eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Computación

Análisis de rendimiento de algoritmos sobre virtualización de GPU Nvidia a través de contenedores Docker y orquestación de Kubernetes

Itziar Irastorza Arriola

Dirección
Iñigo Lopez Gazpio
Roberto Viola

25 de junio de 2023

Agradecimientos

Cuando leí por primera vez el título del que iba a ser mi trabajo de fin de grado, no fue amor a primera vista. No es un tema que he trabajado mucho durante mi carrera y ha sido todo un reto aceptarlo. Después de estos meses de trabajo investigando, probando, analizando... Puedo decir que he superado el objetivo que me propuse en diciembre.

Quiero dar las gracias a Vicomtech, por haberme dado la oportunidad de trabajar con ellos, descubrir el mundo de la investigación y trabajar con un equipo que me ha acogido desde el primer día. Especialmente quiero agradecer a Roberto Viola, mi tutor en la empresa, por haber dirigido mi proyecto. También quiero mencionar a Iñigo Lopez, mi tutor en la universidad, por confiar en mí y en mi proyecto.

Resumen

La computación de alto rendimiento se ha vuelto imprescindible, ya que es la base de cualquier investigación científica. El crecimiento masivo de esta área ha traído ciertos problemas en la computación, como el crecimiento de la cantidad de datos en la computación. Para ello, existen ciertas soluciones para ayudar a facilitar y aligerar la computación en estos casos. Este proyecto se centrará en el estudio de los algoritmos de alto rendimiento, como los de detección y seguimiento de objetos en tiempo real, cuando ejecutamos en un entorno virtualizado y acelerado por la GPU.

En primer lugar, se analizará e investigará sobre los algoritmos de alto rendimiento, en concreto centrándose en los de detección y seguimiento de los objetos. También se estudiará el tema de la aceleración por GPU, como funciona y en que beneficia a la hora de la computación. Otro de los temas en el que se adentrará en el proyecto será la virtualización y las técnicas que existen para conseguir esta herramienta. Se conocerá la técnica de la virtualización a través de contenedores y se preparará un entorno virtual utilizando esa técnica

Finalmente, en el experimento se ejecutarán varios algoritmos en ese entorno virtual haciendo uso de la GPU y se hará un análisis de los resultados. Habrá una comparación de distintos casos en el que se sacarán conclusiones.

Índice de contenidos

Agradecimientos v

Resumen v

Índice de contenidos v

1 Introducción 1
- 1.1. Motivación . . . 1
- 1.2. Estructura del documento . . . 1

2 Objetivos del proyecto 3
- 2.1. Objetivos . . . 3
- 2.2. Plan de trabajo . . . 4

3 Estado del arte 7
- 3.1. Contexto . . . 7
- 3.2. Computación de alto rendimiento . . . 7
  - 3.2.1. Detección y seguimiento de objetos . . . 8
- 3.3. Aceleración por GPU para computación de alto rendimiento . . . 9
  - 3.3.1. GPU . . . 9
  - 3.3.2. Aceleración . . . 9
  - 3.3.3. Nvidia . . . 10
- 3.4. Virtualización para computación de alto rendimiento . . . 10
  - 3.4.1. Definición . . . 10
  - 3.4.2. Técnicas de virtualización . . . 11
  - 3.4.3. Virtualización a través de contenedores . . . 13

4 Desarrollo del proyecto 15
- 4.1. Preparación del entorno de trabajo . . . 15
  - 4.1.1. Nvidia Drivers . . . 15
  - 4.1.2. Nvidia Container Toolkit . . . 16
  - 4.1.3. Kubernetes Cluster local . . . 16
- 4.2. Creación del contenedor Docker . . . 17
- 4.3. Creación del Helm Chart . . . 18
- 4.4. Ejecución de algoritmos . . . 19
  - 4.4.1. Nvidia DeepStream . . . 19

5 Análisis de los resultados 23

5.1. Análisis de rendimiento de algoritmos 24
5.1.1. Metodología 24
5.1.2. Resultados 25
5.2. Análisis de rendimiento de la GPU 27
5.2.1. Metodología 27
5.2.2. Métricas 32

6 Conclusión **35**
6.1. Producto 35
6.2. Conclusión 36
6.3. Mejoras en el futuro 37

Apéndice **39**
Apéndice 1 40

Bibliografía **41**

CAPÍTULO 1

Introducción

1.1. Motivación

La computación de algoritmos de alto rendimiento, HPC *(High Performance Computing)*, es la capacidad de procesar datos y realizar cálculos a velocidades muy altas. Hoy en día, se ha vuelto imprescindible, ya que es la base de los avances científicos, industriales y sociales. Machine Learning se ha convertido en una herramienta esencial para extraer información de grandes conjuntos de datos. La evolución de las tecnologías IoT, la Inteligencia Artificial y las imágenes en 3D ha tenido como consecuencia el crecimiento exponencial del tamaño y la cantidad de los datos.

En este proyecto analizaremos las posibles soluciones a los problemas de la excesiva subida de la cantidad de datos en este mundo. De las soluciones existentes, nos centraremos en la aceleración por GPU y la virtualización. Este análisis se llevará a cabo ejecutando unos algoritmos de detección y seguimiento de objetos ya implementados por Nvidia haciendo uso de la GPU. Las distintas ejecuciones se harán en un entorno virtualizado, haciendo uso de contenedores para ver el rendimiento de estos en este nuevo entorno. Principalmente, se harán dos análisis, cada uno con relacionado con un objetivo principal. El primer análisis será sobre rendimiento de esos algoritmos de Nvidia. Obtendremos los datos de exactitud, precisión, sensibilidad y el valor F1. El segundo análisis será de la GPU misma, se sacarán los datos sobre la utilización, temperatura, la memoria y la energía consumida.

Para ello, se preparará el entorno virtual y se instalarán todas las dependencias. Se investigará sobre los algoritmos de detección y seguimiento de objetos de Nvidia que se ubican en los contenedores llamados DeepStream. Se hará uso de distintas herramientas de monitorización para el análisis, como Prometheus y Grafana, y las posibilidades que ofrecen para alcanzar los objetivos.

1.2. Estructura del documento

Para empezar se hará una breve introducción al proyecto en la primera sección. Se describirán los objetivos principales y los secundarios de este proyecto en la segunda sección. También como el proyecto ha ido transcurriendo en el periodo de realización y el

plan de trabajo. A continuación, en el tercer apartado, se hablará sobre el estado del arte para situar el tema con los problemas y logros en la actualidad para poner en contexto. Este apartado se centrará mayormente en los algoritmos para detección y seguimiento de objetos (algoritmos de alto rendimiento), la aceleración por GPU y la virtualización a través de las herramientas Docker y Kubernetes.

Una vez situados en el proyecto y en el tema, se seguirá con la parte principal que es el desarrollo del proyecto en el cuarto punto. Como inicio al desarrollo, se explicarán los pasos para la preparación del entorno de trabajo, claves para dar inicio al proyecto. Los controladores de NVIDIA, NVIDIA-Docker y el clúster de Kubernetes desplegado en la máquina serán los puntos principales de este apartado. Seguidamente, se centrará en la ejecución de distintos algoritmos en distintos casos (en este caso, distintos vídeos). Y para finalizar con este apartado, desarrollo del proyecto, se mostrarán las métricas explicando cada una de ellas, haciendo una conclusión de los resultados obtenidos.

A continuación, se hará un análisis de los resultados que se dividirá en dos partes. Por un lado, habrá un análisis de los algoritmos ejecutados y por otro se analizará el rendimiento de la GPU. Por último y como sexto apartado, se hará una conclusión del proyecto y las mejoras para el futuro.

CAPÍTULO 2

Objetivos del proyecto

En este capítulo del proyecto se indicarán los objetivos del proyecto y el plan de trabajo que se ha llevado a cabo para diseñar, construir y analizar el proyecto.

2.1. Objetivos

Al inicio del proyecto se han definido ciertos objetivos principales. Uno de los objetivos principales es **investigar las posibles soluciones a los problemas que se encuentran al ejecutar algoritmos de alto rendimiento**. La necesidad de una gran cantidad de datos hacen que se necesiten más recursos computacionales, ya que el tiempo de ejecución y la complejidad del algoritmo aumentan significativamente, lo que afecta al rendimiento general. Dentro de este objetivo principal, se encontrarán estos sub objetivos.

- Investigar la técnica de aceleración por GPU

  Se investigará y estudiará que es y para que sirve una GPU, como funciona y que ventajas tiene en comparación con una CPU. También se definirá lo que supone la computación por GPU, sus ventajas y sus desventajas. Se indagará en las soluciones dadas por Nvidia para usar esta herramienta.

- Estudiar la técnica de la virtualización del entorno

  Se estudiará que es y en que se basa la herramienta de la virtualización del entorno y en que beneficia al problema planteado al inicio del proyecto. Se podrán observar las diferentes técnicas de virtualización y las ventajas y desventajas de cada uno de ellos. Finalmente, este sub objetivo se centrará en la técnica de virtualización a través de contenedores.

Otro de los principales objetivos es **analizar el rendimiento de los algoritmos y de la GPU cuando se ejecutan algoritmos de alto rendimiento en un entorno virtualizado**, en nuestro caso por contenedores manejados por Kubernetes. Se hará un experimento con las herramientas necesarias para obtener resultados y sacar conclusiones. Este segundo objetivo principal se dividirá en varios sub objetivos.

- Preparar el entorno de trabajo para el experimento

  En esta fase se instalará el software necesario en la máquina y se configurará para poder ejecutar ciertas aplicaciones y para poder monitorizar en la fase final. Se desplegará un clúster local de Kubernetes donde se ejecutarán las aplicaciones.

- Elección de algoritmos de alto rendimiento para comparaciones

  Se hará una elección de diferentes algoritmos de distintos rendimientos para ejecutarlos y comparar el rendimiento de todos ellos. Los algoritmos tendrán diferentes niveles de peso en cuanto al rendimiento. También se elegirán distintos vídeos para hacer las detecciones, vídeos que tengan muchos vehículos por detectar y vídeos que no.

- Ejecución de los algoritmos

  Se diseñará una manera automatizada de ejecutar dichos algoritmos en el entorno virtual preparado anteriormente, para poder cambiar fácilmente de una aplicación a otra y para poder cambiar el vídeo que va a ser analizado. La ejecución se hará en el clúster local de Kubernetes.

- Análisis y monitorización

  Hacer el estudio y la monitorización de las ejecuciones será el último sub objetivo. Se dividirá en dos partes, por un lado, *el análisis de rendimiento de los algoritmos* y por otro lado, *el análisis de los datos de rendimiento de la GPU* obtenidos por en la monitorización.

2.2. Plan de trabajo

El plan de trabajo de este proyecto se ha definido entre el tutor de la empresa y la estudiante. Se ha hecho una reunión por semana de manera presencial para hacer el seguimiento por parte del tutor. Aparte de esa reunión se ha hecho una reunión de departamento una vez por semana y se han compartido los logros de cada semana a los otros miembros del departamento. Al inicio del proyecto se han descrito ciertas actividades para dar comienzo al proyecto. Estas tareas han servido para investigar y entender lo que está estudiado hoy en día sobre este tema.

1. Investigar sobre los recursos de Nvidia respecto a los algoritmos de alto rendimiento.
   a) DeepStream 6.0, Aplicaciones para detección y seguimiento de objetos.
2. Investigar sobre la aceleración por GPU.
   a) ¿Que es? ¿En que se basa?
   b) Investigar sobre los recursos de la máquina.
      1) ¿Que máquina es? ¿Que gráfica tiene?
      2) Como hacer uso de ella
3. Investigar sobre la virtualización
   a) Diferentes técnicas

   b) Virtualización a través de contenedores
   c) Docker y Kubernetes
      1) Cursos de Docker y Kubernetes

4. Instalar recursos necesarios para el desarrollo del experimento
   a) Nvidia Drivers
   b) Cluster local de Kubernetes
   c) Despliegue de herramientas de monitorización

Después de haber profundizado en la parte técnica y haber instalado y desplegado las herramientas necesarias para preparar el entorno de trabajo, se ha definido como ejecutar las aplicaciones de una forma automática haciendo uso de contenedores y desplegándolas a través de Kubernetes. Estas son las actividades que se han definido en esta parte del proyecto entre el tutor y la estudiante:

1. Crear un contenedor de Docker cogiendo como base la imagen de Nvidia DeepStream
   a) Preparar Dockerfile para configurar el contenedor
      1) Instalar dependencias
      2) Copiar los ficheros de configuración para ejecutar distintos algoritmos con distintos vídeos (9 ficheros en total)
      3) Comando para ejecutar la aplicación
2. Implementar Helm Chart para desplegar el contenedor anteriormente creado

Se ha definido un diagrama de Gantt para establecer las tareas en el tiempo. Este diagrama se encuentra en el apartado Apéndice 16.3 de la memoria.

CAPÍTULO 3

Estado del arte

3.1. Contexto

Hoy en día el peso de los algoritmos de Machine Learning ha ido creciendo exponencialmente. El uso de los algoritmos de detección y seguimiento de objetos se está volviendo imprescindible en muchos ámbitos. Se podría remarcar que en el ámbito de la conducción es donde más se usan este tipo de herramientas. Estos tipos de cálculos requieren mucha potencia computacional y por ello supone un problema para implementarlos cuando se tienen ciertos medios para hacerlo.

Como solución a este problema, existe la *Virtualización*. Este fenómeno permite organizar y dividir la computadora física en muchas máquinas virtuales, pudiendo así ejecutar más de un sistema operativo, aplicaciones y sistemas virtuales en un solo servidor. Actualmente, existen herramientas de virtualización tal y como *Hipervisores*, *Contenedores* y más. De estas herramientas, este proyecto se centrará en la virtualización a través de Docker y Kubernetes, esto es, virtualización a través de contenedores.

3.2. Computación de alto rendimiento

La computación de alto rendimiento o HPC (High Performance Computing) se refiere al uso de sistemas de computación y recursos especializados para realizar tareas de procesamiento intensivo de datos y cálculos complejos a una escala mucho mayor y con un rendimiento superior al de los sistemas informáticos convencionales. Se utilizan sistemas de hardware y software altamente optimizados para ejecutar aplicaciones que requieren un gran poder de procesamiento, memoria y almacenamiento. Estos están diseñados para realizar operaciones en paralelo, es decir, realizar múltiples tareas simultáneamente, utilizando múltiples procesadores o núcleos de procesamiento, y a menudo se basan en arquitecturas de clústeres o supercomputadoras.

El objetivo principal de la computación de alto rendimiento es resolver problemas complejos y demandantes en campos como la simulación numérica, la modelización y la simulación de sistemas físicos, la predicción del clima y el tiempo, la investigación científica, la bioinformática, la inteligencia artificial y el aprendizaje automático, entre otros.

Los sistemas de computación de alto rendimiento están diseñados para escalar de manera eficiente y procesar grandes volúmenes de datos en tiempos razonables. Esto implica la utilización de técnicas como la paralelización, la distribución de carga y la optimización de algoritmos para aprovechar al máximo los recursos disponibles. Cuentan con una alta capacidad de procesamiento, que se logra mediante el uso de múltiples núcleos de procesamiento, procesadores de alto rendimiento (como CPUs y GPUs) y arquitecturas de memoria optimizadas. También tienen una gran capacidad de almacenamiento para manejar grandes volúmenes de datos generados por las aplicaciones de alto rendimiento. Esto puede incluir sistemas de almacenamiento en disco, sistemas de archivos distribuidos y tecnologías de almacenamiento en la nube.

3.2.1. Detección y seguimiento de objetos

Detección y seguimiento de objetos en tiempo real[1] son tareas importantes y muy complejas en muchas aplicaciones de visión por ordenador, tal y como la videovigilancia, la navegación robótica y la navegación de vehículos. El mecanismo de seguimiento de objetos requiere un mecanismo de detección de objeto[2]. El seguimiento de objetos es el proceso de localizar un objeto o varios objetos utilizando una cámara (estática, semi-estática o dinámica). En el campo de la visión por computador, la detección y seguimiento de objetos son una parte fundamental. Estas aplicaciones significan localizar/identificar objetos en una secuencia de vídeo y técnicamente el seguimiento consiste en estimar la trayectoria de un objeto en la imagen.

Estas tareas parecen unas tareas fáciles, pero la detección de objetos mediante *Algoritmos de Machine Learning* implica una red neuronal convencional[3] detectando una cantidad limitada de objetos. No puede detectar objetos que antes no hubiera visto, si son de tamaños muy pequeños también tiene dificultades, dificultades de posibles "focos", sombras y el poder de determinar en qué posición se encuentra. Existen diferentes modelos para llevar a cabo estos algoritmos y a continuación se explicarán tres de ellos, considerados más usados.

Las redes neuronales convolucionales basadas en regiones (R-CNN) tratan de extraer las funciones más esenciales (generalmente alrededor de 2000 funciones) haciendo uso de funciones selectivas. El proceso de las extracciones más significativas se pueden computar con la ayuda de un algoritmo de búsqueda selectiva que puede lograr estas propuestas regionales más importantes. El primer paso de este modelo es el trabajo del algoritmo de búsqueda selectiva para seleccionar las propuestas regionales más importantes es asegurarse de generar múltiples sub segmentaciones en una imagen en particular y seleccionar las entradas candidatas para su tarea. Luego se usa un algoritmo para un proceso recurrente de combinación de los segmentos más pequeños para obtener segmentos más grandes. Una vez que el algoritmo de búsqueda selectiva se completa con éxito, se extraen las características y se hacen las predicciones apropiadas para las propuestas candidatas finales. El paso final del R-CNN es hacer las predicciones apropiadas para la imagen y etiquetar el cuadro delimitador respectivo en consecuencia.

R-CNN rápido o *Fast R-CNN*[4] es una red neuronal convolucional profunda que aparece para el usuario como una única red unificada de extremo a extremo. Este es más rápido que el R-CNN, pero lamentablemente descuida cómo se generan las propuestas de la región. Esta red coge como entrada la imagen entera y un conjunto de proposiciones de objetos. La red procesa toda la imagen con varias convoluciones primero. A continuación, para

cada propuesta de objeto, establece una capa de agrupación de regiones de interés y extrae un vector de características de longitud fija del mapa de características. Finalmente, cada vector de características se introduce en una secuencia de capas totalmente conectadas que se ramifican en dos capas de salida: una que produce estimaciones de probabilidad sobre objetos de una clase más de "fondoz otra capa que produce cuatro números de valor real para cada una de las K clases de objetos.

3.3. Aceleración por GPU para computación de alto rendimiento

3.3.1. GPU

La unidad de procesamiento de gráficos o GPU, se ha convertido en una de las herramientas de más importantes en el mundo de la informática. Diseñada para el procesamiento en paralelo, la GPU se utiliza en una amplia gama de aplicaciones, incluida la representación de gráficos y vídeos. Las GPU se están volviendo más populares para su uso en la producción creativa y la inteligencia artificial. Se diseñó originalmente para acelerar la representación de gráficos 3D y con el tiempo se volvieron más flexibles y programables mejorando sus capacidades. Los desarrolladores aprovechan el poder de esta herramienta para acelerar las cargas de trabajo adicionales en computación de alto rendimiento.

El funcionamiento de esta herramienta es la siguiente. El procesador de la computadora calcula qué datos quiere mostrar en la pantalla y los emite como los llamados datos de imagen. En este momento, la GPU convierte todos esos datos, la mayoría a numéricos, para que puedan mostrarse en un monitor u otro dispositivo. Tiene un chip de gráficos como cerebro del sistema. Este chip realiza los cálculos de las imágenes, mientras que el resto de los componentes se ocupan en gran medida de equipar al máximo el chip gráfico para que pueda funcionar a pleno rendimiento y no tenga que esperar a la memoria, por ejemplo.

La CPU es un procesador de uso general que realiza una amplia gama de tareas, mientras que las GPU, son procesadores especializados que están diseñados para manejar tareas de procesamiento de imágenes y gráficos. La CPU maneja los procesos del sistema y hace la gestión de operaciones de entrada/salida. Por otro lado, la GPU está capacitado para el procesamiento en paralelo y puede manejar múltiples tareas simultáneamente. Tiene una gran cantidad de núcleos y los utiliza para realizar cálculos de manera rápida y eficiente.

3.3.2. Aceleración

La computación por GPU[5] se ha convertido en la base para optimizar el aprendizaje profundo, aumentar la velocidad de procesamiento durante la codificación, mejorar la gestión de datos, crear contenido y brindar una visión integral del análisis de datos. Este proceso se lleva a cabo a través de una computación paralela. La GPU interviene y separa los problemas complejos en millones de tareas cuando la CPU se satura con el procesamiento de grandes volúmenes de datos. Tanto la CPU como la GPU pueden trabajar juntas en un ecosistema de inteligencia artificial. El uso de la computación GPU beneficia a la CPU interna, permitiendo procesar y renderizar a un ritmo acelerado. Aunque las CPU tienen velocidades de procesamiento mucho más altas, las GPU tienen capacidades de procesamiento inigualables gracias al paralelismo.

Como ya se ha mencionado anteriormente, con la GPU actuando como un procesador complementario de las CPU, aumentan exponencialmente la velocidad y las capacidades de procesamiento de un sistema. Además, las GPU reducen la carga de la CPU al procesar datos repetitivos en fragmentos más pequeños en varios procesadores. Asimismo, las GPU amplían el ancho de banda de la memoria y trabajando cientos de veces más rápido que las CPU, las GPU hacen posible la automatización y la inteligencia del aprendizaje automático y el análisis del Big Data, ya que procesan cantidades masivas de datos a través de redes neuronales.

El enfoque de TI ha cambiado dado que las demandas informáticas de la Inteligencia Artificial y la ciencia de datos requieren unos recursos computacionales más potentes. Este trabajo lo realizan las GPU. Las aplicaciones que se ejecutan en CPU se aceleran mediante el procesamiento de GPU, lo que optimiza el rendimiento y la capacidad de carga de trabajo. Las GPU procesan miles de tareas en segundos a través de sus cientos de núcleos a través del procesamiento paralelo. El procesamiento paralelo denota una función en la que los conjuntos de datos se canalizan a los núcleos de procesamiento de una GPU y se resuelven todos simultáneamente. El rendimiento aumenta a medida que la GPU procesa y traduce datos mientras la CPU ejecuta las aplicaciones restantes.

3.3.3. Nvidia

Los chips de gráficos comenzaron como canales de gráficos de función fija. Con el paso de los años, estos chips gráficos se volvieron cada vez más programables, lo que llevó a Nvidia a presentar la primera GPU. En el periodo de tiempo 1999-2000, los científicos informáticos, junto con los investigadores en campos como la imagen médica y el electromagnetismo, comenzaron a usar la unidad de procesamiento gráfico para acelerar una variedad de aplicaciones científicas. Este fue el advenimiento del movimiento llamado GPGPU, o computación GPU de Propósito General. Nvidia se dio cuenta del potencial de llevar este rendimiento a la comunidad científica en general e invirtió en la modificación de la GPU para hacerla completamente programable para aplicaciones científicas. Además, agregó soporte para lenguajes de alto nivel como C, C++ y Fortran. Esto condujo a la plataforma de computación paralela CUDA para la GPU.

3.4. Virtualización para computación de alto rendimiento

3.4.1. Definición

La virtualización ha sido una parte muy importante desde hace casi medio siglo. En las décadas 1960 y 1970, IBM desarrolló el Control Program/Cambridge Monitor System (CP/CMS) que condujo a VM/370. Estos sistemas permitían que cada usuario ejecutara lo que parecía ser un sistema aislado, pero todo dentro de un entorno informático de tiempo compartido. La virtualización a nivel de idioma se introdujo alrededor de la década de 1980 para admitir la portabilidad y el aislamiento a nivel de aplicación. La máquina virtual de Java, presentada por Sun en la década de 1990, se encontraba en una posición única: al comienzo de la WWW (World Wide Web), ofrecía a los desarrolladores la oportunidad de agregar contenido ejecutable a la Web de manera portátil y segura. Aunque mencionamos las máquinas virtuales cuando hablamos sobre virtualización, otros incluyen el uso compartido de escritorio, redes virtuales, almacenamiento virtual y mucho más.

3.4.2. Técnicas de virtualización

Hoy en día hay diferentes técnicas[6] de virtualización en el mercado. Existe la virtualización del sistema operativo invitado, la virtualización de núcleo compartido, la virtualización a nivel de kernel y la virtualización de hipervisor.

La *Virtualización del sistema operativo invitado* es la forma más simple y fácil de hacer virtualización según Educba [7]. En esta virtualización, está presente un sistema operativo host y el software de virtualización está instalado en ese sistema operativo host. El sistema operativo host puede ser Windows, como Mac o Linux, y el software de virtualización se ejecutara como cualquier otra aplicación en el sistema operativo. Este software se encarga de todas las tareas de virtualización y ayuda a ejecutar el sistema operativo invitado. Uno puede ejecutar múltiples sistemas operativos usando ese software de virtualización que se encarga de todo como la gestión de la memoria, la gestión de recursos, la partición del disco duro...

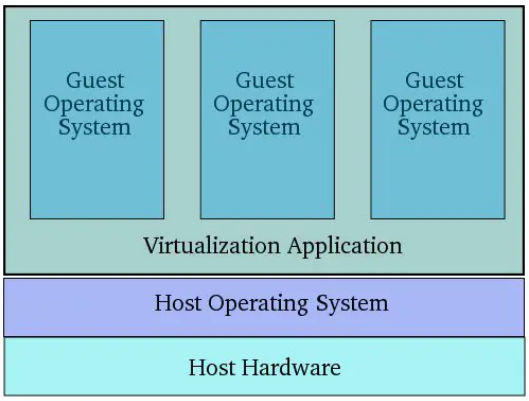


Figura 3.1: Virtualización del sistema operativo invitado

La *Virtualización de núcleo compartido* no se requiere una configuración adicional para el sistema operativo y hardware del host. La aplicación de virtualización utiliza el mismo hardware que el sistema operativo host y se ejecuta dentro de este. La ventaja en este caso es que no hay un costo adicional para la configuración del hardware y se requiere la configuración del software. Es la forma más económica de virtualización, aunque esto signifique problemas de rendimiento debido a los altos niveles de abstracción. VMWare y VirtualBox son el software utilizado para esta virtualización.

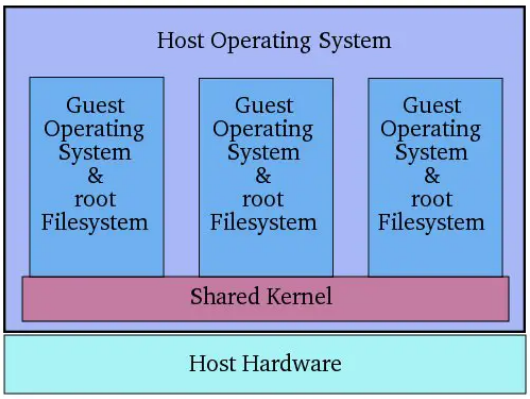


Figura 3.2: Virtualización de núcleo compartido

La *Virtualización a nivel de kernel* es cuando el sistema operativo invitado ejecuta su kernel, a diferencia de la virtualización de kernel compartida. Puede haber varios sistemas operativos, pero cada uno con su kernel. El Kernel del sistema operativo invitado debe tener una configuración similar a la del host, de lo contrario habrá problemas de compatibilidad. Este tipo de virtualización incluye Linux en modo usuario y máquinas virtuales basadas en kernel.

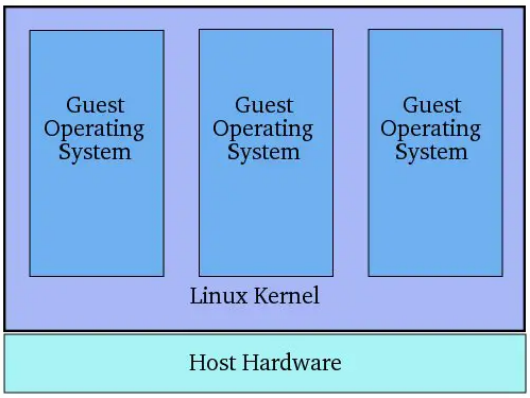


Figura 3.3: Virtualización a nivel de kernel

La *Virtualización de hipervisor*[8] se basa en un programa llamado hipervisor que se ejecuta directamente en el hardware de la CPU, que generalmente se denomina anillo 0, que es el nivel más alto de privilegios que brinda el hardware de la CPU a cualquier software. Generalmente, el sistema operativo solo tiene los privilegios para ejecutarse en el anillo 0 por lo que el hipervisor se ejecuta en él. Como el nombre define, este programa supervisa todos los sistemas operativos invitados instalados en la máquina virtual y proporciona interfaces para administración y supervisión de nivel superior.

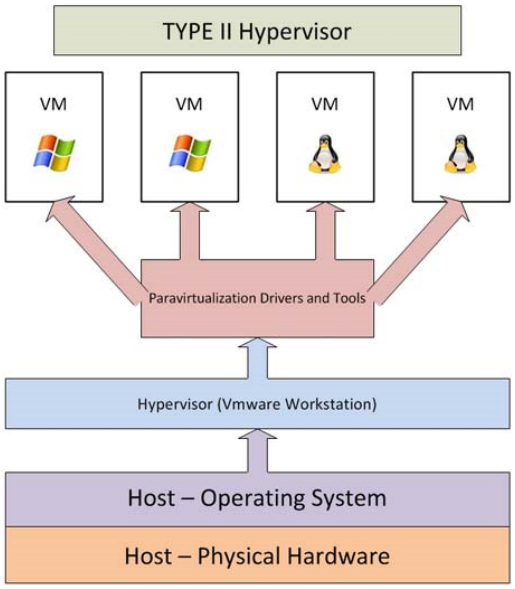


Figura 3.4: Virtualización de hipervisor

3.4.3. Virtualización a través de contenedores

En este proyecto usaremos la técnica de la virtualización a través de contenedores para el experimento. Se ha preparado un contenedor Docker para almacenar y preparar las aplicaciones que se ejecutarán para el análisis y se instalará un clúster local de Kubernetes para administrar este contenedor en un entorno virtual. A continuación se explicarán estas herramientas.

3.4.3.1. Contenedores Docker

Los contenedores[9] son un tipo de software que puede empaquetar y aislar virtualmente aplicaciones para su implementación. Los contenedores pueden compartir el acceso a un kernel de sistema operativo sin la necesidad tradicional de máquinas virtuales. La tecnología de contenedores empezó en la década de 1970. La contenedorización de aplicaciones, Docker, y contenedorización de sistemas, como LXC (Linux Containers), permiten que un equipo de TI abstraiga el código de la aplicación de la infraestructura subyacente, lo que simplifica la administración de versiones y permite la portabilidad en varios entornos de implementación.

Las imágenes de contenedor incluyen la información que se ejecuta en tiempo de ejecución en el sistema operativo, a través de un motor de contenedor. Las aplicaciones en contenedores pueden estar compuestas por varias imágenes de contenedores. Por ejemplo, una aplicación de tres niveles puede estar compuesto por varios servidores y contenedores de bases de datos, cada uno de los cuales se ejecuta de forma independiente. Varias instancias de una imagen de contenedor pueden ejecutarse simultáneamente, y las nuevas instancias pueden remplazar las fallidas sin interrumpir el funcionamiento de la aplicación. Estos contenedores son usados durante el desarrollo y las pruebas y cada vez más, los equipos de operaciones de TI implementan entornos de TI de producción en vivo en contenedores, que pueden ejecutarse en servidores sin sistema operativo, en máquinas virtuales y en la nube.

Estos contenedores tienen los componentes necesarios para ejecutar el software deseado. Estos componentes incluyen archivos, variables de entorno, dependencias y bibliotecas. El sistema operativo host restringe el acceso del contenedor a los recursos físicos, como la CPU, el almacenamiento y la memoria, por lo que un solo contenedor no puede consumir todos los recursos físicos de un host. Las imágenes de Docker se componen de varias capas, que comienzan con una imagen base que incluye todas las dependencias necesarias para ejecutar código en un contenedor.

3.4.3.2. Kubernetes

El uso de los contenedores es una buena forma de desplegar y ejecutar las aplicaciones, pero en un entorno de producción se necesita administrar los contenedores que ejecutan las aplicaciones y asegurarse de que no haya tiempo de inactividad, por ejemplo cuando un contenedor cae, otro contenedor tiene que comenzar. Kubernetes[10] se ocupa de la escalabilidad y la recuperación automática de su aplicación, además de ofrecer pautas de implementación y otras funcionalidades adicionales.

Kubernetes pueden exponer un contenedor usando el nombre DNS o usando su propia dirección IP y si el tráfico a un contenedor es alto, puede equilibrar la carga y distribuir el tráfico. También permite montar automáticamente un sistema de almacenamiento de elección, como almacenamiento local. Puede describir el estado deseado para sus contenedores implementados utilizando Kubernetes, y puede cambiar el estado real al estado deseado a un ritmo controlado. Le proporciona a Kubernetes un grupo de nodos que puede usar para ejecutar tareas en contenedores, y puedes decidir cuanta CPU y memoria (RAM) necesita cada contenedor, pudiendo así aprobechar al máximo los recursos de sus nodos mediante la colocación eficiente de contenedroes. Por último, Kubernetes reinicia los contenedores que fallan y remplaza los contenedores, elimina los contenedores que no responden a su verificación de estado definida por el usuario y no los presenta a los clientes hasta que estén preparados para ser servidos.

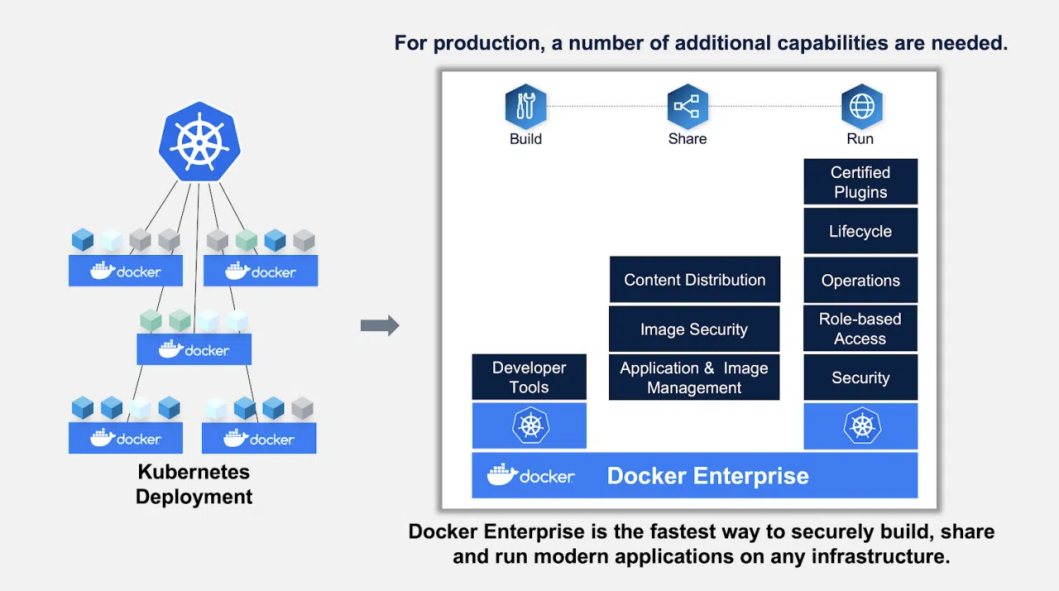


Figura 3.5: Estructura virtualización a través de contenedores

CAPÍTULO 4

Desarrollo del proyecto

4.1. Preparación del entorno de trabajo

El primer paso para llevar a cabo el proyecto ha sido preparar el entorno de trabajo. Se han descargado los controladores de Nvidia para que el sistema operativo y la aplicación de software utilicen el hardware de gráficos del ordenador. También han instalado los componentes de Docker necesarios para desplegar el clúster de Kubernetes. Lo siguiente antes mencionado ha sido desplegar el clúster de Kubernetes, instalando el software necesario para hacer uso de ello. Por último, se han desplegado los contenedores necesarios para recoger las métricas de la GPU y esto se ha llevado a cabo a través de Kubernetes.

4.1.1. Nvidia Drivers

Nvidia diseña unidades de procesamiento de gráficos para los mercados de videojuegos y profesionales, así como sistema en unidades de chip (SoC) para el mercado de computación móvil y automotriz. Su línea principal de productos, GeForce, es la principal competencia directa con los productos Radeon de AMD. Además de la fabricación de GPU, Nvidia proporciona en todo el mundo capacidades de procesamiento en paralelo a investigadores y científicos, que les permiten ejecutar de manera eficiente aplicaciones de alto rendimiento. La GPU donde se ha trabajado ha sido la siguiente: **Nvidia GeForce GTX 1080 6GB**. Tal y como se ha mencionado en la introducción, se han instalado los controladores de Nvidia para que la aplicación que se va a ejecutar haga uso de la GPU. Los controladores han sido instalados de la página web oficial de Nvidia[11], siguiendo los pasos indicados.

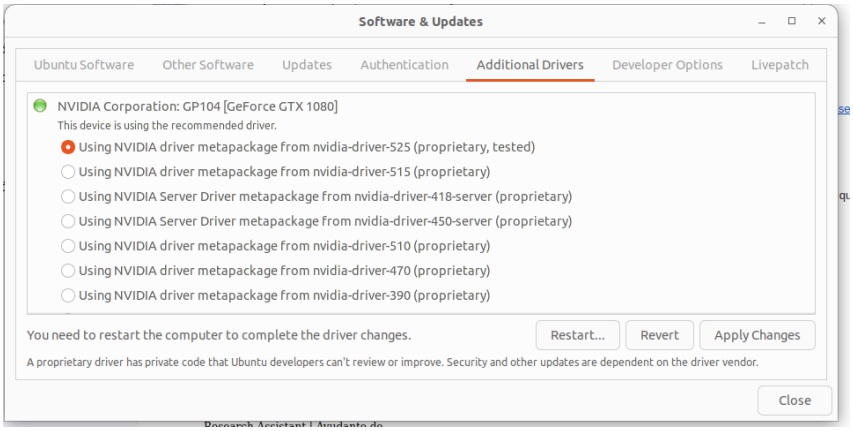


Figura 4.1: Drivers instalados en la máquina

4.1.2. Nvidia Container Toolkit

La instalación de *Nvidia Container Toolkit*[12] ha sido esencial, ya que permite utilizar la potencia de las GPU para el trabajo. Para poder funcionar con todas las características de Docker al tener GPU de Nvidia, la solución es la de tener imágenes con el driver de Nvidia, de tal forma que cuando se inicie el contenedor virtual, también se monte automáticamente todo lo necesario para funcionar.

Uno de los requisitos para poder instalar esta herramienta es que el sistema operativo Linux debe estar basado en arquitectura x64 y tiene que tener un Kernel superior a la versión 3.10. Además, es necesario tener una versión de Docker 1.9 o superior y tener una GPU Nvidia con arquitectura Fermi (2.1) o superior. Por último, debemos utilizar los drivers de Nvidia 340.29 o superior. Estos requisitos se han obtenido de la página web oficial de Nvidia.

4.1.3. Kubernetes Cluster local

Este proyecto se ha llevado a cabo a través de un clúster local de Kubernetes. Se han desplegado los contenedores necesarios para la ejecución de los algoritmos de detección y seguimiento de objetos y también los contenedores necesarios para recoger las métricas de la GPU (Prometheus y Grafana). La instalación se ha llevado a cabo a través de un script y estos son los recursos que se han instalado: Docker, containerd, kubectl, kubeadm, kubelet y Helm.

Kubernetes es una plataforma de automatización para administrar aplicaciones de diversos contenedores individuales dentro de múltiples cluústeres. Esta plataforma presenta el patrón de implementación de las aplicaciones y administra las conmutaciones por error y las copias de seguridad. El enfoque principal es facilitar la implementación y la administración sin cancelar los beneficios que permiten los contenedores. Se pueden compilar, ejecutar y escalar las aplicaciones en contenedores más rápido.

Una de las ventajas de esta herramienta es que si hay algún error en alguno de los pods, la herramienta misma hace que vuelva a reintentar. Tiene la posibilidad de monitorización. Esto es, hay servicios ya disponibles que al desplegarlos en el propio clúster, puedes monitorizar ciertos aspectos de la máquina. Por ejemplo, en este proyecto, hemos desplegado los

servicios **Prometheus y Grafana** para la monitorización. Estos recursos de monitorización se explicarán en la sección **Resultados**.

4.2. Creación del contenedor Docker

Para ejecutar las aplicaciones de Nvidia ya implementadas, se ha creado un contenedor de Docker donde se ha cogido como base una imagen de Nvidia que es la siguiente: **nvcr.io/nvidia/deepstream:6.0.1-samples**[13]. Se ha implementado un Dockerfile así pudiendo hacer las modificaciones necesarias en la imagen para poder automatizar las ejecuciones de manera fácil y efectiva.

Dockerfile[14] es un script que está compuesto por varios comandos y argumentos. Estos comandos y argumentos son los que se van a ejecutar sucesivamente y automáticamente para realizar una acción en una imagen base o crear una nueva. Es decir, se puede partir desde una imagen ya creada anteriormente y ejecutar ciertas instrucciones en ella o se puede crear directamente la imagen desde cero.

En este caso, como se ha mencionado anteriormente, se ha partido desde una imagen creada por Nvidia y se han ejecutado ciertas instrucciones para moldearlo al proyecto. Estas son las acciones que se han utilizado en el Dockerfile.

1. RUN - Instalar dependencias necesarias para la ejecución de las aplicaciones
2. ADD - Crear ficheros nuevos de configuración vacios
3. COPY - Copiar nuestros ficheros de configuración ya cambiados desde local al contenedor
4. CMD - Comando para ejecutar la aplicación

Los ficheros de configuración insertados en la imagen son los ficheros que determinan qué algoritmo ejecutar con que vídeo. Se ha obtenido el fichero de configuración que hay por defecto en la imagen de Nvidia y se han hecho 9 copias. Cada fichero es de un cierto vídeo para ejecutar con cierto algoritmo. El primero, por ejemplo, es para ejecutar el primer vídeo con el primer algoritmo. Como se han ejecutado tres algoritmos y cada uno con tres vídeos distintos, son 9 los distintos ficheros de configuración.

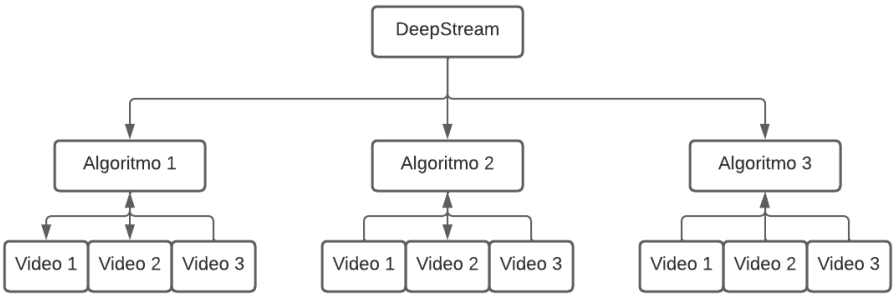


Figura 4.2: Diagrama de las 9 aplicaciones

4.3. Creación del Helm Chart

El despliegue de la aplicación en el clúster local de Kubernetes se ha llevado a cabo a través de una herramienta llamada **Helm Chart**. Los Helm Charts son utilizados para la gestión y el despliegue de los contenedores/aplicaciones en Kubernetes. Es un paquete pre configurado que contiene todos los recursos necesarios para desplegar una aplicación. Tiene archivos YAML donde se definen los pods, servicios, volúmenes y más, necesarias para el despliegue de las aplicaciones.

El Helm Chart para la ejecución de las aplicaciones contiene los siguientes elementos. En el fichero **values.yaml** se han implementado los recursos que va ha usar dicho helm en el despliegue. Se ha definido *nvidia.com/gpu:1* para que haga uso de la GPU. Para definir que imagen tiene que ejecutar dicho helm se le ha definido como imagen el repositorio anteriormente creado en el Dockerfile. El repositorio ha sido subido a DockerHub, ya que el Helm no era capaz de encontrar dicho repositorio en la máquina local. Para que baje el repositorio desde DockerHub siempre, se le ha asignado *pullPolicy: Always*.

También se ha modificado el fichero **pod.yaml**. Se le han asignado unos comandos para que ejecute el pod. En este caso los comandos son comandos para la ejecución de la aplicación. Es decir, cuando el pod se despliegue, tiene que ejecutar la aplicación que se le marque en estos comandos. Para cambiar de aplicación, hay que comentar la línea que no se quiera ejecutar y des-comentar la que se quiera ejecutar. En este fichero se ha definido un volumen, esto es, para que el resultado (los vídeos con las detecciones) se guarde en la máquina local. Con esto se ha conseguido obtener estos vídeos de manera permanente, ya que si se dejan que se guarden en el contenedor, al acabar el pod se vuelven a destruir. Se ha creado una carpeta en el escritorio local y una carpeta en el contenedor, los cuales se quedan sincronizados a través de este volumen. Si se crea un fichero dentro del contenedor, también se crea en esa carpeta local.

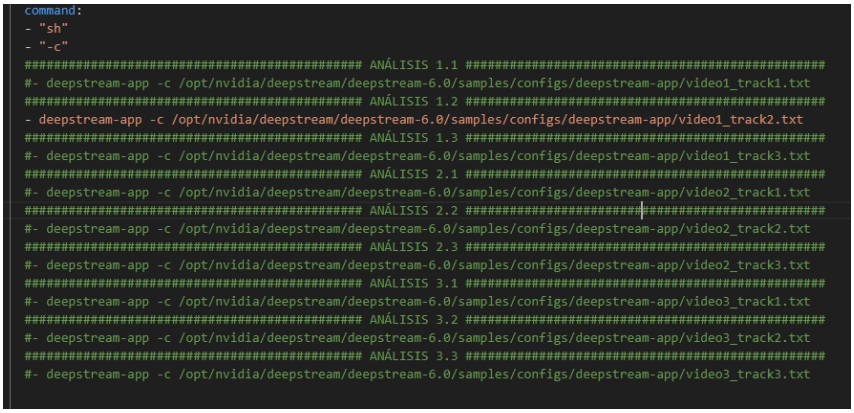


Figura 4.3: Los comandos para la ejecución de las aplicaciones

4.4. Ejecución de algoritmos

Después de haber preparado el entorno de trabajo y creado los elementos necesarios para la ejecución de las aplicaciones, se ha iniciado la ejecución de dichas aplicaciones. Se ejecutarán tres algoritmos distintos con distintos pesos de rendimiento con tres vídeos distintos. En el primer algoritmo se hará solo una detección primaria, mientras que en el segundo, además de esa detección, también se hará una secundaria. En el tercero se hará una primaria y dos secundarias. El primer vídeo es grabado por una cámara estática y hay muchas detecciones, mientras que en el segundo solo hay dos vehículos por detectar. Por último, el tercer vídeo es grabado por una cámara en movimiento, es decir, además de haber mucho tráfico, la cámara está situada dentro de un vehículo en movimiento. El primer objetivo de estas ejecuciones es examinar el rendimiento de los algoritmos. Que ocurre cuando se ejecutan con distintos tipos de cámara, si la cámara está en estático o si está en movimiento, etc. El segundo objetivo es estudiar el rendimiento de la GPU misma cuando ejecutamos en un entorno virtual. Las diferencias de rendimiento ejecutando un algoritmo u otro, si la diferencia es significante o no. También se quiere analizar lo pasa si tiene cero vehículos para detectar o mil.

4.4.1. Nvidia DeepStream

Para el desarrollo del proyecto, se han usado aplicaciones ya implementadas por Nvidia DeepStream[13] que se encuentra en una imagen de un contenedor. DeepStream SDK de Nvidia es un conjunto de herramientas de análisis de transmisión basado en GStreamer para el procesamiento de múltiples sensores basados en IA, vídeo, audio y comprensión de imágenes. Según Nvidia, es ideal para desarrolladores de IA de visión, socios de software, nuevas empresas y OEM que crean aplicaciones y servicios de IVA. Los desarrolladores ahora pueden crear canalizaciones de procesamiento de transmisión que incorporan redes neuronales y otras tareas de procesamiento complejas, como el seguimiento, la codificación/descodificación de vídeo y la reproducción de vídeo. Deepstream permite el análisis en tiempo real de los datos de vídeo, imágenes y sensores.

Esta herramienta ofrece una amplia compatibilidad con modelos de IA para modelos populares de detección y seguimiento de objetos, como SSD de última generación, YOLO, FasterRCNN y MaskRCNN. También puede integrar funciones y bibliotecas personalizadas.

Los algoritmos ejecutados en este proyecto son:

1. **CarDetection**: Detección y seguimiento de todo tipo de vehículos.
2. **CarDetection + VehicleType**: Además de hacer una primera detección de todo tipo de vehículos, se hace una clasificación secundaria donde se clasifican dependiendo del tipo de vehículo (van, large vehicle, etc.).
3. **CarDetection + VehicleType + CarMake** : Aparte de la detección hecha en el segundo algoritmo, también se clasifican por marca de coche (audi, toyota, etc.).

Cada uno de los algoritmos se ha ejecutado con tres vídeos distintos. El primer vídeo es un vídeo grabado por una cámara estática. Hay dos carriles, una lejana y otra cercana, y la cámara está a un lado. El segundo vídeo es una cámara de garaje, es decir, está grabando desde una perspectiva más alta. El tercer vídeo muy similar al primero, aunque esta vez la cámara está en movimiento, va dentro de un vehículo.

Figura 4.4: Video 1

Figura 4.5: Video 2

Figura 4.6: Video 3

CAPÍTULO 5

Análisis de los resultados

Respecto a las métricas del proyecto, estas son las dos ramas principales que se han analizado en este proyecto: análisis de dichos algoritmos de detección y seguimiento de objetos y el análisis de rendimiento de la GPU de Nvidia virtualizada corriendo dichos algoritmos. Para ello se han usado distintos métodos.

5.1. Análisis de rendimiento de algoritmos

5.1.1. Metodología

El análisis de rendimiento de los algoritmos de detección y seguimiento de objetos se ha llevado a cabo a través de las matrices de confusión. Una vez completada la matriz se han sacado tres parámetros: la exactitud, la precisión, la sensibilidad y F1.

Las matrices de confusión se completan de la siguiente manera:

		Actual	
		Positivo	Negativo
Predicción	Positivo	TP	FP
	Negativo	FN	TN

TP: Verdaderos Positivos. Cuando el algoritmo detecta positivo y actualmente también es positivo.

FP: Falsos Positivos. Una detección falsa de un positivo, es decir, el algoritmo detecta algo que no existe.

FN: Verdaderos Negativos. Un objeto/persona no detectado. El algoritmo no detecta algo que tiene que detectar.

TN: Falsos Negativos. En los algoritmos de detección y seguimiento de objetos no podemos analizar este caso, ya que hay muchos objetos en el video que pueden ser no detectados. Este valor no nos importa.

Una vez hechas las matrices de confusión, se han calculado unos parametros. *La Exactitud* es la proporción entre las predicciones correctas que ha hecho el modelo y el total de predicciones.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

La Precisión indica el porcentaje de todos los casos en el que el algoritmo a predicho como positivo, cuales son realmente positivos.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{alldetections} \tag{5.2}$$

La Sensibilidad indica el porcentaje de todas las clases positivas, cuántas ha predicho correctamente el algoritmo.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{alltruths} \tag{5.3}$$

F1 resume la precisión y sensibilidad en una sola métrica. Es muy útil cuando la distribución de las clases es desigual. En este caso no es un parámetro muy significante.

$$F1 = 2\left(\frac{Precision * Recall}{Precision + Recall}\right) \tag{5.4}$$

5.1.2. Resultados

5.1.2.1. Video 1

En el caso del **primer video** se han obtenido las métricas que se muestran en la tabla 5.1.

Tabla 5.1: Video 1

	Precision	Recall	F1 score
CarDetection	1	0,9776119403	0,9886792453
CarDetection + VehicleType	1	0,5114503817	0,6767676768
CarDetection + VehicleType + CarMake	1	0,3740458015	0,5444444444

En el caso de la métrica de la *Precisión*, el resultado obtenido en la ejecución de los tres algoritmos en este vídeo, has ido perfecto. Esto quiere decir, que entre las detecciones que ha hecho el algoritmo no ha habido ningún error.

Respecto al valor de la *Sensibilidad* o también llamado *Recall* se puede observar claramente que en la primera ejecución del algoritmo el valor es casi perfecto, esto es, el primer algoritmo ha detectado casi todos los casos que tenía que detectar. No se puede decir lo mismo del segundo algoritmo, donde el valor de esta métrica ha bajado considerablemente (casi a la mitad). Lo mismo ha pasado en el tercer caso, donde se ha ejecutado una detección primaria y dos secundarias.

Si se analiza el valor del *F1* concluyendo las dos métricas anteriores, se ha obtenido un valor casi perfecto en el primer caso, donde solo se hace una detección primaria. En el segundo caso (detección primaria + una secundaria) ha bajado el resultado, aunque no tan considerablemente como en el *Recall*. También ha bajado el valor del *F1* del segundo caso al tercero, pero no ha sido una bajada significante. Resumiendo, hemos obtenido buenos resultados en este vídeo, donde hay muchas detecciones posibles, y teniendo en cuenta que el carril lejano no es tan visible como el carril cercano. Algunos de los casos no detectados por el algoritmo también no eran posibles de detectar por el ojo humano.

5.1.2.2. Video 2

En el caso del **segundo vídeo** todos los valores obtenidos han sido 0. Hay que tener en cuenta que en este vídeo hay dos detecciones posibles, ya que pasan solo dos vehículos y la cámara está apuntando desde arriba. Como se puede observar, el algoritmo no es capaz de detectar nada en ninguno de los casos. Por lo tanto, los valores de la *Precisión*, *Sensibilidad* y *F1* ha sido 0 en todos los casos. Si el algoritmo de detección primaria (CarDetection) no es capaz de detectar ningún vehículo, los algoritmos de una detección secundaria no serán capaz de detectar nada.

5.1.2.3. Video 3

Por último, estos son los datos de los resultados obtenidos ejecutando los algoritmos en el **tercer vídeo**. Este vídeo se acerca mucho al primer vídeo, aunque hay una diferencia significante. En este caso la cámara se sitúa dentro de un vehículo en movimiento. La cámara iría en la misma dirección que los vehículos del carril cercano, teniendo más tiempo de detección. Sin embargo, si nos centramos en el carril lejano, estos vehículos pasan a una velocidad más rápida, ya que van en dirección contraria.

Tabla 5.2: Video 3

	Precision	Recall	F1 score
CarDetection	1	0,92592593	0,9615384615
CarDetection + VehicleType	1	0,5	0,66666666
CarDetection + VehicleType + CarMake	0	0	0

Los valores obtenidos en la *Precisión* han sido buenos en los casos de solo una detección primaria y una detección primaria más una secundaria. De las detecciones que hayan hecho en estos dos casos los dos algoritmos, no ha habido ningún error, las detecciones se han hecho correctamente. El tercer algoritmo no ha sido capaz de hacer ninguna detección.

En cuanto al valor de *Recall* se puede observar que en el primer algoritmo el valor ha sido casi perfecto. En el segundo algoritmo ha bajado hasta la mitad y en el tercer algoritmo el valor ha sido de 0.

En cuanto al valor del *F1* como conclusión de los otros dos valores, hemos obtenido un resultado parecido a las ejecuciones del primer video. Se ve claramente que la diferencia entre las ejecuciones del primer video y del tercer video es el tercer algoritmo. Se ha concluido que el tercer algoritmo tiene problemas de detección cuando la cámara está en movimiento, ya que no ha detectado ningún vehiculo.

5.2. Análisis de rendimiento de la GPU

5.2.1. Metodología

5.2.1.1. Prometheus

Para el análisis del rendimiento de la GPU se ha usado un conjunto de herramientas de monitoreo llamado *Prometheus*[15]. *Prometheus* recopila y almacena las métricas a lo largo del tiempo. Prometheus utiliza un modelo de recolección de datos basado en scrapping (raspado) o pull. Los componentes denominados .exporters.exponen las métricas a través de una interfaz HTTP, y Prometheus realiza solicitudes periódicas a estos exportadores para obtener los datos. Cuenta con un lenguaje de consulta propio llamado PromQL. Con este lenguaje, los usuarios pueden formular consultas flexibles y expresivas para extraer y manipular datos de métricas. Esto permite realizar análisis avanzados y generar visualizaciones personalizadas.

El almacenamiento de las métricas en una base de datos los hace en series temporales que permiten el análisis y la recuperación eficientes de los datos. También proporciona herramientas y bibliotecas para la recolección y el etiquetado de métricas. Esto permite a los desarrolladores instrumentar sus aplicaciones y servicios para exponer métricas personalizadas y agregar etiquetas a las métricas existentes para una mejor clasificación y agrupación. Se pueden definir reglas y condiciones para generar alertas cuando las métricas superan ciertos umbrales o cumplen ciertas condiciones. Las alertas se pueden enviar a través de diferentes canales, como correo electrónico, sistemas de mensajería o integraciones con sistemas de gestión de incidentes.

Se integra estrechamente con Grafana, una popular plataforma de visualización de datos. Esta integración permite crear paneles y gráficos personalizados para visualizar y analizar los datos de métricas recopilados por Prometheus que explicaremos a continuación. Se configura en una configuración de alta disponibilidad para garantizar la recopilación continua de datos, incluso en caso de fallas en los componentes individuales. Además, Prometheus es altamente escalable y admite la configuración de múltiples instancias para manejar grandes volúmenes de métricas. Por último, esta herramienta de monitorización cuenta con una comunidad activa y un ecosistema creciente de integraciones y complementos. Esto incluye una amplia variedad de exportadores disponibles para recopilar métricas de diversos sistemas y servicios, así como bibliotecas y herramientas de apoyo desarrolladas por la comunidad.

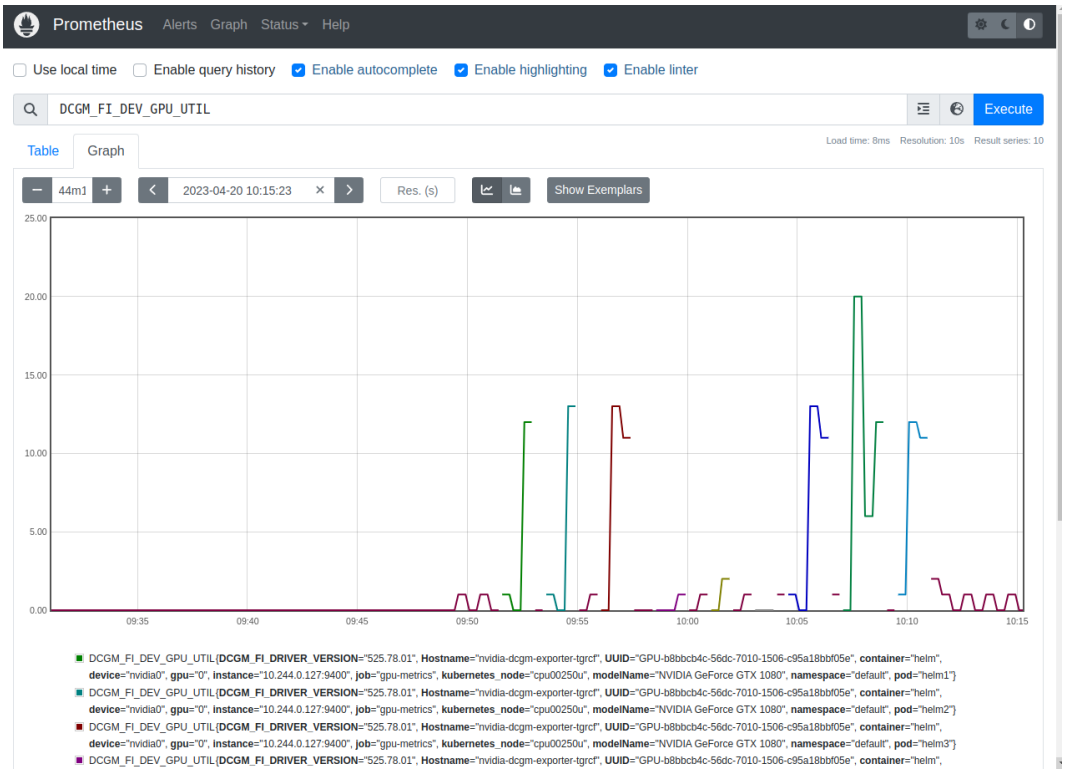


Figura 5.1: Dashboard de Prometheus con las ejecuciones de los Helm Charts

5.2.1.2. DCGM Exporter

DCGM Exporter[16] es un componente de software utilizado en el ámbito de la monitorización y gestión de GPU (Unidades de Procesamiento Gráfico) en entornos de infraestructura de centros de datos. DCGM significa "Data Center GPU Manager"(Administrador de GPU para centros de datos) y es una herramienta desarrollada por Nvidia. Esta herramienta permite recopilar métricas y datos de rendimiento de las GPU Nvidia instaladas en un sistema y exponerlos a través de una interfaz HTTP que sigue el estándar de exposición de métricas de Prometheus. Esto significa que DCGM Exporter es compatible con Prometheus y puede integrarse fácilmente en una infraestructura de monitorización basada en Prometheus y Grafana.

Al utilizar DCGM Exporter, puedes obtener información detallada sobre el estado y el rendimiento de las GPU en tu sistema. Esto incluye métricas como la temperatura, el uso de la GPU, la memoria utilizada, la velocidad del reloj, las tasas de transferencia, los errores y muchas otras métricas relacionadas con el rendimiento y la salud de las GPU. Al exponer estas métricas a través de la interfaz HTTP, DCGM Exporter permite a los usuarios recopilar y almacenar los datos de la GPU en un sistema centralizado de Prometheus, que luego se puede utilizar para monitorear y visualizar el rendimiento de las GPU en tiempo real utilizando Grafana u otras herramientas compatibles con Prometheus. Es particularmente útil en entornos de centros de datos que utilizan GPU para cargas de trabajo intensivas, como el aprendizaje automático (machine learning), la inteligencia artificial (AI), la renderización 3D y la computación de alto rendimiento (HPC), donde es importante monitorear y optimizar el rendimiento de las GPU para garantizar un funcionamiento eficiente y confiable.

5.2.1.3. Grafana

Grafana[17] es una plataforma de visualización y análisis de datos de código abierto que permite monitorear, analizar y comprender los datos en tiempo real. Proporciona herramientas flexibles para la creación de paneles interactivos y tableros de control también llamado *Dashboard*, lo que permite a los usuarios visualizar y explorar datos complejos de manera efectiva.

Esta herramienta ofrece una amplia gama de opciones de visualización, incluyendo gráficos de líneas, gráficos de barras, diagramas de dispersión, gráficos circulares, tablas y mapas, por lo que permite representar los datos de manera clara y comprensible. Además, es compatible con una gran variedad de fuentes de datos, incluyendo bases de datos relacionales, bases de datos NoSQL, servicios de nube, sistemas de monitorización y herramientas de métricas, facilitando la integración de datos provenientes de diferentes fuentes en un solo panel de control. El lenguaje usado por Grafana se llama Grafana Query Language(GQL), y permite realizar consultas complejas y aplicar

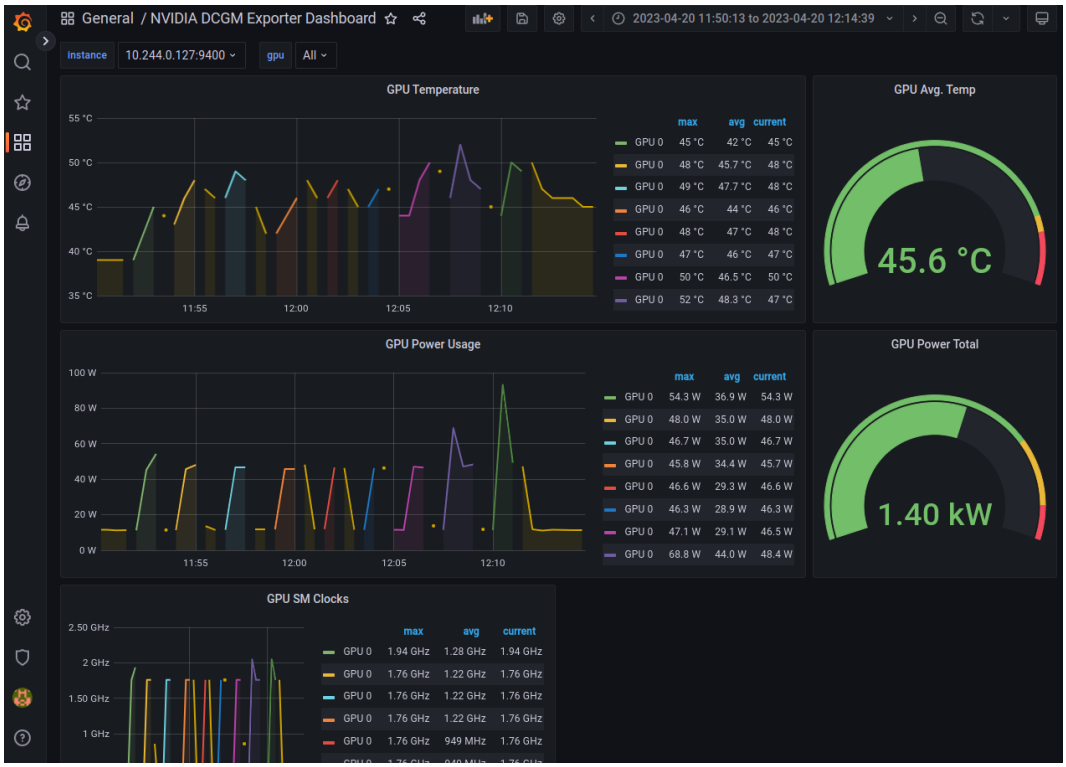


Figura 5.2: Dashboard de Grafana con las ejecuciones de los Helm Charts

5.2.1.4. Métricas de la GPU

De todas las métricas posibles para analizar la GPU, se han elegido cuatro y son las siguientes.

La métrica **GPU utilization**[18] proporciona información sobre el nivel de uso o utilización de una unidad de procesamiento gráfico (GPU). Esta métrica generalmente se refiere a la cantidad de tiempo que la GPU está ocupada realizando tareas de procesamiento en comparación con el tiempo total disponible.Este valor se representa como un porcentaje, donde un valor del 0 % indica que la GPU no está siendo utilizada en absoluto, mientras que un valor del 100 % indica que la GPU está completamente ocupada realizando tareas de procesamiento. Esta métrica permite tener una visión clara del nivel de carga y rendimiento de la GPU, lo que puede ayudar a identificar partes costosas, optimizar el uso de recursos y tomar decisiones informadas sobre la capacidad y escalabilidad del sistema.

GPU memory used proporciona información sobre la cantidad de memoria de la unidad de procesamiento gráfico (GPU) que se encuentra en uso en un momento dado. Esta métrica indica la cantidad de memoria que se está utilizando activamente para almacenar datos y realizar operaciones de procesamiento en la GPU. El valor se puede expresar en diferentes unidades, como bytes, kilobytes, megabytes o gigabytes, dependiendo de cómo esté configurado el sistema de monitoreo. Este valor puede cambiar a medida que las aplicaciones o procesos en ejecución en la GPU asignan o liberan memoria. Es importante para el monitoreo y la gestión de la memoria en sistemas que utilizan GPU para tareas de alto rendimiento porque permite realizar un seguimiento del uso de la memoria en la GPU, identificar patrones de consumo y detectar posibles problemas de asignación o fugas de memoria.

GPU power usage en Prometheus proporciona información sobre la cantidad de energía eléctrica que está siendo consumida por una unidad de procesamiento gráfico (GPU) en un momento dado. Esta métrica refleja el nivel de consumo de energía de la GPU durante su funcionamiento. Este valor se expresa en vatios (W) o en alguna otra unidad de medida de energía y puede variar en función de la carga de trabajo de la GPU y de otros factores, como la configuración de energía del sistema. Esta métrica es importante porque permite tener una idea de la cantidad de energía eléctrica que la GPU está consumiendo, lo cual puede ser útil para evaluar la eficiencia energética del sistema y optimizar el uso de recursos.

La métrica **GPU temperature** tal y como indica el nombre, proporciona información sobre la temperatura actual de una unidad de procesamiento gráfico (GPU). Esta métrica indica la temperatura a la que se encuentra operando la GPU cada momento de la ejecución. Este valor generalmente se indica en grados Celsius (°C) o en alguna otra unidad de medida de temperatura. Analizar la temperatura es importante para mantener un rendimiento óptimo y prevenir problemas relacionados con el sobrecalentamiento. Una temperatura excesivamente alta puede indicar que la GPU está operando en condiciones de estrés térmico, lo que puede afectar su rendimiento y vida útil. Además, altas temperaturas pueden ser un indicador de problemas en la refrigeración del sistema.

5.2.2. Métricas

Estos son los datos que se han obtenido de las ejecuciones de los Helm Charts en cada uno de los casos. Como se ha explicado antes en la sección de *Desarrollo del proyecto*, se han implementado nueve Helm Charts. Se han ejecutado tres diferentes algoritmos, en tres vídeos distintos. Las barras de color gris claro indican los datos de las ejecuciones del primer algoritmo (solo se hace una detección primaria). Las barras de color gris del medio nos indican las métricas del segundo algoritmo (una detección primaria más una secundaria). Para terminar, el color negro nos indica los datos del tercer algoritmo (una detección primaria más dos secundarias). Los colores se mantienen para las cuatro métricas.

Para empezar en el siguiente gráfico tenemos los resultados de la métrica *GPU Utilization*. Si comentamos los valores en general, se puede apreciar que el valor de la utilización de la GPU ha sido baja, como máximo se ha obtenido un 16 %. Este valor se ha dado en el caso de la ejecución del tercer algoritmo mientras se ejecutaba el tercer vídeo. Como en los tres vídeos el valor más alto obtenido ha sido en el tercer algoritmo, se puede concluir que el tercer algoritmo es más costoso que los dos anteriores. Es lógico ya que el tercero ejecuta lo mismo que los dos anteriores más una detección secundaria. También es de mencionar el hecho que en las ejecuciones del segundo vídeo, independientemente del algoritmo ejecutado, los valores obtenidos son muy bajos. Esto se puede dar porque en este vídeo no se hace ninguna detección en ninguno de los tres casos.

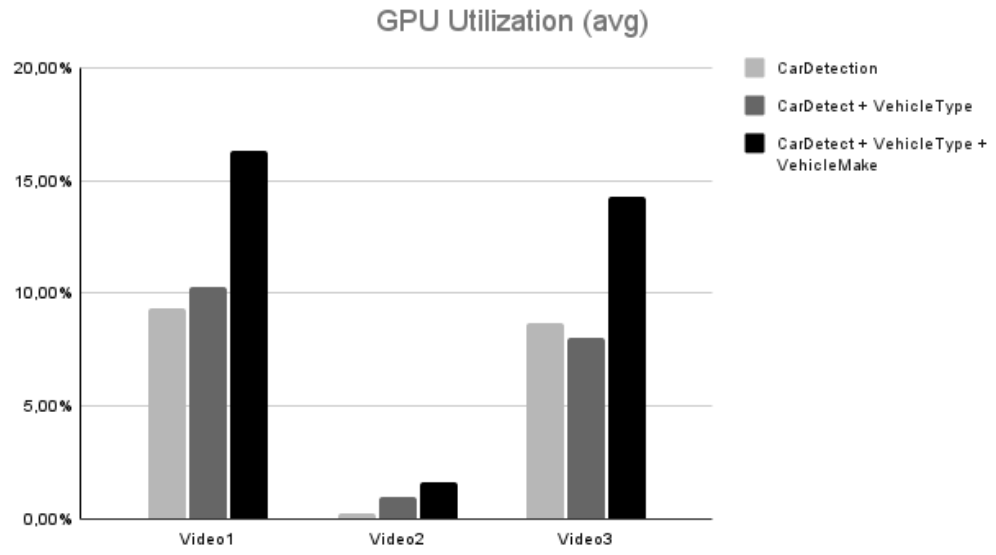


Figura 5.3: Utilización de la GPU

En el segundo gráfico se muestran los resultados de la energía usada en la GPU. El mayor valor obtenido en esta métrica es de 35 W, se podría decir, que es considerablemente baja. También es de considerar que de vídeo a vídeo el valor no cambia significativamente, es decir la GPU gasta la misma energía si ha detectado cincuenta vehículos o si ha detectado cero vehículos. Sí que varía la energía consumida de algoritmo a algoritmo, el primer algoritmo gasta menos energía que el segundo, y el segundo gasta menos que el tercero. Aunque en este caso no suba exageradamente el valor.

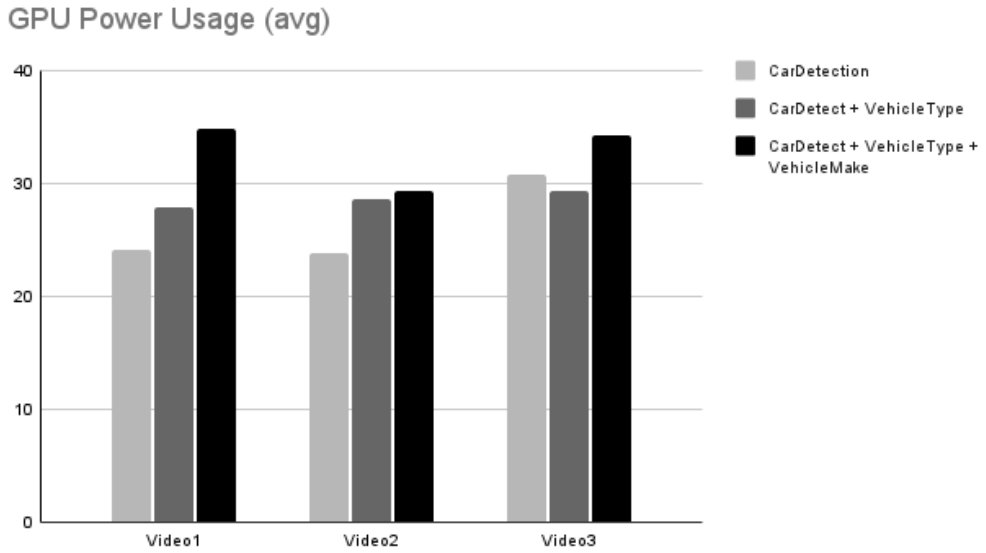


Figura 5.4: Uso de la energía GPU

A continuación tenemos los datos de la Temperatura de la GPU. Es de mencionar que de ejecución a ejecución de los Helm Charts se ha dejado el suficiente tiempo para que la temperatura vuelva a su estado normal. El mayor valor obtenido en esta métrica ha sido de 51,3ºC y en la ejecución del tercer vídeo con el primer algoritmo. Después de haber obtenido estos datos, se puede decir que no se observa ninguna diferencia significante para que decir que cierto algoritmo o cierto vídeo produce una subida en la temperatura. Se puede concluir que la temperatura se mantiene a lo largo de todas las ejecuciones de los Helms.

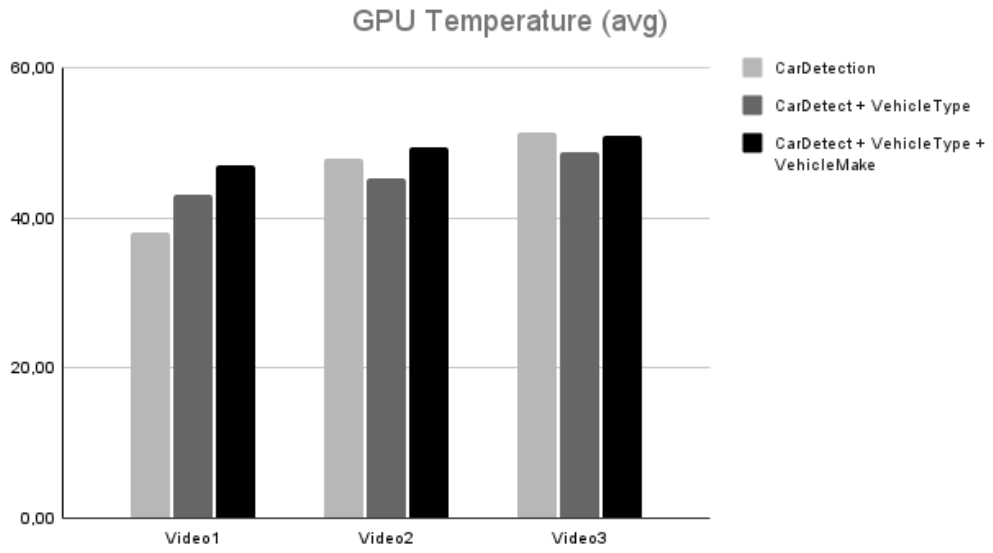


Figura 5.5: Temperatura de la GPU

Para terminar, se han sacado los datos de la memoria usada en las distintas ejecuciones. El valor máximo obtenido ha sido de 1,280 GB y se ha obtenido en dos casos, en las ejecuciones del tercer vídeo con el segundo y tercer algoritmo. Al igual que en el caso de la temperatura, la conclusión sacada de estos datos es que la diferencia en la cantidad de datos que ha utilizado en la memoria en cada uno de los casos no ha sido apreciable en estos casos.

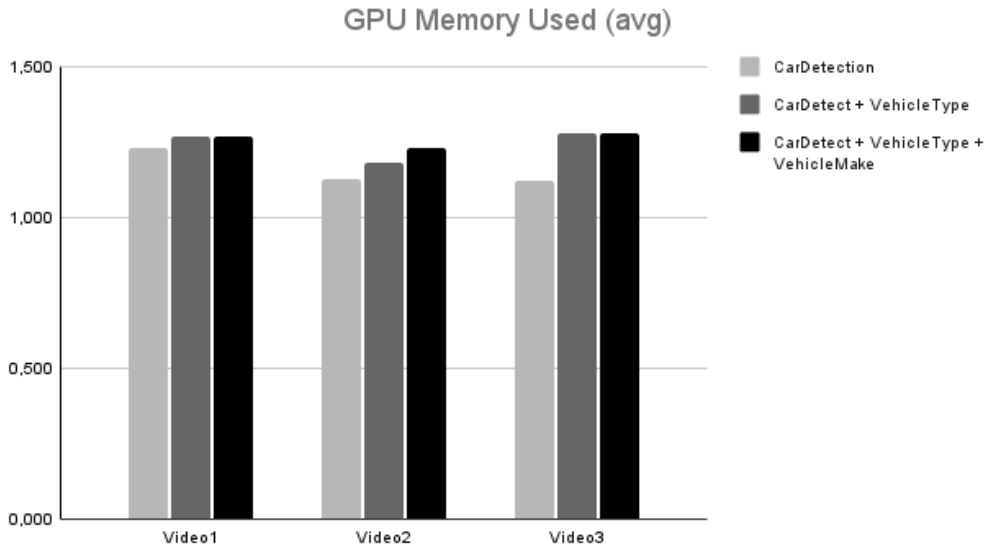


Figura 5.6: Memoria usada de la GPU

CAPÍTULO 6

Conclusión

6.1. Producto

Se han implementado y obtenido varios productos en este proyecto. Primero, es de mencionar el análisis y la información que se ha recopilado antes de dar inicio al proyecto. Se ha recopilado información y se ha investigado mayormente sobre las herramientas usadas en el desarrollo, como por ejemplo: algoritmos de alto rendimiento, algoritmos de detección y seguimiento de objetos, la aceleración por la GPU, la virtualización y las técnicas, etc.

En el desarrollo, se ha instalado un clúster local de Kubernetes para desplegar los contenedores y los Helm Charts. Se han desplegado los servicios para la monitorización de la GPU. En el siguiente paso, se ha preparado e implementado un contenedor Docker. Cogiendo como base una imagen de Nvidia DeepStream donde están las aplicaciones que se han usado para el análisis. También se ha implementado un Helm Chart que se ha utilizado para desplegar el contenedor creado anteriormente. En este Helm Chart, se han automatizado las distintas ejecuciones de distintos algoritmos y distintos vídeos para comparar resultados.

La creación de un repositorio de GitHub también ha sido otro de los productos que se han logrado en este proyecto. En él se han subido los productos anteriores, esto es, el contenedor Docker con su Dockerfile y el Helm Chart con sus componentes. También se ha redactado un Readmeçon todos los pasos para la instalación de los recursos, la puesta en marcha del Helm Chart y los pasos para la monitorización. El repositorio ha sido clave para que otras personas del departamento de Vicomtech puedan hacer uso de este proyecto para un proyecto mayor.

6.2. Conclusión

En conclusión, los resultados obtenidos demuestran que el proyecto ha alcanzado sus objetivos principales. El primer objetivo principal era preparar los recursos y la máquina para poder ejecutar los algoritmos de alto rendimiento acelerados por GPU en este entorno virtual. Como conclusión de este objetivo se ha instalado un clúster de Kubernetes como entorno virtual y se ha configurado para que haga uso de la GPU en sus ejecuciones.

Otro de los principales objetivos era analizar el rendimiento de estos algoritmos cuando ejecutamos con aceleración por GPU y en un entorno virtual. Como resultado, hemos visto que el algoritmo es capaz de detectar lo que detecta el ojo humano en casi todos los casos. En el caso del segundo vídeo, cuando la perspectiva de la cámara es desde arriba, estos algoritmos no son capaces de detectar ningún vehículo.

Como tercer y último objetivo principal, el análisis de rendimiento de la GPU, se han sacado cuatro métricas: temperatura de la GPU, la utilización, la memoria usada y la energía consumida en la ejecución del algoritmo. De estas cuatro métricas, en la primera hemos podido observar la diferencia entre una ejecución u otra. Se ha visto claramente que cuantas menos detecciones haya, hay una bajada significante en la utilización de la GPU.

En definitiva, se han alcanzado los objetivos definidos al inicio del proyecto.

6.3. Mejoras en el futuro

Como investigación adicional, uno de los temas en el que investigaría más, sería la aceleración por GPU. Investigar y estudiar más en esa área ayudaría a tener mejor rendimiento tanto en los algoritmos como en la computación. También contribuiría analizar y comparar más profundamente las diferentes técnicas de virtualización, saber y entender mejor que técnica usar en cada caso y los beneficios que traería usar uno u otro en cada momento. Entender mejor las aplicaciones de DeepStream de Nvidia también ayudaría para sacar las conclusiones cuando sacamos los datos de rendimiento. Los contenedores de Nvidia DeepStream tienen un abanico amplio de aplicaciones y algoritmos que ayudarían a tener más comparaciones en los resultados.

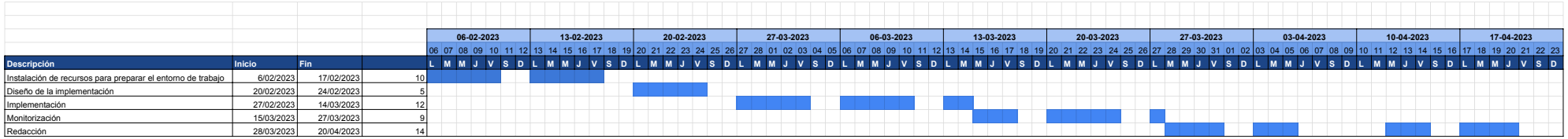
Para dar seguida al proyecto ampliaría el estudio a más tipos de algoritmos de alto rendimiento aparte de los de detección y seguimiento de objetos. Machine Learning tiene gran cantidad de algoritmos de alto rendimiento que en este proyecto no se han mencionado y que son muy importantes hoy en día en muchas áreas de investigación. Sería muy importante replicar el estudio en un contexto más amplio o con una muestra más representativa. Ayudaría a validar los resultados y tener una visión más completa sobre el tema.

Enfocando más en la metodología usada en el proyecto, ha habido problemas en la preparación del entorno del trabajo, que ha sido el primer paso en el desarrollo del proyecto. Instalar estos recursos trajo un desperdicio de tiempo más grande del que se esperaba. Sin mirar demasiado los requisitos de cada herramienta, instalábamos los recursos, y esto nos llevaba a tener problemas de versiones a la hora de instalar otras herramientas también necesarias. Investigar más sobre estas herramientas, nos llevaría a reducir el tiempo malgastado en la preparación del entorno.

Apéndice

Apéndice 1

Diagrama Gantt del proyecto.



Descripción	Inicio	Fin	
Instalación de recursos para preparar el entorno de trabajo	6/02/2023	17/02/2023	10
Diseño de la implementación	20/02/2023	24/02/2023	5
Implementación	27/02/2023	14/03/2023	12
Monitorización	15/03/2023	27/03/2023	9
Redacción	28/03/2023	20/04/2023	14

Bibliografía

[1] Jorge Sánchez-Alor Expósito et al. Evaluación de algoritmos de detección de objetos basados en deep learning para detección de incidencias en carreteras. 2020. Ver página 8.

[2] Rupesh Kumar Rout. A survey on object detection and tracking algorithms. 2023. Ver página 8.

[3] Na8. Modelos de detección de objetos. 2020. Ver página 8.

[4] Ross Girshick. Fast r-cnn. 2015. Ver página 8.

[5] Hewlett Packard. What is gpu computing? Ver página 9.

[6] Virtualtopia. An overview of virtualization techniques. Ver página 11.

[7] Buscar autor. Virtualization techniques. *Virtualization techniques*, 2023. Ver página 11.

[8] Bill Kleyman. Understanding server virtualization. Ver página 12.

[9] Alexander S.Gillis. Container-based virtualization or containerization. Ver página 13.

[10] Kubernetes. Kubernetes documentation. Ver página 14.

[11] NVIDIA. Nvidia drivers downloads. 2023. Ver página 15.

[12] Sergio De Luz. Nvidia docker te permitirá ejecutar aplicaciones con la gpu en contenedores virtuales. 2016. Ver página 16.

[13] NVIDIA. Sdk de nvidia deepstream. 2023. Ver páginas 17, 19.

[14] Shubham Agarval. What is a dockerfile and how does it work? 2022. Ver página 17.

[15] Prometheus. Prometheus monitorization. *Prometheus monitorization*, 2023. Ver página 27.

[16] NVIDIA. Gpu monitoring. *DCGM Exporter*, 2023. Ver página 29.

[17] Grafana Labs. Grafana dashboards overview. 2023. Ver página 29.

[18] ExxactCorp. Top 5 metrics for evaluating your deep learning program's gpu performance. 2019. Ver página 31.