.

IA Inteligencia artificial.

IDE Entorno de desarrollo integrado: creado con el objetivo de facilitar el desarrollo de aplicaciones al programador.

RTSP Protocolo a través del cual se transmite en tiempo real.

Bibliografía

[1] E. A. Group, "El secreto de las aplicaciones de negocio de computer vision." https://blog.enzymeadvisinggroup.com/computer-vision-aplicaciones-negocio. Visitado: 15-06-2021.

[2] Microsoft, "¿qué es computer vision?." https://docs.microsoft.com/es-es/azure/cognitive-services/computer-vision/overview. Visitado: 15-06-2021.

[3] Nae, "Expectativas de mercado para la inteligencia artificial." https://nae.global/es/expectativas-de-mercado-para-la-inteligencia-artificial/. Visitado: 15-06-2021.

[4] statista, "Las aplicaciones más rentables de la inteligencia artificial." https://es.statista.com/grafico/9437/las-aplicaciones-mas-rentables-de-la-inteligencia-artifi Visitado: 15-06-2021.

[5] NVIDIA, "Kit para desarrolladores jetson nano para ia y robótica." https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/. Visitado: 15-06-2021.

[6] NVIDIA, "Nvidia jetson nano developer kit." https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Visitado: 15-06-2021.

[7] NVIDIA, "Jetson community projects." https://developer.nvidia.com/embedded/community/jetson-projects. Visitado: 15-06-2021.

[8] Jetbot, "Jetbot." https://jetbot.org/v0.4.3/. Visitado: 15-06-2021.

[9] jaybdub, "Real-time pose estimation accelerated with nvidia tensorrt." https://github.com/NVIDIA-AI-IOT/trt_pose. Visitado: 15-06-2021.

[10] jaybdub, "Update 2/25/2021 to include hand pose." https://forums.developer.nvidia.com/t/update-2-25-2021-to-include-hand-pose-real-time-human-pose-estimati 83549. Visitado: 15-06-2021.

[11] winter2897, "Real-time auto license plate recognition with jetson nano." https://github.com/winter2897/Real-time-Auto-License-Plate-Recognition-with-Jetson-Nano. Visitado: 15-06-2021.

[12] oliver almaraz, "Ai food container identifier for the jetson nano." https://github.com/oliver-almaraz/food_container_identifier. Visitado: 15-06-2021.

[13] bdtinc, "Ai food container identifier for the jetson nano." https://github.com/bdtinc/maskcam. Visitado: 15-06-2021.

[14] haotian liu, "Yolact edge." https://github.com/haotian-liu/yolact_edge. Visitado: 15-06-2021.

[15] Google, "Google trends." https://trends.google.es/trends/?geo=ES. Visitado: 15-06-2021.

[16] python, "Pep 373 – python 2.7 release." https://www.python.org/dev/peps/pep-0373/. Visitado: 15-06-2021.

[17] Pypi, "Cython – pypi." https://pypi.org/project/Cython/. Visitado: 15-06-2021.

[18] NVIDIA, "Keras and scipy throwing weird errors." https://forums.developer.nvidia.com/t/keras-and-scipy-throwing-weird-errors-while-installing-on-jetson-nano-no-li 107220#5415052. Visitado: 15-06-2021.

[19] L. Rendek, "How to install g++ the c++ compiler." https://linuxconfig.org/how-to-install-g-the-c-compiler-on-ubuntu-18-04-bionic-beaver-linux. Visitado: 15-06-2021.

[20] Github, "About - github." https://github.com/about. Visitado: 15-06-2021.

[21] CMake, "Overview - cmake." https://cmake.org/overview/. Visitado: 15-06-2021.

[22] Virtualenv, "Virtualenv - virtualenv 20.4.8." https://virtualenv.pypa.io/en/latest/. Visitado: 15-06-2021.

[23] Pypi, "Jetson-stats . pypi." https://pypi.org/project/jetson-stats/. Visitado: 15-06-2021.

[24] OpenCV, "Cuda - opencv." https://opencv.org/platforms/cuda/. Visitado: 15-06-2021.

[25] NVIDIA, "Nvidia tensorrt." https://bit.ly/3zwyssb. Visitado: 15-06-2021.

[26] NumPy, "Numpy." https://numpy.org/. Visitado: 15-06-2021.

[27] S. Dahal, "Understanding numpy for computer vision." https://www.suntos.com.np/computer-vision-for-robotics/understanding-numpy-for-computer-vision.html#what-is-numpy-routine-for-computing- Visitado: 15-06-2021.

[28] SciPy, "Scipy library." https://www.scipy.org/scipylib/index.html. Visitado: 15-06-2021.

[29] TensorFlow, "Por qué tensorflow." https://www.tensorflow.org/?hl=es-419. Visitado: 15-06-2021.

[30] scikit learn, "Scikit-learn." https://scikit-learn.org/stable/index.html. Visitado: 15-06-2021.

[31] PyTorch, "Pytorch." https://pytorch.org/. Visitado: 15-06-2021.

[32] PyTorch, "Tensors - pytorch." https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html. Visitado: 15-06-2021.

[33] keras, "About keras." https://keras.io/about/. Visitado: 15-06-2021.

[34] Matplotlib, "Matplotlib : Python plotting." https://matplotlib.org/. Visitado: 15-06-2021.

[35] Pillow, "Pillow." https://pillow.readthedocs.io/en/stable/. Visitado: 15-06-2021.

[36] Pypi, "imutils . pypi." https://pypi.org/project/imutils/. Visitado: 15-06-2021.

[37] OpenCV, "Opencv." https://opencv.org/. Visitado: 15-06-2021.

[38] L. del Valle Hernández, "Visión artificial, opencv y python, primeros pasos para analizar imágenes." https://programarfacil.com/podcast/81-vision-artificial-opencv-phyton/. Visitado: 15-06-2021.

[39] R. Marín, "Opencv, instalación en python y ejemplos básicos." https://revistadigital.inesem.es/informatica-y-tics/opencv/. Visitado: 15-06-2021.

[40] L. Gracia, "¿qué es opencv?." https://unpocodejava.com/2013/10/09/que-es-opencv/. Visitado: 15-06-2021.

[41] React, "React – una biblioteca de javascript para construir interfaces de usuario." https://es.reactjs.org/. Visitado: 15-06-2021.

[42] S. Daityari, "Angular vs react vs vue: Which framework to choose in 2021." https://www.codeinwp.com/blog/angular-vs-vue-vs-react/. Visitado: 15-06-2021.

[43] FastAPI, "Fastapi." https://fastapi.tiangolo.com/. Visitado: 15-06-2021.

[44] PyPI, "Flask - pypi." https://pypi.org/project/Flask/. Visitado: 15-06-2021.

[45] Qt, "Qt wiki." https://wiki.qt.io/Main. Visitado: 15-06-2021.

[46] M. L. Mendoza, "Frameworks c++." https://openwebinars.net/blog/frameworks-c/. Visitado: 15-06-2021.

[47] D. Coppola, "Market share of the most used c/c++ ides in 2018, stats and estimates." http://blog.davidecoppola.com/2018/02/market-share-most-used-c-cpp-ides-in-2018-statistics-estimates/. Visitado: 15-06-2021.

[48] FFMPEG, "About ffmpeg." https://www.ffmpeg.org/about.html. Visitado: 15-06-2021.

[49] FFMPEG, "ffmpeg documentation." https://www.ffmpeg.org/ffmpeg.html. Visitado: 15-06-2021.

[50] A. GuiaDev, "Mysql vs sqlite: ¿cuál es mejor? comparativa." https://guiadev.com/mysql-vs-sqlite/. Visitado: 15-06-2021.

[51] V. GuiaDev, "Mysql vs mongodb: Diferencias, ventajas y desventajas." https://guiadev.com/mysql-vs-mongodb/. Visitado: 15-06-2021.

[52] WebEmpresa, "¿qué es un servidor web y para qué sirve?." https://www.webempresa.com/hosting/que-es-servidor-web.html. Visitado: 15-06-2021.

[53] NGINX, "What is nginx? nginx." https://www.nginx.com/resources/glossary/nginx/. Visitado: 15-06-2021.

[54] A. R. pyimagesearch, "How to configure your nvidia jetson nano for computer vision and deep learning." https://www.pyimagesearch.com/2020/03/25/how-to-configure-your-nvidia-jetson-nano-for-computer-vision-and-deep-lea Visitado: 15-06-2021.

[55] N. Forums, "[solved] warning disk space remaining using a 64 gb sdcard." https://forums.developer.nvidia.com/t/solved-warning-disk-space-remaining-using-a-64-gb-sdcard/79905. Visita-

do: 15-06-2021.

[56] NVIDIA, "Jetpack sdk." https://developer.nvidia.com/embedded/jetpack. Visitado: 15-06-2021.

[57] balena, "balenaetcher - flash os images to sd cards & usb drives." https://www.balena.io/etcher/. Visitado: 15-06-2021.

[58] jocover, "jetson-ffmpeg: ffmpeg support on jetson nano." https://github.com/jocover/jetson-ffmpeg. Visitado: 15-06-2021.

[59] OpenCV, "Opencv: cv namespace reference." https://docs.opencv.org/4.2.0/d2/d75/namespacecv.html. Visitado: 15-06-2021.

[60] OpenCV, "Opencv: Opencv-python tutorials." https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html. Visitado: 15-06-2021.

[61] Unipython, "Transformaciones morfológicas." https://unipython.com/transformaciones-morfologicas/. Visitado: 15-06-2021.

[62] A. R. Pyimagesearch, "Opencv fast fourier transform (fft) for blur detection in images and video streams." https://www.pyimagesearch.com/2020/06/15/opencv-fast-fourier-transform-fft-for-blur-detection-in-images-and-video-streams/ Visitado: 15-06-2021.

[63] A. R. Pyimagesearch, "Opencv – stream video to web browser/html page." https://www.pyimagesearch.com/2019/09/02/opencv-stream-video-to-web-browser-html-page/. Visitado: 15-06-2021.

[64] A. R. Pyimagesearch, "Multi-template matching with opencv." https://www.pyimagesearch.com/2021/03/29/multi-template-matching-with-opencv/. Visitado: 15-06-2021.

[65] A. R. Pyimagesearch, "Drawing with opencv." https://www.pyimagesearch.com/2021/01/27/drawing-with-opencv/. Visitado: 15-06-2021.

[66] jkjung avt, "tensorrt_demos: Tensorrt yolov4, yolov3, ssd, mtcnn, and googlenet." https://github.com/jkjung-avt/tensorrt_demos. Visitado: 15-06-2021.

[67] R. Alake, "Deep learning: Googlenet explained." https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765. Visitado: 15-06-2021.

[68] pawangfg GeeksforGeeks, "Understanding googlenet model - cnn architecture." geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/. Visitado: 15-06-2021.

[69] L. Dulčić, "Face recognition with facenet and mtcnn." https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/. Visitado: 15-06-2021.

[70] Y. Zhang, "A deep learning approach for face detection and location on highway." https://www.researchgate.net/publication/328749692_A_Deep_Learning_Approach_for_Face_Detection_and_Location_on_Highway. Visitado: 15-06-2021.

[71] J. H. Medium, "Ssd object detection: Single shot multibox detec-

tor for real-time processing." https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processi~:text=SSD%20is%20a%20single%2Dshot,offsets%20to%20default%20boundary%20boxes. Visitado: 15-06-2021.

[72] npm, "Object detection (coco-ssd)." https://www.npmjs.com/package/@tensorflow-models/coco-ssd. Visitado: 15-06-2021.

[73] NVIDIA, "Vulkan support on l4t." https://developer.nvidia.com/embedded/vulkan#:~:text=Vulkan%20is%20a%20low%20level,%C2%AE%20or%20OpenGL%C2%AE%20ES.&text=Vulkan%20is%20the%20first%20new,API%20that%20is%20cross%20platform. Visitado: 15-06-2021.

[74] A. Kathuria, "What's new in yolo v3?." https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. Visitado: 15-06-2021.

[75] NVIDIA, "what does jetson_clocks do?." https://forums.developer.nvidia.com/t/what-does-jetson-clocks-do/73788. Visitado: 15-06-2021.

[76] NVIDIA, "Extremely long time to load trt-optimized frozen tf graphs." https://forums.developer.nvidia.com/t/extremely-long-time-to-load-trt-optimized-frozen-tf-graphs/69628. Visitado: 15-06-2021.

[77] NVIDIA, "Scipy not getting installed on jetson nano inspite of all dependencies." https://forums.developer.nvidia.com/t/scipy-not-getting-installed-on-jetson-nano-inspite-of-all-dependencies/110034. Visitado: 15-06-2021.

[78] D. Sayidi, "Install and setup react app on ubuntu 18.04.3 lts." https://medium.com/@DanielSayidi/install-and-setup-react-app-on-ubuntu-18-04-3-lts-fcd2c8758 Visitado: 15-06-2021.

[79] NVIDIA, "Pytorch for jetson - version 1.8.0 now available." https://forums.developer.nvidia.com/t/pytorch-for-jetson-version-1-8-0-now-available/72048. Visitado: 15-06-2021.

[80] D. A., "Instala qt creator y compila tu primer programa en ubuntu." https://ubunlog.com/instala-qt-creator-compila-programa/. Visitado: 15-06-2021.

[81] L. Agocs, "Instala qt creator y compila tu primer programa en ubuntu." https://www.qt.io/blog/2016/11/10/qt-nvidia-jetson-tx1-device-creation-style. Visitado: 15-06-2021.

[82] NVIDIA, "Cross compile qt for jetson nano problem." https://forums.developer.nvidia.com/t/cross-compile-qt-for-jetson-nano-problem/164215. Visitado: 15-06-2021.

[83] AskUbuntu, "Qt5 installation and path configuration." https://askubuntu.com/questions/435564/qt5-installation-and-path-configuration. Visitado:

15-06-2021.

[84] NVIDIA, “Nvidia jetson linux developer guide : Introduction | nvidia docs.” https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/multimedia.html#wwpID0E0JB0HA. Visitado: 15-06-2021.

[85] NVIDIA, “Jetson download center | nvidia.” https://developer.nvidia.com/embedded/downloads. Visitado: 15-06-2021.

Apéndices

A

Configuración e instalación de dependencias y librerías

A.1. Configuración inicial

Como se ha mencionado en el apartado 3.2, como línea general se ha seguido un tutorial de pyimagesearch (véase [54]).

Lo primero a realizar es instalar, si no se tiene instalado previamente en nuestro ordenador, la herramienta screen, introduciendo el siguiente comando por consola:

sudo apt-get install screen

Se trata de un administrador de ventanas, a través del cual se realizará la configuración de la Jetson Nano. Una vez que se tenga todo conectado y funcionando en la Jetson Nano, lo primero que se verá en nuestro portátil será el dispositivo conectado vía USB con el nombre L4T-README. Cuando esto se obtenga, se tendrá que introducir el siguiente comando en la terminal:

ls /dev/ttyACM*

Si se obtiene una salida en dicho comando, en mi caso es /dev/ttyACM0, ese nombre será el correspondiente a la Jetson Nano, y se podrá proceder a ejecutar el siguiente comando:

sudo screen /dev/ttyACM0 115200 -L

Después de esto se procede a la configuración básica, como una imagen del sistema operativo de Ubuntu, debiendo introducir la ubicación, el lenguaje, así como usuario y contraseña. A continuación, se procede a la configuración de red. En mi caso, preferí servirme, en un primer momento, de un adaptador inalámbrico de red, por lo que debí elegir dicha red wifi, así como la contraseña; aunque también he utilizado Ethernet, conectando el cable correspondiente. Una vez que se ha terminado este proceso de configuración, el dispositivo pide hacer login, por lo que se tiene que introducir el usuario y la contraseña previamente elegidos.

Para mayor comodidad y mejor manejo, se conecta vía ssh a la Jetson Nano; ya que ahora ya se ha configurado la herramienta y se puede obtener la IP correspondiente.

A.2. Primeros pasos básicos

Con el objetivo de utilizar la Jetson Nano con la máxima potencia posible, se establece el valor de voltaje, a través del comando nvpmodel: 5W es el modelo 1 y 10W, el modelo 0. Por defecto es el voltaje más alto, pero se recomienda forzarlo antes de ejecutar el comando que se describe a continuación de este. El comando para establecer el nvpmodel es el siguiente:

sudo nvpmodel -m 0

Después, ejecutar el comando jetson_clocks [75]. Se trata de un script que deshabilita el regulador DVFS y bloquea los relojes a sus máximos definido por nvpmodel.

sudo jetson_clocks

Después de haber configurado la Jetson Nano para utilizar la máxima potencia, se podrá purgar LibreOffice, ya que ocupa demasiado espacio y no se utilizará ni se necesita para el campo de Computer Vision y deep learning. Una vez hecho esto, se tiene que actualizar los paquetes a nivel de sistema.

sudo apt-get update && sudo apt-get upgrade

A.3. Instalación de dependencias software y dependencias de OpenCV

Instalar aquellas dependencias software, como pueden ser los editores de texto, en mi caso, nano, como se ha decidido y explicado en el apartado X. Además, también se han instalado los paquetes git, cmake, gfortran, python3-dev, protobuf-compiler y cython3.

Por otro lado, también se instalarán todas las dependencias de OpenCV, como es el metapaquete build-essential o V4L.

A.4. Entorno virtual Python 3

Instalar pip. Esto se hará a través del comando:

wget https://bootstrap.pypa.io/get-pip.py sudo python3 get-pip.py

Una vez ejecutados estos dos comandos, eliminar el fichero correspondiente. Además, se instalarán los paquetes virtualenv y virtualenvwrapper y editar el fichero /.bashrc añadiendo las siguientes líneas al final de dicho archivo:

export WORKON_HOME=$HOME/.virtualenvs

export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3

source /usr/local/bin/virtualenvwrapper.sh

Ya añadidas estas líneas se ejecutará el comando:

source /.bashrc

Terminado este proceso de instalación, se creará el entorno virtual con Python 3, llamado, en mi caso py3cv4 (para tener en cuenta en otros apartados). A partir de aquí, se trabajará sobre este nuevo entorno virtual.

A.5. Compilador Protobuf

Con el objetivo de que TensorFlow sea más rápido, se configurará protobuf y libprotobuf para ello. Si se instala TensorFlow vía pip, se instala una versión de protobuf con un rendimiento demasiado lento, tal y como comentan en un foro de desarrolladores de NVIDIA (véase [76]).

Para comenzar con este apartado, se instalará una implementación eficiente de dicho compilador, con la versión 3.6.1, mediante los siguientes comandos:

wget https://raw.githubusercontent.com/jkjung-avt/jetson_nano/master/install_protobuf-3.6.1.sh

sudo chmod + x install_protobuf-3.6.1.sh

./install_profobuf-3.6.1.sh

Este proceso es largo, ya que lleva una duración de 1 hora aproximadamente. Una vez terminado dicho proceso, se tendrá que instalar en el entorno virtual creado anteriormente (py3cv4).

workon py3cv4

cd ~

cp -r /src/protobuf-3.6.1/python/ .

cd python

python setup.py install -cpp_implementation

Se instala mediante setup.py, para obtener el beneficio de la compilación de software específico para el procesador de la Jetson Nano, para optimizar la librería o el compilador correspondiente, en vez de utilizar paquetes precompilados.

A.6. Instalación JetsonStats y librerías

Además de la herramientas JetsonStats, se instalan otras librerías.

A.6.1. Jetson-Stats

La herramienta JetsonStats para la monitorización de la NVIDIA Jetson Nano se instala vía pip.

A.6.2. NumPy

La librería NumPy se ha instalado vía pip, pero con permisos:

sudo pip install numpy

Para que sea instalado correctamente en el entorno virtual, se tendrán que usar los siguientes comandos:

cd ~/.virtualenvs/py3cv4/lib/python3.6/site-packages/

ln -s /usr/local/lib/python3.6/dist-packages/numpy numpy

cd ~

Para comprobar su correcto funcionamiento, se trabaja sobre el entorno virtual y se prueba a hacer el import:

workon py3cv4

python

>import numpy

En dicho import sale el siguiente error: Illegal instruction (core dumped)

Para solucionar dicho error, se ha recurrido a realizar un export de la siguiente variable, incorporándolo en el fichero /.bashrc, con el objetivo de que sea permanente:

export OPENBLAS_CORETYPE=ARMV8

Una vez que se ha hecho esto, se puede ver que dicho import se realiza sin ningún tipo de problema.

A.6.3. Cython

Esta librería se ha instalado vía pip, sin ser necesarios privilegios:

pip install cython

A.6.4. SciPy

Se ha decidido instalar la versión 1.3.3, ya que es compatible con la versión de TensorFlow más adecuada para la Jetson Nano. Es por este motivo por el que no se debe instalar vía pip, tal y como indican en un foro de NVIDIA (véase [77]). Para ello, se han ejecutado los siguientes comandos:

wget https://github.com/scipy/scipy/releases/download/v1.3.3/scipy-1.3.3.tar.gz

tar -xzvf scipy-1.3.3.tar.gz scipy-1.3.3

cd scipy-1.3.3/

python setup.py install

Esta instalación requiere un tiempo aproximado de 35 minutos.

A.6.5. TensorFlow

Como se ha mencionado en el apartado A.6.4 del anexo A, se instalará la versión de TensorFlow más adecuada para la Jetson Nano. En este caso, se trata de la versión de NVIDIA 1.13, ya que está optimizada para dicho dispositivo. Además, se ha decidido no instalar TensorFlow 2.0 debido a las posibles incompatibilidades con TensorRT, herramienta de la que se dispone con el JetPack de NVIDIA.

Para instalar esta librería con la versión mencionada, se ejecuta el siguiente comando:

pip install –extra-index-url https://developer.download.nvidia.com/compute/redist/jp/v42 tensorflow-gpu==1.13.1+nv19.3

A.6.6. Keras

Keras se instala vía pip:

pip install keras

Sin embargo, al momento de ejecutar dicha sentencia, reporta el siguiente error:

ImportError: Keras requires TensorFlow 2.2 or higher.

Esto es porque la versión de Keras actual requiere la versión detallada de TensorFlow. Para que dichas herramientas sean compatibles entre sí, se instala la versión 2.1.5:

pip install keras==2.1.5

A.6.7. Matplotlib, scikit-learn, pillow, imutils, flask, FastAPI

Las librerías Matplotlib, scikit-learn, pillow, imutils y flask se han instalado vía pip.

La instalación de FastAPI requiere a su vez, la instalación de uvicorn, ambas también vía pip.

A.6.8. MySQL, SQLite, MongoDB

Se ha instalado MySQL mediante el siguiente comando:

sudo apt install mysql-server

A continuación, configurar dicho gestor de base de datos, ajustando la autenticación y los privilegios de los usuarios.

SQLite3 se ha instalado ejecutando lo siguiente:

sudo apt-get install sqlite3

Un punto negativo en este punto es que ninguna de las herramientas para la gestión de bases de datos son compatibles con la Jetson Nano. Es decir, no se puede instalar ni SQLite Browser, ni DBeaver, ni dbForge, ni HeidiSQL. Por lo que, toda la visualización de estas bases de datos deberá ser por consola.

MongoDB también se ha instalado mediante apt:

sudo apt install -y mongodb

A.6.9. Nginx

Nginx puede ser instalado con el siguiente comando:

sudo apt install nginx

A.6.10. React

Para instalar React [78], es necesario primero instalar Nodejs, ejecutando los siguientes comandos:

sudo apt-get install curl

curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -

sudo apt-get install -y nodejs

Una vez hecho esto, se instalará create-react-app mediante:

npm install -g create-react-app

A partir de haber ejecutado el comando anterior, se puede crear una app de React.

A.6.11. PyTorch

Para instalar la versión más adecuada en la NVIDIA Jetson Nano, se ha recurrido a un foro de dicha compañía [79], siguiendo los siguientes pasos:

wget https://nvidia.box.com/shared/static/j2dn48btaxosqp0zremqqm8pjelriyvs.whl -O torch-1.1.0-cp36-cp36m-linux_aarch64.whl

pip install torch-1.1.0-cp36-cp36m-linux_aarch64.whl

A.7. Instalación OpenCV

Con la configuración inicial del JetPack, se encuentra instalada la versión 3.3.1 de OpenCV. Se ha decidido purgar dicha versión, para el correcto funcionamiento de OpenCV con otras librerías. Después de esto se decide la versión a instalar de OpenCV. En algunos foros de NVIDIA recomiendan la versión 4.2.1 de OpenCV, por la posible incompatibilidad de otras versiones con la Raspberry Pi. Sin embargo, en ningún momento se indica que no se puede utilizar la opción de aceleración con CUDA del módulo DNN de OpenCV, idea de la que después de instalar dicha versión fue notable al intentar probar dicha funcionalidad. Esto es de gran interés, ya que si no se dispone de esa opción, no sería correcto medir el rendimiento sólo con CPU, por lo que decidí desinstalar la versión 4.2.1 y probar con una versión que sí dispone de dicha funcionalidad, la versión 4.2.0.

El proceso de instalación de esta librería requiere bastante tiempo, y será desde código fuente:

cd ~

wget -O opencv.zip https://github.com/opencv/opencv/archive/4.2.0.zip

wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.2.0.zip

Las versiones de ambos ficheros tendrán que coincidir para que sean compatibles y poder seguir con el proceso de instalación. Después de esto, se deben descomprimir los ficheros y almacenar su contenido en las carpetas llamadas opencv y opencv_contrib, respectivamente. A partir de aquí, se deberá trabajar sobre el entorno virtual creado anteriormente (py3cv4 en mi caso). Nos situaremos sobre la carpeta opencv, crearemos un directorio llamado build y lo abriremos.

Una vez que estemos en dicha carpeta ((py3cv4) /home/<nombre_usuario>/opencv/build), usaremos CMake como herramienta de preparación de compilación, mediante el siguiente comando:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D WITH_CUDA=ON \-D CUDA_ARCH_PTX= \
-D CUDA_ARCH_BIN="5.3,6.2,7.2"\
-D WITH_CUBLAS=ON \
-D WITH_LIBV4L=ON \
-D BUILD_opencv_python3=ON \
-D BUILD_opencv_python2=OFF \
-D BUILD_opencv_java=OFF \
-D WITH_GSTREAMER=ON \
-D WITH_GTK=ON \
-D BUILD_TESTS=OFF \
-D BUILD_PERF_TESTS=OFF \
-D BUILD_EXAMPLES=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D OPENCV_EXTRA_MODULES_PATH=/home/‘whoami‘/opencv_contrib/modules ..
```

El primer parámetro a destacar es WITH_CUDA=ON, ya que esto significa que se compilará con las optimizaciones que ofrece CUDA. También es de destacar OPENCV_EXTRA_MODULES_PATH, que debe coincidir con la dirección a la carpeta procedente de descomprimir el código fuente. Por último, es importante mencionar que OPENCV_ENABLE_NONFREE=ON, ya que indica que OpenCV se instalará con soporte completo para ciertos algoritmos.

Se prestará atención a la salida de este comando, para evitar errores futuros. Después de revisar dicha salida, se puede ejecutar el comando:

```
make -j4
```

Este proceso de compilación lleva un tiempo aproximado de 2 horas y media. Una vez que llegue al 100 %, se podrá proceder a instalar, mediante la siguiente sentencia:

```
sudo make install
```

A.7.1. Problemas surgidos de la instalación

El único problema que sí que ha surgido durante todo este proceso, llegó cuando llevaba el 99 % la compilación y apareció el mensaje killed por pantalla. Dicho error fue solucionado apagando la Jetson Nano y volviendo a empezar todo este punto.

A.7.2. Enlace con entorno virtual

Para que esta instalación pueda utilizarse con nuestro entorno virtual de Python 3, se debe realizar un link con la imagen de esta librería, ejecutando por consola lo siguiente:

cd ~/.virtualenvs/py3cv4/lib/python3.6/site-packages/

ln -s /usr/local/lib/python3.6/site-packages/cv2/python3.6/cv2.cpython-36m-aarch64-linux-gnu.so cv2.so

Se puede comprobar la instalación correcta utilizando Python, por consola, comprobando dicha versión. Si aparece correctamente ya se pueden a empezar a realizar diversas pruebas utilizando Python 3 con esta librería, como se muestra en la figura A.1.

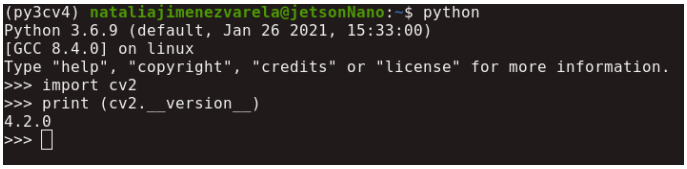

Figura A.1: Comprobación de instalación de la librería OpenCV

A.8. Instalación Qt

Para comenzar, se instaló la versión por defecto de Qt junto a Qt Creator [80], con los siguientes comandos:

sudo apt install qtcreator

sudo apt install qt5-default

Dicha versión fue la 5.9.5. Sin embargo, para contar con funcionalidades más interesantes, como puede ser la clase QColor, era necesaria una versión superior a la 5.14, por lo que decidí desinstalar esta.

Se deben instalar todas las dependencias de la versión que se escoja. Se ha realizado correctamente mediante el comando [81]:

sudo apt-get install '.*libxcb.*' libxrender-dev libxi-dev libfontconfig1-dev libudev-dev

A continuación, se probó a instalar una versión superior a la mencionada anteriormente, por lo que se iba a proceder a instalarlo a partir de código fuente. Sin embargo, no tuvo éxito esta decisión, ya que realizando el configure correspondiente y ejecutando el comando make produjo el error de que no tenía suficiente espacio. Esto es por lo que se decidió cambiar el tamaño de la tarjeta microSD, ampliándolo a 64GB.

Después de cambiar la tarjeta, se tuvieron que realizar todos los pasos anteriormente descritos. A continuación, se volvió a realizar el proceso de nuevo, pero siguió produciendo el mismo error, por lo que se decidió optar por realizar compilación cruzada [82].

Cross-compilation, o compilación cruzada, se produce cuando se compila código para un sistema informático en otro sistema diferente. Esta técnica es de mucha utilidad cuando el dispositivo es demasiado pequeño para almacenar todos los ficheros generados del proceso de compilación y se quiere aumentar la velocidad.

En este caso, se ha decidido optar por la versión 5.15.2 de Qt:

wget http://master.qt.io/archive/qt/5.15/5.15.2/submodules/qtbase-everywhere-src-5.15.2.tar.xz

tar -xpf qtbase-everywhere-src-5.15.2.tar.xz

cd qtbase-everywhere-src-5.15.2/

Una vez que se han ejecutado estos comandos, para realizar compilación cruzada se ejecuta lo siguiente: ./configure -xcb

Después de haber hecho este paso, se realizan los siguientes comandos:

make -j4

sudo make install

Si termina satisfactoriamente este proceso, se habrá instalado en /usr/local/Qt-5.15.2.

Para que esta versión sea funcional, se deben cambiar los ficheros de configuración de versión por defecto de qtchooser [83]. En dicho fichero, se indican las direcciones absolutas a las carpetas bin y lib de la versión instalada, siendo las representadas en la figura A.2.

```
nataliajimenezvarela@jetsonNano:/usr/share/qtchooser$ cat qt5-aarch64-linux-gnu.conf
/usr/local/Qt-5.15.2/bin
/usr/local/Qt-5.15.2/lib
```

Figura A.2: Cambio de configuración de qtchooser con la versión actual de QT

Para operar con OpenCV desde Qt, bastaría añadir la dirección de instalación de OpenCV en el fichero .pro del proyecto correspondiente, junto a las flags necesarias, representadas en la figura A.3.

```
nataliajimenezvarela@jetsonNano:~/getOutputsDnn$ cat test_dnn.pro
QT += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TEMPLATE = app
TARGET = test_dnn
QT *= core

OPENCV = /usr/local
INCLUDEPATH += $$OPENCV/include/opencv4 \

LIBS += -L$$OPENCV/lib -lopencv_highgui -lopencv_core -lopencv_imgproc -lopencv_dnn
INCLUDEPATH *= $$PWD

SOURCES   *=    main.cpp
MOC_DIR = .moc
OBJECTS_DIR = .obj
```

Figura A.3: Fichero .pro del proyecto de QT, incluyendo direcciones de OpenCV

A.9. Instalación FFMPEG

La primera versión que se decide instalar es la de por defecto mediante el siguiente comando:

sudo apt install ffmpeg

Dicha versión es la 3.4.8. El punto negativo de esta instalación es que no cuenta con la posibilidad de la aceleración de la GPU, por lo que decido intentar esto último.

En la página web oficial de NVIDIA se indica que cuenta con la posibilidad de acelerar la parte de decodificación de FFMPEG [84]. Si se tiene la versión instalada anteriormente, se procede con los siguientes comandos:

echo "deb https://repo.download.nvidia.com/jetson/ffmpeg main main"| sudo tee -a /etc/apt/sources.list

echo "deb-src https://repo.download.nvidia.com/jetson/ffmpeg main main"| sudo tee -a /etc/apt/sources.list

sudo apt update

apt source ffmpeg

Estos comandos escritos anteriormente se han obtenido de un foro de NVIDIA, ya que los pasos oficiales en la página oficial están incompletos. Una vez que se haya realizado esto, la versión que está disponible ahora es la 4.2.2.

Existe la posibilidad de acelerar tanto el proceso de codificación como el de decodificación, gracias a un “parche” encontrado en Github (véase [58]). Siguiendo las instrucciones de instalación de dicho repositorio, se notifica un error de que no se ha podido finalizar con éxito dicha instalación.

El error indica que no tengo disponible la multimedia API de Jetson, por lo que la descargo directamente del centro de descargas de Nvidia Jetson [85]. La versión es la correspondiente al LTS del JetPack, es decir, la 32.2.1. Una vez incorporada esta API, ya se puede seguir con el proceso de

instalación.

A.10. Uso de TensorRT

Para utilizar la herramienta TensorRT con Python en nuestro dispositivo Jetson Nano, bastaría con ejecutar el siguiente comando, bien por consola o bien, para que sea permanente, añadiendolo al fichero /.bashr:

```
export PYTHONPATH=/usr/lib/python3.6/dist-packages:$PYTHONPATH
```