INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"AUTOMATIZACIÓN DE UN AULA INTELIGENTE BASADA EN UNA ARQUITECTURA DE CÓMPUTO OSMÓTICO Y APRENDIZAJE PROFUNDO"

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERÍA MECATRÓNICA

PRESENTA:

ING. PABLO CANO FUENTES

DIRECTOR DE TESIS:
M.C. ALBERTO PACHECO GONZÁLEZ

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

CHIHUAHUA, CHIH., AGOSTO 2020

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MEXICO
INSTITUTO TECNOLÓGICO DE CHIHUAHUA

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Chihuahua, Chih., 18 de agosto de 2020

M.C. ROGELIO E. BARAY ARANA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de la presente notificamos a usted que en cumplimiento de los requerimientos para la obtención de grado de Maestro, el documento de tesis del c. **PABLO CANO FUENTES**, ha sido aprobado y aceptado para su impresión. El título de la tesis es:

"AUTOMATIZACIÓN DE UN AULA INTELIGENTE BASADA EN UNA ARTQUITECTURA DE CÓMPUTO OSMÓTICO Y APRENDIZAJE PROFUNDO"

Por lo que proponemos, le sea concedida la autorización de. impresión correspondiente.

Agradeciendo la atención a la presente, quedamos de usted:

ATENTAMENTE
Excelencia en Educación Tecnológica®

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA | TECNOLOGICO NACIONAL DE MEXICO
INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

M.C. Alberto Pacheco González
MIEMBRO DEL JURADO DE EXAMEN

Dr. Isidro Robledo Vega.
MIEMBRO DEL JURADO DE EXAMEN

M.C. Pedro Rafael Márquez Gutiérrez
MIEMBRO DEL JURADO DE EXAMEN

Dra. Carmen Leticia García Mata.
MIEMBRO DEL JURADO DE EXAMEN

c.c.p.archivo
/reba

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Chihuahua, Chih., 18 de agosto de 2020

PABLO CANO FUENTES
PRESENTE

Por este conducto le comunico que, a propuesta del Jurado de Examen, la División de Estudios de Posgrado e Investigación le ha concedido la autorización para la impresión de tesis para obtener el grado de Maestro, cuyo título es:

"AUTOMATIZACIÓN DE UN AULA INTELIGENTE BASADA EN UNA ARTQUITECTURA DE CÓMPUTO OSMÓTICO Y APRENDIZAJE PROFUNDO"

Con el siguiente contenido de capítulos:

I. Introducción.
II. Contexto, área de investigación y trabajos relacionados.
III. Marco Teórico.
IV. Desarrollo del proyecto.
V. API para el Aula Inteligente.
VI. Implementación y pruebas.
VII. Análisis de resultados.
VIII. Conclusiones y trabajo futuro.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®

M.C. ROGELIO E. BARAY ARANA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p. archivo
/reba

DEDICATORIA

Este trabajo está dedicado primeramente a Dios por haberme guiado en el camino en el que me encuentro y por permitirme haber conocido a personas que me ayudaron a formar este trabajo. Entre estas personas están mis padres, mi hermano, mis amigos y mis maestros que siempre estuvieron apoyándome y dándome ánimo para seguir adelante.

AGRADECIMIENTOS

Primero a Dios por darme vida.

A mis padres que me dieron ánimos para inscribirme a la Maestría y con su apoyo incondicional me ayudaron a seguir adelante. A mi padre por sus consejos para mi desarrollo profesional y personal. A mi madre por su amor y paciencia. Muchos de mis logros se los debo a ustedes incluyendo este.

A mi asesor y maestros por guiarme en el trabajo de esta tesis desde mi propuesta hasta los últimos ajustes que se implementaron. Les agradezco sobre todo a mi asesor por ayudarme a que este trabajo llegara a tener presencia en varios congresos y también por su apoyo y confianza en estos dos años

A mis compañeros de Maestría por haberme apoyado en el transcurso de estos dos años y haberme enseñado que existe la amistad. Sobre todo, a una compañera que al final llegué a tenerle mucho afecto y me enseñó a tener más confianza en mí.

A CONACyT por darme la oportunidad de continuar mis estudios apoyándome económicamente y por la confianza para desarrollar este trabajo.

RESUMEN

AUTOMATIZACIÓN DE UN AULA INTELIGENTE BASADA EN UNA ARQUITECTURA DE CÓMPUTO OSMÓTICO Y APRENDIZAJE PROFUNDO

Pablo Cano Fuentes
Maestro en Ingeniería Mecatrónica
División de Estudios de Posgrado e Investigación del
Instituto Tecnológico de Chihuahua
Chihuahua, Chih. 2019
Director de Tesis: Alberto Pacheco González

Estudios recientes pronostican que la mayor tasa de crecimiento de tráfico en red será para teléfonos inteligentes y dispositivos conectados a Internet. Un mayor número de dispositivos tiende a realizar tareas cada vez más demandantes operando sobre una mayor cantidad de datos. El Aprendizaje Automático y la Computación Osmótica se perfilan como metodologías eficaces e innovadoras para procesar las grandes cantidades de datos para el Internet de las Cosas y sus aplicaciones emergentes. La Computación Osmótica descompone un sistema en microservicios para distribuir el procesamiento a lo largo de la infraestructura disponible en tres capas: el Cómputo en la Frontera, la Niebla y la Nube. Se propone una arquitectura IoT osmótica para realizar un sistema de automatización de un aula inteligente con la ayuda de modelos de Aprendizaje Automático pre-entrenados para la detección de personas y así poder realizar un análisis comparativo de los nodos de esta arquitectura midiendo los cuadros por segundo que cada capa de la arquitectura puede conseguir para detectar personas de un video con flujo continuo y en tiempo real. Al mismo tiempo poder controlar algunos dispositivos IoT del aula inteligente como las luces, el aire acondicionado y un proyector. En este trabajo fue posible demostrar que efectivamente la capa correspondiente al borde de la red de la arquitectura IoT osmótica ofrece un mejor rendimiento para ejecutar inferencias CNN y tomar acciones de control en el menor lapso. Sin embargo, la capa del Cómputo en la Niebla ofrece ventajas adicionales con un rendimiento ligeramente inferior a la capa de Cómputo en la Frontera de alrededor de 1.5 cuadros por segundo.

INDICE

LISTA DE FIGURAS

LISTA DE TABLAS

I. INTRODUCCIÓN

En los últimos años, el Internet de las cosas (IoT, *Internet of Things*) se ha convertido en un paradigma informático omnipresente en nuestras vidas cotidianas, debido a que estos sistemas son capaces de rastrear la actividad física de personas desde sus teléfonos inteligentes, cámaras de vigilancia y altavoces inteligentes tipo Alexa para las casas inteligentes, entre otras aplicaciones. La Inteligencia Artificial (IA) ha recobrado un auge dentro de los entornos de IoT y debido al rotundo éxito de una de sus áreas, el Aprendizaje Profundo (DL, *Deep Learning*), más específicamente a las Redes Neuronales Convolucionales (CNN, *Convolutional Neural Networks*) que brindan la capacidad para analizar grandes volúmenes de datos (*Big Data*), logrando un mejor desempeño en tareas como el reconocimiento de voz y la detección de objetos en imágenes, lo que les confiere un lugar muy destacado para ser utilizadas como herramientas fundamentales en muchas de las aplicaciones emergentes dentro de IoT en los hogares, edificios, plantas industriales y ciudades inteligentes (Pacheco et al., 2018). Estas aplicaciones y servicios proporcionados por grandes compañías como Google, Microsoft y Amazon, normalmente se soportan de manera remota con las colosales infraestructuras de Cómputo en la Nube (*Cloud Computing, CC*), tecnología que ha logrado alcanzar su etapa de madurez desde finales de la década pasada (Zhang, 2010), pero tiene sus limitaciones en cuanto a latencia, seguridad, privacidad, el ancho de banda, entre otros (Singh, 2017). Sin embargo, como soluciones a los problemas de CC emergieron otras dos arquitecturas: el Cómputo en la Frontera (*Edge Computing, EC*) y Cómputo en la Niebla (*Fog Computing, FC*). El procesamiento de datos y los cálculos que ocurren en el borde de la red se llevan a cabo más cerca del origen de los propios datos y desde luego tiene también sus limitaciones como lo son la dependencia de consumo de energía de baterías, su limitada capacidad de procesamiento debido al tamaño de sus procesadores y la baja capacidad de almacenamiento local (Mach & Becvar, 2017). Por otro lado, Cómputo en la Niebla es un intermedio entre la capa inmediata del Cómputo en la Frontera y la muy lejana capa de Cómputo en la Nube, y cuyos nodos son distribuidos a través de la red intermedia y se ubican a un paso más cerca del usuario final (Cisco, 2015). Por lo que tanto el Cómputo en la Niebla y Cómputo en la Frontera son una solución para sustituir o complementar el Cómputo en la Nube que los denominaremos como FEC (*Fog and Edge Computing*).

La tesis forma parte del proyecto de investigación "Aula Inteligente con Interfaz Natural basada en Realidad Aumentada y Aprendizaje Automático" del Laboratorio de Aprendizaje Móvil y Sistemas Embebidos Inteligentes de la Maestría en Ingeniería Mecatrónica en el Instituto Tecnológico de Chihuahua, de donde se derivaron diversas publicaciones disponibles en la biblioteca digital de IEEE (Pacheco et al., 2018a; Pacheco et al., 2018b). En dicho proyecto se diseñó una plataforma IoT para edificios inteligentes capaz de ejecutar en distintos dispositivos IoT de bajo costo con la ayuda de diversos modelos de redes neuronales convolucionales para la detección y seguimiento de personas en un aula inteligente. El enfoque de esta tesis consiste en diseñar e implementar una arquitectura IoT distribuida y escalable para dicha aula inteligente que conectará los distintos dispositivos IoT comunes de las aulas, evaluando la mejor combinación de las capas de la arquitectura de Cómputo Osmótico, los cuales son: el EC, FC y CC, como se observa en la figura 1.1. Para evaluar la implementación de dicha arquitectura osmótica de IoT se propuso realizar un estudio comparativo entre las diversas configuraciones de nodos CC, FC, y EC para determinar cuál opción resulta más conveniente para procesar datos en tiempo real (minimizar la latencia de transferencia de datos) y requerir menos recursos de cómputo (minimizar el costo del sistema IoT), para derivar, a partir del presente trabajo, una serie de recomendaciones y criterios de diseño más acordes para ejecutar en dispositivos IoT modelos de aprendizaje automático para soportar aplicaciones de IoT para automatización en tiempo real, aplicando las arquitecturas de Cómputo Osmótico.

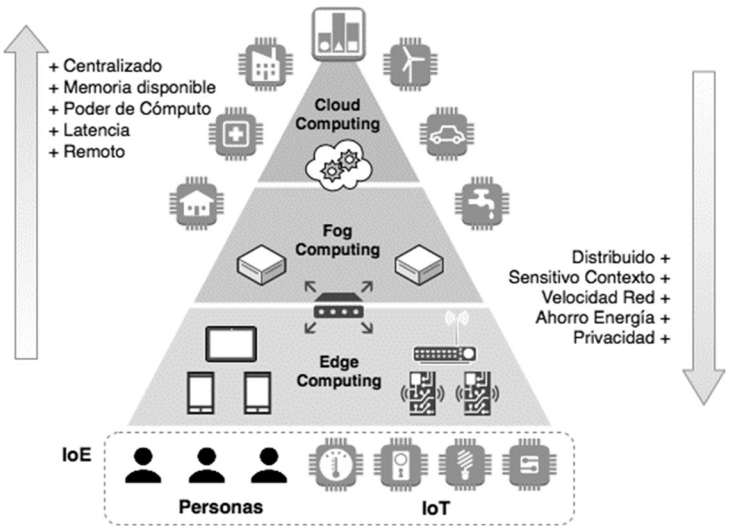

Figura I.1 Arquitectura IoT. Fuente: (Pacheco et al.,2018a).

Para implementar y realizar las pruebas comparativas del rendimiento de cada alternativa de nodos FEC, se pretende establecer ya sea un *framework*, contenedor o API que haga posible transferir y ejecutar los mismos procesos de aplicación (ejecución de modelos CNN y control de ambientación del aula) en cada uno de los distintos nodos de la arquitectura osmótica de IoT. Para ello es importante enfatizar la observación del hecho de que cada nodo FEC posee muy distintas arquitecturas de *hardware,* es decir, que tienen muy distintos requerimientos de consumo de energía y brindan muy distintas capacidades de procesamiento y almacenamiento, por lo tanto, es parte del presente trabajo abordar dicha problemática y encontrar una solución aunado al problema de interoperabilidad y compatibilidad para satisfacer los requerimientos de control y ejecución de modelos CNN. Mediante el estudio comparativo, será posible determinar si hay diferencias apreciables de rendimiento y latencia, analizando además las virtudes y limitaciones de cada configuración de topología de la arquitectura osmótica IoT propuesta. En todas las pruebas realizadas, se definirá un escenario de prueba estandarizado para que las mediciones realizadas puedan ser comparadas.

Además, fue indispensable definir un protocolo de comunicación de las distintas tecnologías de radiocomunicación para el intercambio de paquetes de datos entre los niveles de la arquitectura IoT de Cómputo Osmótico y, a partir de esta, tomar una decisión para que un equipo del aula realice una acción de control específica. Se buscó determinar los aspectos principales que influyen en cada nivel de la arquitectura IoT para así utilizar los recursos disponibles en cada nodo de la forma más apropiada posible. Al final, la visión del proyecto al cual pertenece la tesis es lograr que el aula inteligente esté completamente automatizada, sea autónoma, escalable y distribuida, siendo la aportación del presente trabajo establecer un primer bosquejo de la arquitectura IoT y un *framework* que permita implementar dichos sistemas a muy bajo costo, en tiempo real y reduzca el consumo de energía.

1.1 El Estado del Arte en IoT

El crecimiento exponencial con respecto a las capacidades de cómputo, almacenamiento y consumo de energía de los dispositivos electrónicos ha permitido que existan dispositivos IoT más potentes donde se puede obtener el análisis de datos más rápido y con menor consumo de

energía (Villari et al., 2016) a diferencia de cómo ha sido realizado normalmente, sobre todo en la década pasada donde el procesamiento y almacenamiento de datos es transferido a los grandes servidores centralizados disponibles en la Nube. A este paradigma dominante se le ha denominado Cómputo en la Nube o solo como la Nube, donde el concepto fundamental es crear un centro de cómputo al que cualquier persona pueda tener acceso a los recursos y servicios que este provee y la modalidad que actualmente promocionan para el cobro del servicio es que solo pagues los recursos que utilizas (G., Mujthaba et al., 2018).

Estudios recientes pronostican que la mayor tasa de crecimiento de tráfico en red será para teléfonos inteligentes y dispositivos conectados a Internet. Un mayor número de dispositivos tiende a realizar tareas cada vez más demandantes operando sobre una mayor cantidad de datos (CISCO, 2017). El Aprendizaje Automático, la Computación en la Niebla y la Computación en la Frontera se perfilan como opciones eficaces e innovadoras para procesar las grandes cantidades de datos sobre todo para el Internet de las Cosas y sus aplicaciones emergentes (Géron, 2017). La Computación Osmótica descompone un sistema en microservicios para distribuir el procesamiento a lo largo de la infraestructura disponible de la frontera, la niebla y la Nube. Tanto el Cómputo en la Frontera como el Cómputo en la Nube tienen sus limitaciones: latencia, seguridad, privacidad, precisión, consumo de recursos de procesamiento, ancho de banda, consumo de energía, entre otros. Por otro lado, el Cómputo en la Niebla tiende a aminorar algunas desventajas del Cómputo en la Nube, al procesar los datos más cerca del lugar de donde éstos se generan (Villari et al., 2016; Morshed et al., 2017).

Las aplicaciones de la Inteligencia Artificial se extienden también a un gran ritmo, con expectativas de adopción en un 35% de las compañías a nivel global para 2030 (Bughin, Seong, Manyika, Chui, & Joshi, 2018). Siendo una de las ramas de IA, el Aprendizaje Automático (*Machine Learning*, ML) tiene actualmente una industria global con una tasa compuesta anual de crecimiento (CAGR) del 42%. El Aprendizaje Profundo se destaca de las demás áreas de ML y posee un crecimiento de $100 millones de dólares en 2018 a una proyección de $935 millones para 2025 (Fakhruddin, 2018). Además, existe un creciente interés académico por estos temas emergentes, por ejemplo, los artículos de IA publicados en arXiv en 2018 fueron tres veces mayores a los de 2015 (Hao, 2019) y en Scopus, el porcentaje de trabajos de IA que abordan aprendizaje automático creció de un 28% en 2010 a 56% en 2017 (Shoham et al., 2018).

1.2 Justificación

Estudios recientes pronostican que para 2021 el tráfico en la Nube se triplicará, la velocidad se duplicará, el tráfico móvil superará al generado por computadoras personales, la mayor tasa de crecimiento será para teléfonos inteligentes y dispositivos conectados a Internet que se estima generarán alrededor de 847 zettabytes en datos (Cisco, 2017; Pacheco et al., 2018a). México tuvo una creciente utilización en la telefonía celular desde el 2001 con un 16% hasta con un 80% en el 2013 (INEGI, 2014). Los teléfonos inteligentes ocupan la mayor parte de las actividades de las personas y cada vez se utilizan más para realizar tareas más demandantes, por lo que la optimización de las aplicaciones móviles para la eficiencia energética ayudaría enormemente a la satisfacción del usuario, por el mejor uso de los recursos. (Hao, 2012)

Algunas investigaciones señalan al Cómputo en la Frontera como una alternativa viable de solución a esta problemática relacionada con la explosión de datos en la red generados por un creciente número de dispositivos IoT, sin embargo, en la mayoría de las investigaciones solo se han contemplado escenarios muy simples o algunas simulaciones y evaluaciones analíticas. Para evaluar de manera más confiable el potencial que ofrece el Cómputo en la Frontera, se requieren pruebas y ensayos bajo supuestos más complejos, heterogéneos y realistas (Pacheco et al., 2018b). Además, el Cómputo en la Frontera cuenta con sus propias limitaciones, tales como las restricciones en cuanto a poder de cómputo, batería y capacidad de almacenamiento de los dispositivos actuales (Mach & Becvar, 2017). Otra alternativa de solución es la Computación en la Nube que se ha convertido en la plataforma por defecto para aplicaciones de Aprendizaje Automático. Sin embargo, estos modelos de computación parcialmente centralizados no están calificados para aplicaciones dispersas de gran volumen de datos en tiempo real, donde los flujos de video pueden capturarse desde múltiples ubicaciones. También tiene otras desventajas importantes como la latencia de respuesta de la red, la seguridad del sistema y la privacidad de los datos (Herrera, Demir, Yousefi, Prevost, & Rad, 2018).

El Cómputo en la Niebla combina algunas ventajas importantes del Cómputo en la Nube y la Frontera, por el hecho de estar relativamente cerca de donde se generan los datos, es posible atender procesos en tiempo real de manera menos eficiente que en la frontera de la red, pero

puede llegar a tener más poder de cómputo que los dispositivos disponibles en la frontera de la red (Slingh, 2018; Mohammadi & Al-Fuqaha, 2018).

Desde el punto de vista de automatización del aula e iluminación inteligente (*smart lighting*), hay estudios que ubican a la iluminación como la segunda fuente más importante de gasto energético en edificios públicos, que de acuerdo con De Buen (2017), es posible lograr por lo menos un ahorro del 30% de consumo de energía eléctrica, al momento de reemplazar a luminarias con tecnología LED y ajustar su intensidad luminosa, además de la posibilidad de evitar posibles casos de desperdicio de energía, como al dejar las luces encendidas (Pacheco et al.,, 2018a). Por último, como parte del proyecto de investigación del laboratorio al cual pertenece este trabajo, fue integrado el trabajo de mejora de rendimiento en la ejecución de modelos pre-entrenados CNN en dispositivos IoT desarrollado por otro tesista (Flores, 2019).

1.3 Hipótesis.

En la actualidad, el Cómputo en la Nube ofrece una inmensa infraestructura disponible para las industrias y organizaciones de pequeña y gran escala para administrar, almacenar, procesar y ejecutar sus aplicaciones. Sin embargo, las nuevas necesidades de la industria, particularmente la industria 4.0, busca obtener resultados en menor tiempo manejando gran cantidad de datos y una de las herramientas que promete solucionar estos tipos de problemas son los algoritmos de IA. Sin embargo, ejecutar modelos de Aprendizaje Profundo en la Nube puede ser válido pero resulta impráctico, sobre todo por los tiempos de respuesta de la red al momento de transmitir grandes volúmenes de datos, lo cual imposibilita llevar a cabo acciones de control de forma inmediata para los dispositivos IoT que son obtenidos al momento de ejecutar los modelos de Aprendizaje Profundo, por ejemplo, para detectar personas a partir de flujos de imágenes de video tomados directamente desde una videocámara. El problema y objetivo del presente trabajo consiste en implementar un sistema IoT basado en una arquitectura de Cómputo Osmótico donde sea posible elegir la ejecución de distintos modelos de Aprendizaje Profundo de detección de objetos para realizar su etapa de inferencia entre diversos dispositivos como nodos FEC caracterizados por contar con limitados recursos de cómputo como alternativa o complemento

al esquema basado en Computo en la Nube considerando como requerimiento que sea posible efectuar un control de los dispositivos IoT en tiempo real para un aula inteligente.

Para el presente trabajo de tesis, se ha planteado la siguiente hipótesis:

"Dado un sistema IoT, con una implementación de una arquitectura de Cómputo en la Nube y dispositivos IoT (CIoT), es posible mejorar los tiempos de respuesta, si se incorpora a dicha arquitectura las capas de Cómputo en la Niebla y Cómputo en la Frontera (FEC) para realizar procesos de inferencia de un modelo de redes neuronales convolucionales dentro de los nodos FEC para el reconocimiento de objetos a partir del flujo de imágenes de video".

1.4 Requerimientos del Sistema

En esta sección se presentan los requerimientos del sistema IoT acotados al desarrollo de la presente tesis. Estos mismos van desde el planteamiento de los dispositivos a utilizar hasta los requisitos mínimos para implementar un sistema de un aula inteligente, considerando un escenario de validación de las distintas configuraciones para una arquitectura IoT osmótica para ejecutar modelos CNN dentro de distintos nodos y dispositivos disponibles.

1.4.1 Un Aula Inteligente como Escenario de Prueba.

Para el escenario de prueba del aula inteligente debe ser factible incorporar distintos dispositivos o servicios de cómputo capaces de ejecutar los mismos modelos CNN en cada uno de los distintos niveles de la arquitectura de Cómputo Osmótico. Se seleccionó un aula tradicional de clases como escenario de pruebas debido a la disponibilidad, grado de innovación y potencial de automatización para iluminación, climatización y control de acceso y poder escalar en el futuro a varias aulas, o incluso hasta un edificio inteligente (Pacheco et al., 2018a).

1.4.2 Dispositivos IoT del Aula Inteligente.

Entre las diversas alternativas de implementación de un aula inteligente, se utilizará una videocámara para detectar personas y controlar los dispositivos IoT del aula para reducir el consumo de energía eléctrica y mejorar el grado de confort para los usuarios. El algoritmo que se pretende desarrollar para el control de los dispositivos IoT del aula dependerá de qué dispositivo se quiere controlar. En el caso de la iluminación se busca que, si una persona es detectada, vía modelos CNN, en una de las cuatro zonas dentro del campo de visión de una cámara de video instalada en el aula de manera fija, se encenderá la luminaria correspondiente a la zona, como se observa en la figura 1.2. Con el termostato se busca que si se encuentran personas en el aula se encienda el aire acondicionado (AC) y si llega a dejar de detectar personas y el AC está encendido éste lo apaga. El proyector deberá encender cuando detecte a una persona acercándose a la zona de proyección por la zona 4 y se apagará cuando se retire de la zona 4 (véase figura 1.2).

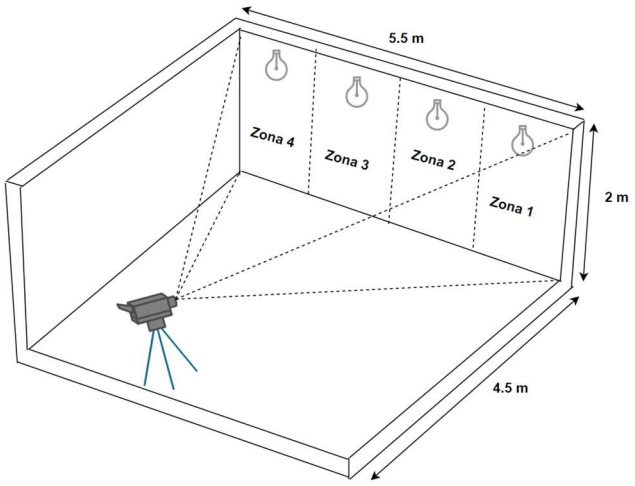

Figura I.2 Escenario de validación para el control de dispositivos IoT

1.4.3 Detección de Personas dentro del Aula Inteligente.

El control de dispositivos IoT se limita a la detección de personas analizando secuencias de video en tiempo real aplicando arquitecturas IoT y modelos de CNN pre-entrenados para detectar objetos (Pacheco et al., 2018b). Se deben revisar y evaluar diversos modelos CNN de detección de personas en tiempo real. Para el sistema del aula inteligente es un requisito

importante lograr un mínimo de 2 cuadros por segundo o *frames per second*, *fps* (Nikouei et al., 2018), considerando el tiempo de procesamiento requerido por los modelos CNN y el algoritmo de control de los dispositivos IoT.

Otro requerimiento importante en cuanto al desarrollo del software consiste en usar, adaptar o construir, en la medida de lo posible, código de forma modular, configurable y bien documentado ya sea conformando una librería o API y/o recurriendo a esquemas de virtualización del hardware para dispositivos IoT donde sea factible migrar a distintas plataformas y diferentes capas dentro de la arquitectura osmótica de IoT. En este trabajo no se consideraron elementos de seguridad, acceso y encriptado dentro del sistema, como por ejemplo la implementación de un algoritmo de encriptación de los datos que se envían a los distintos dispositivos que están integrados a la red.

1.4.4 Selección del Modelo CNN para la Detección de Personas.

Para el proceso de selección del modelo CNN se realizó una revisión de la literatura más reciente para encontrar algoritmos y modelos CNN de detección de objetos que contarán con la capacidad de detectar personas y de identificar el área dentro de la imagen donde se detectó a cada persona (Flores, 2019). Además, dichos modelos CNN deberían ser compatibles con las plataformas de dispositivos IoT y los distintos *frameworks* de desarrollo para soportar su ejecución en los dispositivos seleccionados para las tres capas de la arquitectura osmótica y así poder realizar las pruebas correspondientes.

1.4.5 Ejecución del Modelo CNN en las Capas de la Arquitectura.

Para validar la portabilidad y comparar el rendimiento del modelo CNN en distintas plataformas es indispensable implementar el sistema del aula inteligente en al menos un nodo basado en una de las capas de la arquitectura de Cómputo Osmótico, las cuales son CC, FC y EC, con o sin unidades de aceleración NPUs (*Neural Processing Unit*). La finalidad es evaluar y comparar la velocidad de ejecución de un mismo modelo CNN en cada capa de la arquitectura incluyendo la determinación de las distintas modalidades de ejecución, es decir, si están basadas

en CPU, GPU (*Graphics Processing Unit*) y/o acelerador neuronal (NPU) y mostrar cuál se ajusta mejor dentro del sistema de control desarrollado para el aula inteligente. Los dispositivos o servicios que se usarón en las capas de la arquitectura tienen que cumplir algunos requisitos. Para el dispositivo del nodo EC debe ser un dispositivo de recursos limitados con microprocesador ARM y con alrededor de 4 GB de memoria o menos, para los dispositivos de los nodos FC puede incluir el mismo dispositivo del nodo EC o una computadora PC convencional con un procesador Intel x64 de uno o dos núcleos y con una memoria de 16 GB o menos. Y para CC es necesario un servicio en la Nube capaz de almacenar y ejecutar un proyecto desarrollado en Python para correr modelos CNN para la detección de objetos y el *hardware* para este caso sólo depende de la disponibilidad del proveedor.

1.5 Metodología

Para el desarrollo del presente trabajo de tesis se establecieron y ejecutaron un conjunto de actividades encaminadas a lograr los objetivos del trabajo de forma gradual y sistemática, las cuales se agrupan en las siete etapas que se enumeran a continuación:

- Investigación del estado del arte y trabajos relacionados.
- Selección de equipo, aceleradores y dispositivos IoT.
- Selección y evaluación de modelos CNN para detección de objetos.
- Habilitación de la red osmótica de dispositivos IoT.
- Desarrollo de un API para establecer nodos FEC capaces de ejecutar modelos CNN.
- Mediciones y pruebas comparativas de los distintos nodos de Cómputo en la Nube, Niebla y Frontera.
- Validación de la hipótesis y documentación final del sistema.

La primera etapa del trabajo consistió en buscar trabajos relacionados acerca de arquitecturas para el Internet de las Cosas, Cómputo Osmótico, Cómputo en la Nube, la Niebla y la Frontera, con el fin de tener evidencias y conocimiento de diseños, tecnologías y desarrollos similares al sistema propuesto. En la segunda etapa se eligieron, instalaron y probaron los distintos

dispositivos, *frameworks* y servicios a integrar durante el desarrollo del sistema de aula inteligente aplicando arquitecturas CC, FC y EC. En el caso de EC se necesitó de equipo adicional para correr modelos CNN por lo que también se buscaron procesadores neuronales compatibles con los modelos CNN elegidos y los equipos seleccionados anteriormente. Una vez seleccionados los dispositivos en la tercera etapa, se determinó el modelo CNN apropiado y compatible con todos los dispositivos y que fuese capaz de reconocer personas en tiempo real, para ello se realizaron diversas pruebas para observar el rendimiento (*fps*) de cada uno de ellos.

La cuarta etapa consistió en buscar los sensores y actuadores habilitados como dispositivos IoT integrados dentro del aula. Se investigó además a detalle acerca de los protocolos de comunicación más adecuados para interconectar en red todos los dispositivos IoT a manejar y controlar. En la quinta etapa, considerando la experiencia de las etapas anteriores, se procedió a modularizar y estructurar dentro de una librería del proyecto integrando buena parte del código desarrollado para la interconexión, adquisición de video, ejecución de modelos CNN bajo distintas configuraciones y los algoritmos de control para proponer un API para implementar un aula inteligente incorporando internamente los *frameworks* necesarios para ejecutar modelos CNN acelerados junto con las funciones necesarias para controlar los dispositivos IoT.

La sexta etapa corresponde a la implementación de las funciones necesarias para correr el sistema completo del aula inteligente considerando cada uno de los dispositivos y las configuraciones necesarias para soportar las distintas capas de una arquitectura osmótica de IoT.

Por último, en la séptima etapa se consideraron las pruebas correspondientes para cada una de los distintos escenarios y capas de la arquitectura osmótica de IoT (CC, FC y EC) para validar el sistema y determinar los parámetros que tienen un mayor impacto en el rendimiento y funcionalidad apropiada del sistema de control del aula inteligente.

II. CONTEXTO, ÁREA DE INVESTIGACIÓN Y TRABAJOS RELACIONADOS

El Internet de las Cosas (IoT) es considerado una de las tecnologías disruptivas que está promoviendo el uso de sensores y distintos dispositivos capaces de comunicarse entre ellos para automatizar los trabajos, ya sea en la industria, en la sociedad o en la academia (Ibarra-Esquer et al., 2017). La automatización tiene el potencial de acrecentar la economía mundial hasta $15.7 mil millones de dólares para 2030 (Martinho-Truswell et al., 2018). Esto es equivalente a un aumento del 14% en el PIB mundial. Además, solamente considerando tecnologías con base en aprendizaje profundo, se estima una generación anual para los próximos años de entre $3.5 y $5.8 billones de dólares (Chui et al., 2018). La distribución de estos beneficios económicos entre los países dependerá de factores como la velocidad de adopción dentro del sector privado, la definición de estrategias de automatización en las economías sectoriales, además de las políticas gubernamentales para apoyar la innovación, investigación y desarrollo tecnológico (Martinho-Truswell et al., 2018).

2.1 El Internet de las cosas

El IoT es una tecnología emergente que se le describe como una infraestructura de red global dinámica con capacidades de autoconfiguración basadas en protocolos de comunicación estándar e interoperables donde las 'cosas' virtuales y físicas tienen identidades, atributos físicos y personalidades virtuales y utilizan interfaces inteligentes y se integran perfectamente en la red de información (Xu et al., 2014).

En los últimos años existe un aumento en la creación de nuevos dispositivos electrónicos capaces de poder comunicarse a través de internet y además de poder interactuar con otros dispositivos, esto nos habla del impacto que tiene el IoT en la sociedad relacionados con la vida cotidiana (Suwimon Vongsingthong & Smanchat, 2014). Podemos observar que el internet se ha vuelto algo cotidiano en nuestras vidas e incluso hemos visto cómo desde un dispositivo móvil es capaz de comunicarse con una televisión para mostrar fotos, videos, música e incluso usar el dispositivo móvil como control remoto para la televisión. No es de sorprender que el crecimiento del internet esté presente no solo en los hogares sino también en el sector industrial

que siempre está en la búsqueda de desarrollar nuevas y mejores técnicas que logren una mayor eficiencia y facilidad en los procesos de manufactura o de producción. Existe desde luego, una controversia acerca de que los dispositivos conectados almacenan datos de nuestra vida diaria, y de no tener bien protegida la red es muy posible que un individuo con intenciones delictivas puede usurpar los datos y usarlos en contra del usuario, en otras palabras, sería más fácil interferir en la privacidad del usuario; por lo que, tanto social como jurídicamente este servicio presenta nuevos retos para su desarrollo (Xu et al., 2014; Al-Gumaei et al., 2018).

2.2 Edificios Inteligentes

Un Edificio Inteligente (del inglés, *Smart Building*) se define como un edificio con la capacidad de monitorear e integrar las diferentes condiciones de sus infraestructuras principales, comunicaciones, servicios de agua y energía, con la cualidad de poder optimizar sus recursos, planear mantenimientos y monitorear aspectos de seguridad. Además, con el sistema de monitoreo avanzado y los sensores inteligentes incorporados, los datos pueden ser recolectados y evaluados en tiempo real, mejorando la toma de decisiones de la administración del edificio. Es así como este concepto plantea mejorar el uso de recursos e infraestructura disponible de una ciudad de manera sostenible (H. Song, 2017).

Las áreas que también han tomado fuerza y se han impulsado por el crecimiento mismo de la IA, es la de Ciudades, Edificios y Casas Inteligentes (CECI) (Pacheco et al., 2018). Las técnicas de IA y ML muestran capacidades probadas para aprender a partir de conjuntos de datos heterogéneos. Dichas técnicas pueden identificar patrones o tendencias que existen en los datos y extraer conocimiento de desempeño crucial, hacer predicciones precisas de estados futuros del sistema e identificar escenarios anómalos que pueden conducir a un comportamiento óptimo debido a fallas (Mohammadi & Al-Fuqaha, 2018). Estos enfoques se pueden utilizar eficazmente para tareas que van desde la gestión de la energía del edificio, la adaptación a la seguridad, garantía de la información y resistencia de dichos sistemas (Manic et al., 2016). Esto con los objetivos de mejorar el confort, ahorrar energía, tiempo y dinero, además de garantizar la seguridad y protección para los habitantes (Wilson, Hargreaves, & Hauxwell-Baldwin, 2017).

Un concepto clave de los edificios inteligentes es la generación de datos y existen algunos desafíos que incluyen el reciclaje de datos, el muestreo eficiente y el diseño de modelos escalables. La Ciencia de los Datos (*Data Science*) y Grandes Datos (*Big Data*) ofrecen métodos de vanguardia que pueden adoptarse para los edificios inteligentes para el análisis de datos, y aborda los desafíos presentados y resalta la posición del marco en varios dominios de aplicaciones de edificios inteligentes. Finalmente, al incorporar el aprendizaje automático con lo datos obtenidos del *Big Data* se pueden realizar nuevos servicios automatizados para edificios inteligentes (Manic et al., 2016).

2.3 Arquitecturas IoT y sus paradigmas

Las arquitecturas son un marco de tecnología que permiten que las cosas se interconectan e interactúen con objetos similares o diferentes imponiendo al ser humano como una capa sobre él. De hecho, está claro que el paradigma actual de IoT, que apoya las comunicaciones máquina-máquina (Machine to machine, M2M), ahora se ve limitado por una serie de factores (Ray, 2018). Las arquitecturas tradicionales del CC simplemente no fueron diseñadas para IoT, no tuvieron en mente la distribución geográfica, heterogeneidad y dinamismo en mente. Como tal, se requiere un enfoque novedoso para cumplir los requisitos de IoT, incluidos los requisitos de escalabilidad, interoperabilidad, flexibilidad, confiabilidad, eficiencia, disponibilidad y seguridad. Varios autores proponen una arquitectura capaz de proporcionar administración de datos y/o servicios de comunicaciones para facilitar la ejecución de aplicaciones IoT relevantes (Buyya & Srirama, 2019). Estas propuestas son diferentes y cada una tiene sus limitaciones y ventajas, pero este trabajo, como se mencionó en el primer capítulo, se enfoca en la integración de los tres paradigmas el CC, CF y EC desde una perspectiva orientada a IoT.

Actualmente, los sistemas IoT de las grandes corporaciones se soportan en grandes infraestructuras disponibles en el Cómputo en la Nube (CloT) en la que los objetos físicos están representados en forma de recursos web que son administrados por los servidores en el Internet global (Preden et al., 2015). Fundamentalmente, para interconectar las entidades físicas a Internet, el sistema utilizará varios dispositivos IoT, como sensores cableados o inalámbricos, actuadores y lectores para interactuar con ellos. Además, los dispositivos IoT tienen la

conectividad a Internet a través de los nodos de la puerta de enlace media tales como módems de Internet, enrutadores, conmutadores, estaciones de celulares, etc. En general, el sistema IoT común involucra tres tecnologías principales: sistemas integrados, *middleware* y servicios en la Nube, donde los sistemas integrados proporcionan inteligencia a los dispositivos IoT, el *middleware* interconecta los sistemas integrados heterogéneos de dispositivos frontales a la nube y, por último, la Nube proporciona mecanismos integrales de almacenamiento, procesamiento y administración. Aunque el modelo CloT es un enfoque común para implementar sistemas IoT, enfrenta los crecientes desafíos en IoT. Específicamente, CloT enfrenta desafíos en: ancho de banda, latencia, restricción de recursos y seguridad (Chiang & Zhang, 2016).

Los problemas con el ancho de banda son debido al flujo masivo y continuo de datos producidos por los dispositivos IoT lo que puede rápidamente exceder la disponibilidad de ancho de banda. Por ejemplo, un automóvil conectado puede generar decenas de megabytes de datos por segundo para la información de su ruta, velocidades, estado de funcionamiento del automóvil, condición del conductor, entorno circundante, clima, etc. Por lo tanto, confiar en la Nube distante para gestionar las cosas se vuelve poco práctico (Al-Qamash et al., 2018). El CIoT se enfrenta a los desafíos de lograr el requisito de controlar la latencia de extremo a extremo en decenas de milisegundos. Diversas áreas necesitan la información en tiempo real como los sistemas industriales de redes inteligentes, las redes vehiculares autónomas, las aplicaciones de realidad virtual y aumentada, las aplicaciones de comercio financiero en tiempo real, la atención médica y las aplicaciones de atención de personas mayores no pueden permitirse las causas derivadas de la latencia de CIoT (Buyya & Srirama, 2019).

Comúnmente, muchos dispositivos IoT tienen recursos de cómputo limitados en los que no pueden realizar tareas computacionales complejas y, por lo tanto, los sistemas CloT generalmente requieren dispositivos IoT para transmitir continuamente sus datos a la Nube. Sin embargo, tal diseño no es práctico en muchos dispositivos que funcionan con energía de la batería porque la transmisión de datos de extremo a extremo a través de Internet aún puede consumir una gran cantidad de energía (Mach & Becvar, 2017). Es posible que una gran cantidad de dispositivos IoT con restricciones pueden no tener los recursos suficientes para

protegerse de los ataques informáticos. El atacante podría dañar o controlar el dispositivo de IoT y enviar datos falsos a la Nube.

Estas particularidades propias de los sistemas CIoT están limitando la solución a muchos problemas relacionados con IoT en las empresas por lo que ha crecido el número de artículos de estos paradigmas de cómputo y ha llevado a que varias empresas se unieran y crearan un consorcio, enfocándose principalmente en el FC, llamada OpenFog. Esta comunidad busca realizar una arquitectura de FC y estandarizar los protocolos de comunicación para que sin importar el equipo de alguna marca este pueda conectarse con otros dispositivos y compartir los recursos de procesamiento con el fin de soportar la cantidad enorme de datos que se producen con los dispositivos de IoT (*IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*, 2018). En este consorcio basa su arquitectura en 8 pilares: seguridad, escalabilidad, código abierto, autonomía, RAS (fiabilidad, disponibilidad y facilidad de servicio), agilidad, jerarquía y programabilidad mostrados en la figura 2.1.

Figura II.1 Pilares de la arquitectura OpenFog.

2.4 Trabajos Relacionados

Las nuevas tecnologías de la computación provocan que el IoT y la IA tomen más fuerza y se creen varios artículos al respecto sobre cómo crear sistemas automatizados cada vez más sofisticados y cómo nos beneficia al usarlos en algunas áreas en particular. Existen algunos artículos que se relacionan con este trabajo de investigación y en este capítulo hablaremos un poco de ellos, se explicará en qué son similares, así como otras perspectivas y posibilidades sobre cómo abordar el problema e hipótesis planteada en el presente trabajo.

Nikouei et al. (2018) presentan una aplicación CECI de vigilancia, destacando su observación sobre el problema de trabajar con video, debido a que representa una carga intensiva para las comunicaciones vía red, por lo que resulta importante y recomendable realizar al menos parte del procesamiento del video antes de transferirlo a la Nube. Utilizan un modelo DL para detección de objetos llamado *Single Shot Multi-Box Detector* (SSD), con la arquitectura MobileNet como clasificador base. Los autores mencionan que los sistemas de vigilancia inteligentes se adaptan de manera efectiva con una arquitectura jerárquica de frontera-niebla-Nube, de modo que su prototipo realiza el cómputo en la frontera usando un nodo implementado en una Raspberry Pi 3, con la cual alcanzan un promedio de 1.79 *fps* y un máximo de 2.06 *fps*. Además, afirman que, rastrear un automóvil en tiempo real requiere una velocidad mucho mayor que rastrear a un peatón en tiempo real, por lo que para rastrear a un peatón los 2 *fps* resultan ser suficientes para desarrollar un sistema funcional.

Mocanu et al., 2017 presentan un artículo donde se explora en el contexto de la red inteligente los beneficios de utilizar Deep Reinforcement Learning, un tipo híbrido de métodos que combina el aprendizaje reforzado o Reinforcement Learning con Deep Learning, para realizar la optimización en línea de los horarios para la construcción de sistemas de gestión de energía. El procedimiento de aprendizaje se exploró utilizando dos métodos, Deep Q-learning y Deep Policy Gradient, extendiéndose ambos para realizar múltiples acciones simultáneamente. El enfoque propuesto fue validado en la base de datos a gran escala de Pecan Street Inc. Esta base de datos altamente dimensional incluye información sobre generación de energía fotovoltaica, vehículos eléctricos y electrodomésticos. Además, estas estrategias de

programación de energía en línea podrían utilizarse para proporcionar información en tiempo real a los consumidores para fomentar un uso más eficiente de la electricidad.

Otro trabajo (Al-Qamash et al., 2018) habla de los paradigmas de CC, FC y EC para organizaciones basadas en datos con el fin de facilitar el cálculo y el procesamiento de datos de una manera más sencilla. En este trabajo presentan una comparación con el fin de clasificar las aplicaciones del mundo real de acuerdo con los paradigmas que mejor satisfagan los fundamentos de cada aplicación y sus objetivos. Las aplicaciones que se discuten en el artículo se clasificaron en tres subcategorías: aplicaciones en tiempo real, casi en tiempo real y no en tiempo real. Según las comparaciones, la computación en la Nube es muy adecuada para aplicaciones en tiempo no real, como el comercio móvil y el aprendizaje, ya que no requieren movilidad, localización o respuesta en tiempo real, por lo que los recursos de la Nube se pueden utilizar por completo en estas aplicaciones con menos costo. Por otro lado, se descubrió que el FC y EC era la más adecuada para el tiempo casi real, como vehículos y redes inteligentes, y aplicaciones en tiempo real como usos relacionados con video, juegos y atención médica, respectivamente. También reconoce que los tres paradigmas se complementan entre sí y que sus arquitecturas combinadas pueden usarse para satisfacer una gran cantidad de aplicaciones del mundo real.

El trabajo de (Chen et al., 2018) realizó un sistema en aulas de campus reales para el control de AC basado en la ocupación del aula con el uso de un algoritmo líder de detección de objetos basado en CNN, YOLOv3. El programa empleado para realizar el control del AC lo llamaron iOccupancy y con YOLOv3 se realizó el conteo de ocupación utilizando imágenes de cámaras de vigilancia. Aunque las pruebas realizadas con el detector de objetos YOLOv3 solo logró una precisión de alrededor del 60% en aulas grandes, con una sola cámara ubicada en la parte posterior de las aulas, lo que indica que podría necesitar más de una cámara para uso práctico. También identificaron oportunidades de ahorro de energía usando este sistema en comparación de una programación de horario para el uso del AC.

El trabajo de Carpio et al., 2020 realizó un estudio comparativo entre el CC, EC y una combinación de ambos. Diseñaron una arquitectura de CC y EC modular utilizando los últimos

avances en plataformas de contenedores y herramientas de código abierto y mostraron los desafíos y la importancia de la latencia de evaluación comparativa, la escalabilidad y la utilización de recursos en el EC. Las mediciones de las pruebas mostraron que, si bien el rendimiento de solo EC supera a otros casos en términos de latencia y escalabilidad, también requiere que la aplicación funcione con datos ubicados en nodos de EC centralizados. La combinación de ambos funciona alrededor de 10 veces mejor en comparación con solo el CC hasta cierto número de procesos simultáneos en los que el sistema ya no se escala y CC funciona mejor. Por lo tanto, la solución de la combinación de CC y EC es una mejor opción hasta que haya cierto número de usuarios concurrentes en el sistema y cierta cantidad de datos; por encima de ese punto, existe una clara desventaja en comparación con la CC únicamente.

III. MARCO TEÓRICO

Se han introducido paradigmas de cómputo como el CC, FC y EC para organizaciones basadas en datos con el fin de facilitar el cálculo y el manejo de datos de una manera más sencilla. Teniendo en cuenta que ha habido un gran aumento en la cantidad de datos producidos en los últimos dos años, y se espera que la cantidad de datos producidos crezca exponencialmente, las investigaciones recientes se han centrado en utilizar estos paradigmas para satisfacer la creciente demanda. de rápida computación y almacenamiento de datos (Al-Qamash et al., 2018). La IA es una herramienta que se utiliza para el manejo de estos extensos datos, pero su funcionamiento en estos paradigmas de cómputo aún es incierto cada uno posee ventajas y desventajas por lo que también se han desarrollado nuevas arquitecturas que implementan la combinación de estos tres paradigmas como el Cómputo Osmótico para obtener la mayor ventaja según sea la aplicación. Algo que realiza para superar las diferencias de hardware entre estos paradigmas es la abstracción de la aplicación en un microservicio que es capaz de armonizar las capacidades de estos paradigmas, pero descritas de una manera diferente (Filocamo et al., 2018). A continuación, para entender mejor estas áreas de investigación se describen cada una de ellas para establecer el marco teórico del trabajo de tesis.

3.1 Cómputo Osmótico

Dentro del estado del arte en la literatura de IoT se menciona la Computación Osmótica (del inglés, *Osmotic Computing*) como un nuevo paradigma que ha cobrado impulso gracias al significativo aumento en la capacidad de los dispositivos IoT ubicados en el borde de la red (EC), junto con el soporte de protocolos para la transferencia de datos, mismos que permiten que dichos dispositivos interactúen más fluidamente. En Química, la "ósmosis" representa la difusión perfecta de las moléculas de una solución de mayor concentración a una más baja, para ilustrar de forma metafórica como los datos se distribuyen dentro de una red de dispositivos IoT y las distintas capas CC-FC-EC. Esto va más allá de la simple administración elástica de los recursos implementados, ya que las estrategias de implementación están relacionadas con los requisitos de infraestructura, como el equilibrio de carga, la confiabilidad y la disponibilidad. Existen diversas arquitecturas alternas como son Cloud + Edge + IoT o más recientes como Cloud + Fog + IoT (Morshed et al., 2017; Villari et al., 2016).

3.2 Cómputo en la Nube

El rápido crecimiento en el desarrollo de procesamiento y almacenamiento de datos, el incremento en la velocidad del Internet, la mayor potencia y el bajo costo de los recursos informáticos, permitieron la creación de un nuevo modelo de computación llamado Computación en la Nube (CC), del inglés *Cloud Computing* (Zhang et al., 2010). Los servicios del CC fueron diseñados para resolver los problemas comunes con los servicios de cómputo y poder acceder a las herramientas a través de la red con el objetivo de mantener todo el poder informático y datos en un solo lugar "en la Nube" y así poder usar los servicios que el cliente necesita de forma segura, flexible y adaptativa (Buyya & Srirama, 2019). Un servicio en la nube proporciona acceso remoto a los datos o archivos desde cualquier lugar, que es algo que una empresa, con varias plantas alrededor del mundo, llega a necesitar. El significado de la nube cubre todos los servicios en línea que un proveedor puede ofrecer como el almacenamiento de datos, análisis de datos, bases de datos, modelos de ML, entre muchos otros. Según el NIST (National Institute of Standards and Technology) el modelo de la nube se compone por 5 características esenciales los cuales son: autoservicio sobre demanda, amplio acceso a la red, infraestructura escalable, previsión de servicios auditables y monetizables (Mell & Grance, 2011).

NIST propone tres diferentes modelos de servicios en la nube, Software como Servicio (SaaS), Plataforma como Servicio (PaaS) e Infraestructura como Servicio (IaaS). SaaS se refiere a aplicaciones basadas en la Nube, como el correo electrónico desde un navegador web, Facebook, Google, YouTube, Yahoo, Twitter son ejemplos de SaaS donde al usuario se le permite enviar y recibir mensajes o buscar información, páginas web, imágenes o videos sin tener que instalar o administrar las características del programa ni mantener los servidores y los sistemas en los que ejecuta los programas. PaaS permite que el usuario pueda implementar y administrar su aplicación, así el consumidor no se preocupa por el mantenimiento de los servidores ni de las complejas tareas que se emplean para ejecutar su aplicación. En IaaS el cliente renta el hardware para correr cualquier software incluso cualquier sistema operativo y posiblemente el control de la seguridad de los componentes de red como los *firewalls* (Geewax & Hölzle, 2018).

Una de las nuevas herramientas con mayor auge en la actualidad para complementar muchos servicios en la Nube es precisamente el aprendizaje automático (ML) que incluye servicios tales como el reconocimiento de texto en imágenes, reconocimiento de voz, reconocimiento de caras o personas en videos o imágenes, e incluso herramientas para el análisis del lenguaje natural. Estas herramientas de aprendizaje automático cada vez toman más interés para las empresas para resolver los problemas con el servicio al cliente o para obtener mejores servicios para ofrecer al cliente, por ejemplo, correctores de ortografía, correo electrónico no deseado, sugerencias de productos ofertados y personalizados. Los principales proveedores que explotan este tipo de servicio en sus productos son Google Cloud, Amazon Web Service (AWS), IBM Cloud y Microsoft Azure (Zhang et al., 2010)

3.3 Cómputo en la Niebla

El CC juega un papel importante en el procesamiento de una gran cantidad de datos, sin embargo, el aumento de dispositivos conectados a la red genera una gran cantidad de datos que deben procesarse y surgen algunos problemas como una sigue habiendo problemas sin resolver como la latencia, ausencia de soporte de movilidad (Sing et al., 2018; Yi et al., 2015). Mientras que el Cómputo en la Niebla (FC) es considerado una extensión del CC entre los dispositivos IoT al borde de la red. Es una plataforma virtualizada que distribuye el procesamiento, almacenamiento y servicios de redes entre los dispositivos que están a un salto del usuario y los servidores tradicionales de la Nube. El FC es también considerado una tecnología que puede auxiliar con los desafíos del procesamiento de grandes datos (Big Data) (Dar & Ravindran, 2019).

Cuando surge la idea de complementar el CC y poder distribuir los recursos de computación, redes y almacenamiento a las ubicaciones que estén más cerca de la fuente de los datos o de las aplicaciones del usuario final, se propuso el término alternativo de cómputo en la neblina, del inglés Mist Computing, que algunos trabajos han descrito como un subconjunto de la niebla y esencialmente la característica principal es otorgar a los dispositivos IoT la capacidad de auto-organización, auto-gestión y varios auto-mecanismos, con la finalidad de que los dispositivos IoT puedan funcionar continuamente incluso cuando la conexión a internet sea inestable (Preden

et al., 2015). Existen incluso otros autores que describen el FC y EC como equivalentes (Yi et al., 2015).

La industria ha invertido en el FC mediante el establecimiento de un consorcio llamado OpenFog (§2.3) fundado por ARM, Intel, Cisco, Dell, Microsoft y la universidad de Princeton. Además, cuenta con más de 60 miembros pertenecientes a la industria y la academia, lo cual demuestra que el cómputo en la niebla se ha convertido en una tendencia importante dentro de las tecnologías de la información y comunicaciones (TIC).

3.4 Cómputo en la Frontera

Recientemente ha emergido un cambio de paradigma en la forma en que se distribuyen las cargas de trabajo para el procesamiento de datos, desplazando parte del trabajo hacia los nodos ubicados en la frontera dentro de los ecosistemas IoT. A este paradigma se le denomina Cómputo en la Frontera (EC), del inglés, *Edge Computing (EC)* (Herrera et al., 2018), que pretende reducir los tiempos de respuesta sin necesidad de transferir a la Nube todos los datos generados por los dispositivos, proporcionando un esquema de seguridad, al igual que proteger la privacidad del usuario. Sin embargo, uno de los obstáculos centrales del procesamiento de datos dentro de los dispositivos IoT es el incremento en el consumo de batería y su limitada capacidad de recursos de cómputo (memoria, almacenamiento, CPU, etc.) que aún plantea un obstáculo significativo restringiendo su uso en aplicaciones IoT altamente exigentes (Mach & Becvar, 2017).

3.5 Cómputo en la Niebla y la Frontera complementando el Cómputo en la Nube

Cuando el FC requiere del soporte de las tecnologías de EC se le conoce como Cómputo Niebla-Frontera (FEC), del inglés *Fog and Edge Computing,* y puede proporcionar un complemento para el Cómputo en la Nube para IoT, al extender este modelo al borde de las redes de IoT donde los nodos intermedios de la red, como enrutadores, conmutadores, concentradores y dispositivos IOT, participan con el procesamiento de la información y toma

de decisiones para mejorar la brecha que existe entre IoT y CC y ofrecer un servicio continuo. FEC brinda nuevas oportunidades y también plantea nuevos desafíos en el desarrollo y la operación. Específicamente, el desarrollo enfrenta desafíos en complejidad y estandarización, lo que potencialmente conduce a la dificultad en la integración del sistema entre diferentes proveedores de FEC y los puntos finales del IOT. La industria es consciente de los desafíos y ha iniciado una serie de plataformas abiertas como WS02, loT, Apache Edgent. Además, el reciente proyecto de la Fundación Linux, EdgeX Foundry (edgexfoundry.org), ha demostrado que el interés industrial en IOT ya no está satisfecho con la conectividad entre dispositivos y la Nube. En cambio, la tendencia ha pasado de cosas conectadas a cosas cognitivas en las que los procesos y las decisiones se realizan lo más cerca posible de los objetos físicos, incluso a los dispositivos IOT en sí (Buyya & Srirama, 2019).

Esta sección describe las ventajas que ofrece FEC y aborda la cuestión de cómo lograr dichas ventajas. En particular, FEC ofrece cinco ventajas principales, que se pueden ejemplificar con SCALE: seguridad, cognición, agilidad, latencia y eficiencia (*IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*, 2018).

Seguridad

FEC brinda seguridad adicional a los dispositivos IoT para garantizar la seguridad y la confianza en las transacciones. Por ejemplo, los sensores inalámbricos de hoy en día implementados en entornos al aire libre a menudo requieren una actualización remota del código fuente inalámbrico para resolver los problemas relacionados con la seguridad. Sin embargo, debido a varios factores ambientales, como la intensidad de la señal inestable, las interrupciones, el ancho de banda, entre otros. El servidor central remoto puede enfrentar desafíos para realizar la actualización rápidamente y, por lo tanto, aumenta la posibilidad de un ataque de ciberseguridad durante dicha transacción. Por otro lado, si la infraestructura FEC está disponible, el *backend* (servidor) puede configurar la mejor opción de enrutamiento entre toda la red a través de varios nodos FEC para realizar rápidamente la actualización de seguridad del software para los sensores inalámbricos.

Autonomía (Servicios cognitivos)

FEC permite conocer los objetivos de sus clientes para respaldar la toma de decisiones autónomas en cuanto a dónde y cuándo implementar las funciones de computación, almacenamiento y control. Recientemente estos mecanismos se agrupan bajo el término conocido como procesamiento cognitivo de FEC, que involucra varios mecanismos en términos de auto adaptación, auto organización, auto recuperación, auto expresión, entre otros, transformando así el rol tradicional de dispositivos IOT pasivos a dispositivos inteligentes activos, que posibilita su operación continua y toma de decisiones basadas en el contexto actual y en los requisitos del cliente sin depender de la decisión de la lejana infraestructura en la Nube.

Agilidad

FEC mejora la agilidad de alto impacto de la implementación del sistema IoT. Esto es, FEC brinda la oportunidad a empresas individuales y pequeñas a participar en la prestación de servicios FEC utilizando las interfaces comunes de código abierto (APIs) o kits de desarrollo de software (SDK). Por ejemplo, el estándar MEC de ETSI (ETSI, 2018) y el modelo de negocio Indie Fog (Chang et al., 2017) acelerarán el despliegue de infraestructuras IOT de gran alcance.

Latencia

Parte de la filosofía del paradigma de FEC es proporcionar respuestas rápidas para las aplicaciones que requieren una latencia ultra baja. Específicamente, en muchas aplicaciones ubicuas o de automatización industrial, donde el sistema necesita recopilar y procesar los datos sensoriales continuamente en forma de flujo de datos (*streams*) para identificar cualquier evento y realizar acciones oportunas en tiempo real. De manera explícita, al aplicar FEC, estos sistemas son capaces de soportar funciones sensibles al tiempo. Además, la característica de virtualización FEC, donde el servidor central distante puede configurar completamente el comportamiento de dispositivos físicos utilizando la abstracción del software, lo cual proporciona una plataforma IoT altamente flexible para una rápida reconfiguración de los dispositivos IoT.

Eficiencia

FEC mejora la eficiencia de CIoT en términos de mejorar el rendimiento y reducir los costos innecesarios. Por ejemplo, al aplicar FEC, un sistema ubicuo de atención médica o de cuidado de personas mayores puede redistribuir una serie de tareas a los dispositivos por medio de una puerta de enlace a Internet (*gateway*) de los sensores de atención médica y utilizar dichos dispositivos de puerta de enlace para realizar las tareas de análisis de datos sensoriales (analítica de datos). Idealmente, dado que el proceso ocurre cerca de la fuente de datos, un sistema FEC puede generar resultados mucho más rápido que su equivalente CIoT. Además, dado que FEC utiliza dispositivos de puerta de enlace para realizar la mayoría de sus tareas, reduce en gran medida el costo y latencia adicional referente al ancho de banda requerido para la comunicación saliente hacia la Nube.

3.6 El Mecanismo SCANC de los Sistemas FEC

Como se describieron anteriormente las ventajas de alto nivel proporcionadas por FEC, es entonces que surge la pregunta: ¿Cómo proporciona FEC dichas ventajas? Para responder a la pregunta, describimos enseguida los cinco mecanismos básicos compatibles con los dispositivos IoT habilitados para FEC. Identificados bajo las siglas SCANC (Storage, Compute, Acceleration, Networking and Control), que corresponde a almacenamiento, cómputo, aceleración, redes y control (Preden et al., 2015; Buyya & Srirama, 2019).

3.6.1 Almacenamiento

El mecanismo de almacenamiento en FEC corresponde al almacenamiento temporal de datos y el almacenamiento en caché en los nodos FEC para mejorar el rendimiento de información o entrega de contenidos. Por ejemplo, los proveedores de servicios de contenido pueden realizar el almacenamiento en caché de contenido multimedia en los nodos FEC que están más cerca de sus clientes para mejorar la calidad de la experiencia (Mach & Becvar, 2017). Además, en

escenarios de vehículos autónomos pueden utilizar los nodos FEC en la carretera para buscar y compartir continuamente la información recogida por otros vehículos.

3.6.2 Cómputo

Los nodos FEC proporcionan los mecanismos de cómputo principalmente en dos modelos: infraestructura o plataforma como servicio (I/PaaS) y software como servicio (SaaS). En general, los proveedores de FEC ofrecen I/PaaS según dos enfoques: máquinas virtuales de hipervisor (VM) o motores de contenedores (CE), que permiten que las plataformas flexibles para los clientes de FEC implementen el software personalizado que necesitan en un entorno de espacio IAslado en los nodos de FEC. Además de I/PaaS, SaaS también promete la prestación de servicios FEC (*IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*, 2018). Recapitulando, los proveedores de SaaS pueden ofrecer dos tipos de servicios: procesamiento de datos bajo demanda (ODP) y contexto como servicio (CaaS). Específicamente, un servicio basado en ODP tiene métodos preinstalados que pueden procesar los datos enviados desde el cliente en la forma de solicitud/respuesta. Considerando además que, el servicio basado en CaaS proporciona un método personalizado de provisión de datos en el que los nodos FEC pueden recopilar y procesar los datos para generar información significativa para sus clientes (Buyya & Srirama, 2019).

3.6.3 Aceleración

Fundamentalmente, los nodos FEC admiten la aceleración en dos aspectos: aceleración de redes y aceleración de computación.

Aceleración de redes. Inicialmente, la mayoría de los operadores de red tienen su propia configuración para el enrutamiento de mensajes y además sus clientes no pueden solicitar sus propias tablas de enrutamiento personalizadas. Por ejemplo, un proveedor de servicios de Internet (ISP) en Europa del Este puede tener dos rutas de enrutamiento con una latencia diferente para llegar a un servidor web ubicado en Europa Central, y la ruta de acceso de un cliente se basa en la configuración de equilibrio de carga del ISP, que en muchos casos no es la

opción óptima para el cliente. Por otro lado, FEC admite un mecanismo de aceleración de red basado en tecnología de virtualización de red, que permite a los nodos FEC operar múltiples tablas de enrutamiento en paralelo y realizar una red definida por software (SDN). Por lo tanto, los clientes de los nodos FEC pueden configurar un enrutamiento personalizado para sus aplicaciones con el fin de lograr una velocidad de transmisión de red óptima.

Aceleración computacional. Los investigadores en cómputo en la niebla han previsto que los nodos FEC proporcionarán aceleración de computación utilizando unidades de procesamiento embebidas avanzadas, como unidades de procesamiento de gráficos (GPU) o unidades de arreglos de compuertas programables (FPGA) (*IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*, 2018). Específicamente, utilizan GPUs para mejorar el procesamiento de algoritmos complejos se ha convertido en un enfoque común en el CC en general. Por lo tanto, es previsible que los proveedores de FEC también puedan proporcionar el equipo que contiene GPU. Además, las unidades de FPGA permiten a los usuarios redistribuir los códigos de programa en ellos para mejorar o actualizar las funciones de los dispositivos host. En particular, los investigadores en tecnologías de sensores (Krasteva et al., 2011) han estado utilizando FPGA para la reconfiguración de sensores de tiempo de ejecución. Además, en comparación con las GPU, FPGA tiene el potencial de ser más eficiente energéticamente y permite a los clientes configurar su código personalizado dentro de los nodos FEC.

3.6.4 Redes de nodos FEC

Las redes de FEC involucran conectividad vertical y horizontal. Las redes verticales interconectan dispositivos IoT con la Nube con las redes IP; mientras que la red horizontal puede ser heterogénea en señales y protocolos de red dependiendo de la especificación de *hardware* soportada de los nodos FEC (Buyya & Srirama, 2019; Verba et al., 2017).

Redes verticales. Los nodos FEC habilitan la red vertical utilizando protocolos estándar basados en redes IP, tales como los sockets TCP/UDP, COAP (mensajería extensible basada en publicación-suscripción), Protocolo de presencia (XMPP), AMQP (*Advanced Message Queuing*

Protocol), MQTT (Transporte de telemetría de la fila de mensajes), y así sucesivamente (Naik, 2017). Específicamente, los dispositivos IoT pueden operar funciones del lado del servidor (por ejemplo, el servidor CoAP) que permiten que los nodos FEC, que actúan como el *proxy* de la Nube, recopilen datos y luego los envíen a la Nube. Además, los nodos FEC también pueden operar como el intermediario de mensajes del protocolo basado en publicación-suscripción que permite a los dispositivos IOT publicar flujos de datos a los nodos FEC y habilitar el *backend* de la Nube para suscribir los flujos de datos desde los nodos FEC.

Red horizontal. En función de diversos requisitos de optimización, como la eficiencia energética o la eficiencia de transmisión de la red, los sistemas IOT a menudo utilizan enfoques de red heterogéneos. En particular, las casas inteligente, las fábricas inteligentes y los vehículos conectados comúnmente utilizan Bluetooth, ZigBee (basado en IEEE 802.15.4) y Z-Wave en los dispositivos IOT y los conectan a una puerta de enlace de red IP para permitir la conectividad entre los dispositivos y la Nube. En general, los dispositivos de puerta de enlace de red IP son las entidades ideales para hospedar servidores FEC, ya que tienen la conectividad con los dispositivos IOT para varios tipos de señales. Por ejemplo, la Nube puede solicitar que un servidor FEC alojado en un automóvil conectado se comunique con el equipo IOT en la carretera utilizando ZigBee para recopilar la información ambiental necesaria para analizar la situación del tráfico en tiempo real.

3.6.5 Control

El mecanismo de control soportado por FEC consta de cuatro tipos básicos: implementación, actuación, mediación y seguridad (Buyya & Srirama, 2019; Naha et al., 2018).

El control de implementación permite que los clientes realicen dinámicamente la implementación del programa de software personalizable. Además, los clientes pueden configurar los nodos FEC para controlar qué programa debe ejecutar el nodo FEC y cuándo debe ejecutarlo (Yousefpour et al., 2019). Además, los proveedores de FEC también pueden proporcionar una topología de red FEC completa como un servicio que permite a los clientes

mover su programa de un nodo FEC a otro. Además, los clientes también pueden controlar múltiples nodos FEC para lograr el rendimiento óptimo para sus aplicaciones.

El control de actuación representa el mecanismo admitido por la especificación de hardware y las conectividades entre los nodos FEC y los dispositivos conectados. Específicamente, en lugar de realizar una interacción directa entre la Nube y los dispositivos, la Nube puede delegar ciertas decisiones a los nodos FEC para controlar directamente el comportamiento de los dispositivos IOT.

El control de mediación corresponde a la capacidad de FEC en términos de interactuar con entidades externas que son propiedad de diferentes partes. En particular, los vehículos conectados soportados por diferentes proveedores de servicios pueden comunicarse entre sí, aunque inicialmente pueden no tener un protocolo común. Con la función de software del nodo FEC, los vehículos pueden tener una actualización de software a pedido para mejorar su interoperabilidad.

El control de seguridad es el requisito básico de los nodos FEC que permite a los clientes controlar la autenticación, autorización, identidad y protección del entorno de tiempo de ejecución virtualizado operado en los nodos FEC.

3.7 Inteligencia Artificial

El cerebro es el órgano más importante del cuerpo del ser humano, y el más complejo. Con él es posible percibir la luz, el sonido, los sabores, y el tacto. Además, nos permite guardar experiencias de nuestra vida, sin estas capacidades seríamos unos seres primitivos incapaces de cuestionarnos el porqué de nuestra existencia o explicarnos cómo funciona el entorno donde vivimos. El cerebro es, sin temor a equivocarme, lo que nos hace inteligentes.

Desde la perspectiva de las Ciencias de la Computación, la Inteligencia Artificial (IA) se puede entender como una disciplina que busca aproximar sus resultados a los del razonamiento

humano a través de la organización y manipulación de datos (Singh, 2014). Sin embargo, la programación lógica ha sido suficiente para poder programar actividades tan triviales y cotidianas que realiza una persona en entornos que se encuentran en constante cambio con innumerables variables que deben tomarse en cuenta para que un mecanismo autónomo pueda funcionar correctamente.

3.8 Aprendizaje Automático

Existen varias áreas que entran a la IA como el Aprendizaje de Máquina o Aprendizaje Automático, del inglés *Machine Learning* (ML), que combina ideas de varias ramas de las ciencias, tales como Estadística, Teoría de la Información, Matemáticas e incluso Biología y Psicología para que una máquina pueda aprender de los datos seleccionados (Athmaja, Hanumanthappa, & Kavitha, 2017; Raschka & Mirjalili, 2017).

ML busca darle sentido a una gran cantidad de datos. Vivimos en una era donde los datos abundan, debido a las grandes cantidades de dispositivos que recolectan información y la comparten con otros equipos para interpretarla y tomar acciones. IA y ML abarcan una multitud de algoritmos y técnicas. Las técnicas tradicionales como la regresión logística, las redes neuronales artificiales (ANN), los sistemas de lógica difusa (FLS), las máquinas de vectores de soporte (SVM) y los diferentes modelos probabilísticos bayesianos son métodos bien documentados y se han utilizado ampliamente en una amplia gama de dominios para reconocer patrones y tendencias en los datos. Una de las técnicas que más ha llamado la atención en la última década es el Aprendizaje Profundo (DL), dando muchos avances principalmente en el área de visión por computadora (Manic et al., 2016).

3.9 Aprendizaje Profundo

El Aprendizaje Profundo, del inglés *Deep Learning* (DL), pretende asimilar el funcionamiento de las neuronas de un cerebro. Por medio del entrenamiento de una red neuronal, que se constituye por varias capas que contiene cierto número de nodos denominadas neuronas,

es capaz de obtener un resultado según los datos de entrada que se le suministraron a la red (Buduma & Lacascio, 2017).

En los últimos años ha habido un gran interés en DL, ya que ha revolucionado una multitud de campos, como el reconocimiento de voz, el procesamiento del lenguaje natural y aplicaciones de visión por computadora como el reconocimiento facial (Buduma & Lacascio, 2017; Manic et al., 2016). Esto debido a las ventajas que presenta en términos del desempeño y flexibilidad, ya que provee una efectiva herramienta para extraer características de alto nivel estructuradas de forma jerarquizada a partir de datos sensoriales de alta dimensión, lo cual es muy útil para tareas de clasificación y regresión. Los modelos DL están basados en el aprendizaje de representaciones a partir de datos sin procesar y su arquitectura contiene más de una capa oculta. La red neuronal se compone de muchas capas de nodos interconectados que son entrenados para el procesamiento de la información y la extracción de características, donde cada capa sucesiva utiliza la salida de la anterior como entrada. Entre los modelos más conocidos de aprendizaje profundo están Long Short Term Memory (LSTM), Convolutional Neural Network (CNN), Deep Belief Network (DBN) y los autoencoders (Buduma & Lacascio, 2017).

Una arquitectura CNN diseñada específicamente para trabajar con imágenes, son las Redes Neuronales Convolucionales (*Convolutional Neural Networks,* CNN) (Chui et al., 2018), las cuales son un tipo de red neuronal que utiliza al menos una capa de tipo convolucional, la cual es particularmente útil para tareas de reconocimiento y clasificación de objetos, entrenadas con conjuntos de imágenes (*datasets*) de entrenamiento (Fan, 2015 ; Hao, Zhang, & Ma, 2016). Además, en los últimos años ha habido un creciente interés por realizar análisis más complejos de imágenes y videos para clasificar los objetos que aparecen en cada imagen, y además determinar con precisión también las ubicaciones de dichos objetos (Agarwal, Terrail, & Jurie, 2018). Esta tarea se conoce como detección de objetos, con amplia aplicación, por ejemplo, en la detección de rostros y de peatones en las calles (Agarwal et al., 2018; Zhao, Zheng, Xu, & Wu, 2019).

3.10 Microservicios y Virtualización

Una aplicación basada en microservicios es una colección de servicios autónomos, cada uno de los cuales hace bien una cosa, que trabajan juntos para realizar operaciones más complejas. En lugar de un único sistema complejo, crea y administra un conjunto de servicios relativamente simples que pueden interactuar de formas complejas (Burns, 2018). Estos servicios colaboran entre sí a través de protocolos de mensajería independientes de la tecnología, ya sea punto a punto o asincrónicamente. Esto puede parecer una idea simple, pero tiene implicaciones sorprendentes para reducir la fricción en el desarrollo de sistemas complejos. La práctica clásica de la ingeniería de software aboga por una alta cohesión y un acoplamiento flexible como propiedades deseables de un sistema bien diseñado (Bruce & Pereira, 2019). Un sistema que tenga estas propiedades será más fácil de mantener y más maleable ante el cambio. La cohesión es el grado en que los elementos de un determinado módulo pertenecen juntos, mientras que el acoplamiento es el grado en que un elemento conoce el funcionamiento interno de otro.

En una aplicación basada en microservicios, el objetivo es lograr estas propiedades en el nivel de unidades de funcionalidad implementables de forma independiente. Un solo microservicio debe ser altamente cohesivo: debe ser responsable de alguna capacidad única dentro de una aplicación. Asimismo, cuanto menos sepa cada servicio sobre el funcionamiento interno de otros servicios, más fácil será realizar cambios en un servicio sin forzar cambios en otros. La virtualización de los sistemas ha sido un gran ejemplo para separar las aplicaciones de las infraestructuras de cómputo para poder agregar servicios de software rápidamente sin tener que preocuparse por los otros servicios ya implementados (Geewax & Hölzle, 2018). Sin embargo, es necesario haber diseñado bien el programa para que se puedan agregar más funciones al programa y para ello se ha extendido cada vez más el uso de las interfaces de programación de aplicaciones (API) para que los diferentes sistemas o usuarios que se quieren conectar con los servicios implementados no tengan algún problema de comunicación y lograr que el sistema sea simple de escalar para poder admitir más accesos a los servicios (Bruce & Pereira, 2019; Burns, 2018).

3.11 Diseño de arquitecturas de API

Los programas no solamente son usados por personas, otros programas hacen uso de las funciones, clases, métodos o librerías para crear sistemas distribuidos de software. Una API (*Application Programming Interface*) es una herramienta de comunicación entre programas que facilita la interacción entre dispositivos (Biehl, 2016). Tanto los desarrolladores, programas o sistemas que usan un API son "clientes", incluso si ellos son parte del mismo sistema o empresa que utilizan APIs internas. Cualquier decisión sobre cómo está conformada una API debe de hacerse de una manera que sea fácil de interpretar para los clientes (Hunter & Cockcroft, 2017).

3.11.1 Términos de Referencia para Arquitecturas en Red.

Antes de entrar a los estilos de arquitectura debemos entender algunos conceptos como HTTP, identificador de recursos uniforme (URI), recursos, interfaz uniforme HTTP, y representación (Fielding et al., 2007).

HTTP. Protocolo de transferencia de Hipertexto (*Hypertext Transport Protocol*) es el método que utilizan los navegadores para comunicarse con los servidores web. Cuando hace clic en un enlace web o escribe una dirección en una barra de URL, su navegador envía una solicitud HTTP al servidor y el servidor devuelve una respuesta HTTP. Cada solicitud HTTP debe tener un identificador; esta dirección se llama un indicador uniforme de recursos (URI) en este caso.

URI. Identificador uniforme de recursos es la dirección e identificador de los recursos de una red y su diferencia con URL Localización de recursos uniforme es que este puede variar con el tiempo ya que es el que identifica a la página web.

Recursos HTTP. Se le denomina recurso a una abstracción de la entidad HTTP, por ejemplo, un sitio web. Un recurso define la abstracción de la sintaxis de la entidad y la expresión concreta de un recurso es llamado una representación. Un recurso es direccionado por un URI y se puede interactuar con los recursos con una interfaz uniforme HTTP. Se le denomina uniforme porque es el mismo para todos los recursos HTTP.

Diversas Representaciones para APIs. Una representación es una entidad concreta, la cual codifica un recurso en HTML, JSON o XML. Un recurso puede estar disponible en múltiples representaciones y la ventaja es que se puede escoger la representación correcta basada en las capacidades de procesamiento del cliente.

Interfaces uniformes HTTP. Los recursos HTTP tiene las mismas Interfaces Uniformes HTTP que son utilizados para realizar operaciones en los recursos. Esta interfaz define operaciones CRUD (Create, Read, Update and Delete) y HTTP define estas operaciones como los métodos POST, GET, PUT y DELETE.

3.11.2 Estilos de arquitecturas para comunicar sistemas distribuidos

En general una arquitectura es una estructura de solución predefinida a gran escala. Con un estilo de arquitectura definida es más rápido y consistente crear una API. Algunos estilos de arquitecturas para la comunicación de sistemas distribuidos son el RPC, SOAP, REST y HATEOAS. (Burns, 2018)

RPC (*Remote Procedure Call*) es un estilo de arquitectura para sistemas distribuidos que data desde los años 80s. En la actualidad los estilos RPC más utilizados extensamente son JSON-RPC y XML-RPC. El concepto central del RPC es el procedimiento. Los procedimientos no necesitan correr en una máquina local, pero pueden ejecutarse en una máquina remota dentro de un sistema distribuido (Hunter & Cockcroft, 2017).

JSON-RPC es usado para llamar a un único procedimiento a una máquina remota. Cuando se serializa la solicitud o la respuesta utiliza un esquema JSON definido para JSON-RPC. Debe contener tres propiedades (Matt Morley, 2013):

- **Método:** una cadena con el nombre del método que se va a invocar.
- **Parámetros:** un objeto o matriz de valores que se pasarán como parámetros al método definido.
- **ID:** un valor de cualquier tipo utilizado para hacer coincidir la respuesta con la solicitud a la que responde.

XML-RPC es usado para llamar a un único método en un equipo remoto. Como el nombre lo dice este estilo utiliza XML para serializar el procedimiento de solicitud (`methodCall`) y respuesta (`methodResponse`). XML permite transportar complejas estructuras de datos. XML-RPC data desde 1998 y después evolucionó a SOAP (Hunter & Cockcroft, 2017).

SOAP es la abreviación del inglés *Simple Object Access Protocol* sigue el estilo de RPC y expone procedimientos como conceptos centrales, como por ejemplo, `getConsmer()`. La arquitectura SOAP estandarizada por la W3C, puede comunicarse por medio de intercambio de datos XML y es el protocolo que extensamente se ha utilizado para servicios web. SOAP puede estar vinculado a diferentes protocolos incluyendo HTTP, TCP, UDP y SMTP. Además, es muy adecuado para la integración en empresas, debido a su estructura rígida en cuestión de las capacidades de seguridad y autorización. Sin embargo, los servicios SOAP no ofrecen una visibilidad, ya que no se puede deducir información semántica sobre el método. SOAP ofrece varias extensiones, solo por mencionar algunos de ejemplo hay para transferir datos binarios, seguridad, encriptación y registros (W3C, 2004; Beihl, 2016).

REST (*Representational State Transfer*) es un estilo de arquitectura que define algunas restricciones y acuerdos. Es una de las arquitecturas más utilizadas para crear un API web y diseñada para hacer el óptimo uso de la infraestructura basada en HTTP y el protocolo HTTP (Hunter & Cockcroft, 2017). Una característica que destaca en REST con respecto a las demás arquitecturas es que está diseñada en torno a la idea de usar "sustantivos" en el sistema en lugar de "verbos". La tabla 3.1 muestra una comparación entre las solicitudes de la arquitectura REST con otras arquitecturas como la SOAP.

Tabla III.1 Comparación de Comandos de Solicitudes de APIs.

Otros	REST
crearCarritodeCompras()	POST https://host/CarritodeCompras
borrarCarritodeCompras()	DELETE https://host/CarritodeCompras

REST impone las siguientes restricciones:

1. Usa las capacidades del protocolo HTTP tanto como sea posible.
2. Diseña los recursos no los métodos u operaciones.
3. Usa la interfaz uniforme, definido por los métodos HTTP, que tiene una semántica bien especificada.
4. Comunicación sin saber el estado entre cliente y servidor.
5. Uso del acoplamiento débil e independencia de las solicitudes web.
6. Usar códigos de retorno HTTP.
7. Usar tipos de medios

La arquitectura REST no es un estándar ni un protocolo. REST es un estilo de arquitectura con restricciones y acuerdos los cuales están basados en HTTP, que es un protocolo de nivel de aplicación de la capa 7 según el modelo OSI (Interconexión de Sistemas Abiertos) (Ferreira et al., 2013).

HATEOAS es una abreviación de hipermedia como motor del estado de la aplicación (*Hypermedia As The Engine Of Application State).* HATEOAS es una extensión de REST y cualquier restricción y ventaja de REST también aplica para HATEOAS. Algo adicional que aporta son las arquitecturas dinámicas, que es una arquitectura capaz de descubrir dinámicamente las acciones disponibles y acceder a los recursos que se necesiten, esto permite que el cliente pueda explorar la API sin tener el conocimiento del formato de los datos o de la misma API (Biehl, 2016). Las restricciones que impone son:

- Todas las restricciones de REST aplican.
- Los recursos deben estar ligados unos con otros. Las respuestas de las representaciones del API contienen hipervínculos dirigiendo a otro recurso.
- La semántica de la respuesta del API es prevista por los tipos de medios.

IV. DESARROLLO DEL PROYECTO

En este capítulo se describen las primeras etapas del trabajo de tesis donde se investigaron y evaluaron los distintos equipos de cómputo y plataformas de servicios de la Nube que fueron utilizados durante el desarrollo del proyecto para cumplir con los requerimientos del mismo (§1.4). El desarrollo de la tesis se implementó siguiendo la metodología descrita inicialmente (§1.5). Primero se seleccionaron los equipos a utilizar para desarrollar el sistema de aula inteligente, luego se realizaron algunas pruebas para obtener el mejor modelo CNN utilizando un *framework* para incorporar dichos modelos de detección de personas en los dispositivos de prueba para proceder a evaluar el rendimiento máximo de cuadros de video por segundo procesados mediante el mecanismo de inferencia de modelos pre-entrenados CNN adaptados para nodos FEC por (Flores,2018). Otro problema por resolver consistió en mantener la compatibilidad de dicha configuración para los distintos sistemas operativos de los nodos FEC habilitados en los distintos equipos seleccionados. Después se implementó el algoritmo de control del sistema de iluminación del aula inteligente, para finalmente desarrollar un API capaz de englobar toda funcionalidad y características mencionadas anteriormente para realizar las pruebas para las distintas configuraciones de nodos FEC contemplados para el aula inteligente, los cuales se hablará en capítulos posteriores.

4.1 Selección de equipo de cómputo para los nodos FEC.

Como se mencionó anteriormente se pretende realizar un estudio comparativo del rendimiento en el proceso de inferencia de modelos CNN en las distintas capas de una arquitectura osmótica (CC, FC y EC). Tanto los servicios de la Nube como los nodos FC y EC deben ser capaces de soportar el mismo sistema de control para el aula inteligente. Los nodos EC se caracterizan por ser equipos con recursos de cómputo limitados y los nodos FC pueden basarse en equipos convencionales. Para los nodos EC se eligieron los equipos RockPro64 y una Raspberry Pi 4 modelo B (RPi) y para los nodos FC se contó con un equipo DeepLens AWS y una computadora Intel NUC, mismos que son descritos más delante de forma detallada. Cada uno fue seleccionado por sus características y funciones que ayudaría a desarrollar el proyecto del aula inteligente, como se vio en el capítulo §1.4.5. Los aceleradores neuronales (*Neural*

Procesing Units, NPUs) son procesadores dedicados a correr modelos CNN, de los cuales destacan la Intel Movidius NCS (Neural Compute Stick) y Google Coral Stick (Google, 2018a; Intel Corp., 2018a). Sin embargo, la unidad Coral aún estaba en desarrollo y no se obtuvo el dispositivo para realizar las pruebas, por lo que se optó por la unidad Intel Movidius. Para realizar las pruebas en la capa de la arquitectura de la Nube y compararlas con las realizadas en los otros equipos se necesitó habilitar un servicio en la Nube capaz de correr el mismo programa que en los demás nodos FEC, ya que se busca la interoperabilidad y compatibilidad para ejecutar los mismos procesos de operación para el sistema del aula inteligente y posteriormente, realizar las pruebas comparativas de cada una de las capas de la arquitectura IoT osmótica. Para controlar los dispositivos IoT del aula se utilizaron aquellos que tuvieran la capacidad de conectarse a una red inalámbrica y comunicarse entre dispositivos a través de su correspondiente API. Los dispositivos IoT que controlamos con nuestro sistema de aula inteligente son un termostato para controlar la temperatura del sistema de aire acondicionado (AC), un proyector multimedia convencional y cuatro luminarias LED.

4.1.1 Movidius NCS y OpenVINO

La unidad Intel Movidius NCS, es un dispositivo tipo memoria USB que cuenta con un kit de desarrollo para efectuar de manera dedicada y eficiente la etapa de inferencia de modelos CNN con un consumo reducido de energía (aprox. 1 watt vs 30 wats de Nvidia TX2), es una NPU externa especialmente dirigida a desarrolladores, departamentos de investigación y desarrollo empresarial e investigadores de universidades que trabajan en aplicaciones de Aprendizaje Automático (ML) y Ciencia de Datos. El dispositivo de computación neuronal lleva incorporada la unidad de procesamiento de visión (VPU) Movidius, que ofrece una alta eficiencia energética y es capaz de ejecutar CNNs. Existen 2 versiones (figura 4.1), la primera versión de la NCS tiene una VPU llamada Myriad 2 y la segunda versión (NCS2) tiene una VPU más actualizada llamada Myriad X. La diferencia entre ambas versiones es que el modelo más reciente NCS2 ofrece una arquitectura con un rendimiento ocho veces mayor a la versión anterior (NCS), para la ejecución de modelos CNN para detección de objetos pueden llegar a tener un rendimiento similar a la de un equipo promedio con un CPU multi-core o una costosa tarjeta de video (GPUs) pero con un menor consumo de energía.

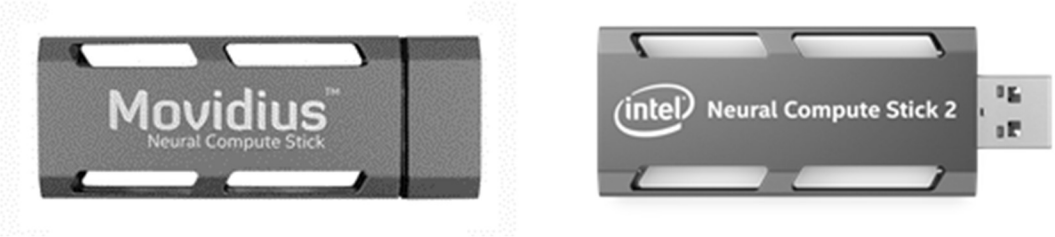

Figura IV.1 Aceleradores Neuronales Intel Movidius NCS (Myriad 2) y NCS2 (Myriad X).

Para utilizar estos aceleradores existe el kit de desarrollo de software de Intel (Software Development Kit, SDK) denominado NCSDK (Neural Compute SDK) que incluye herramientas de software, una API y ejemplos, para que los desarrolladores puedan crear software que aproveche la capacidad del *hardware* que ofrece la unidad Intel Movidius NCS (Intel Corp., 2019c). También existen dos versiones de este SDK, sin embargo, ninguno soporta la unidad NCS2 más reciente, por lo que se usó el kit de desarrollo de Intel llamado OpenVINO (*Open Visual Inference and Neural Network Optimization).* Esta herramienta de Intel es muy útil para correr modelos de redes neuronales convolucionales (CNN) para aplicaciones de visión por computadora y soporta distintos *frameworks* DL como Caffe, Tensorflow, MXNet, ONNX y Kaldi. OpenVINO soporta distintos tipos de hardware como CPUs, GPUs, FPGAs y NPUs. Las CPUs compatibles con OpenVINO abarcan la arquitectura Intel x64 multi-core desde la sexta hasta la octava generación, la familia Xeon (v6 y v5) y el procesador Pentium N4200/5, N3350/5, N3450/5 con Gráficos Intel HD (Intel Corp, 2018b). Para la unidad Movidius NCS, OpenVINO también soporta equipos PC limitados basados en ARM como la tarjeta Raspberry Pi, lo cual posibilita desarrollar, ajustar e implementar modelos de CNN en aplicaciones con bajo consumo de energía que requieren inferencia en tiempo real en equipos de bajo costo. Además, esta herramienta proporciona una biblioteca de funciones y modelos pre-entrenados. Según el autor Mircea H. Ionica del artículo "*An analysis of the suitability of the Movidius Myriad architecture for scientific computing*", la unidad Movidius NCS es muy eficiente energéticamente comparando los resultados de las pruebas hechas a diferentes dispositivos con multiplicaciones de matrices densas, como se muestra en la tabla 4.1.

Tabla IV.1 Rendimiento estimado de los dispositivos (Ionica & Gregg, 2015).

Dispositivos	GFLOPS	GFLOPS/W
Core i7 960	96	1.2
Nvidia GTX 280	410	2.6
Cell	200	5.0
Nvidia GTX 480	940	5.4
Stratix IV FPGA	200	7.0
TI C66x DSP	74	7.4
Xeon Phi7120D	2225	8.2
Tesla K40 (+CPU)	3800	10.0
Myriad I	**8.11**	**23.2**
Tegra K1	290	26.0

4.1.2 Equipos con recursos de cómputo limitados

Para los nodos FEC de la arquitectura de IoT osmótica se probaron los dispositivos RPi4 y RockPro 64. La tarjeta RockPro 64 es una minicomputadora de placa única, es el modelo más potente que ofrece su fabricante PINE64 (figura 4.2). La RockPro 64 cuenta con un SOC de núcleo hexadecimal Rockchip RK3399, así como un Mali-T860MP4 de cuatro núcleos y 4 GB de memoria de sistema LPDDR4 de doble canal. Además, la placa cuenta con un puerto USB 3.0 y un puerto USB tipo C. También cuenta con pines de comunicación para I2C, SPI, UART y GPIO. La placa puede interconectarse con diversos periféricos PINE64, como el módulo Wifi / BT, el módulo de cámara, el panel LCD, entre otros (PINE64, 2018).

Se implementaron algunas pruebas con esta minicomputadora, sin embargo, solo soporta la primera versión del NCSDK de Movidius, por lo que fue descartada para futuras pruebas.

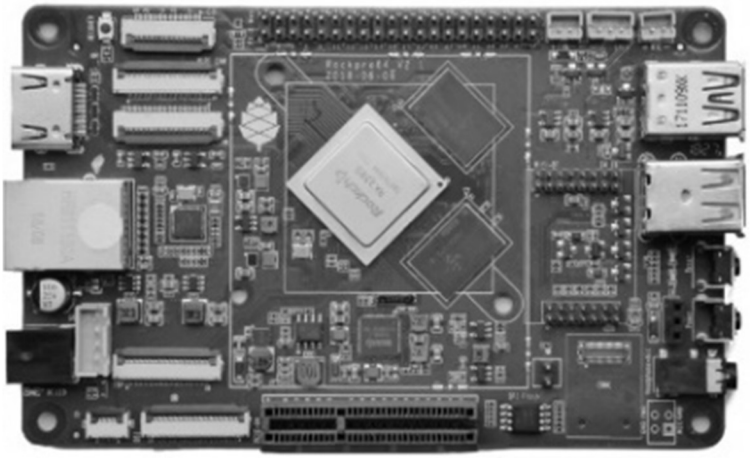

Figura IV.2 Tarjeta RockPro 64.

Por otra parte, la Raspberry Pi 4 (RPi) (figura 4.3) es una minicomputadora con un microprocesador ARM Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) de 64-bits tipo SoC @ 1.5GHz, 4GB de RAM LPDDR4, 2 puertos USB 3.0, un interconector GPIO estándar de 40 pines, Wifi 2.4 GHz y 5.0 GHz (IEEE 802.11ac), Bluetooth 5.0 y un puerto Gigabit Ethernet. Se instaló el sistema operativo Raspbian que es compatible con el *framework* de OpenVINO y se realizaron todas las pruebas posteriores en este equipo configurado tanto como nodo FC Y EC dentro de la arquitectura IoT del sistema.

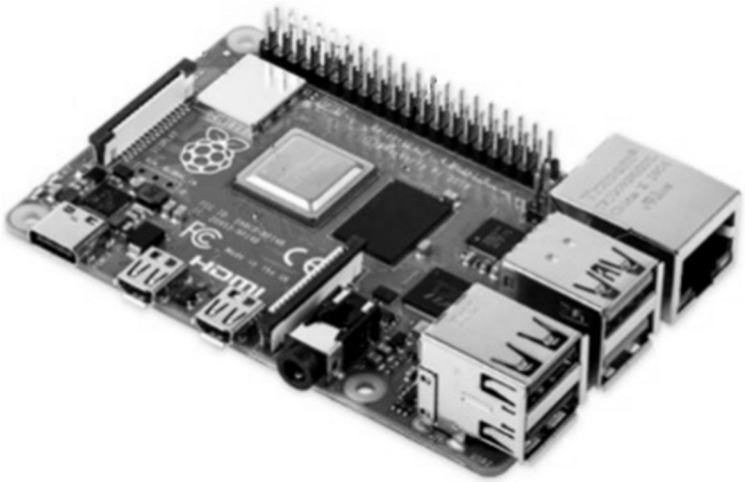

Figura IV.3 Tarjeta Raspberry Pi 4 modelo B.

4.1.3 Nodos FC en Computadoras Personales (PCs)

Para los nodos FC se evaluaron los equipos AWS DeepLens y la computadora PC Intel NUC. La minicomputadora AWS DeepLens 2019 de Amazon (figura 4.4) tiene una cámara de video integrada de 4 Megapixeles (1080p, MJPG) y el equipo está diseñado para integrarse a la plataforma de AWS Cloud de Amazon. El equipo cuenta con un microprocesador Intel Atom, una memoria de 8 GB, 16 GB de almacenamiento. El dispositivo soporta diversas herramientas proporcionadas por AWS para desarrollar aplicaciones de visión por computadora basados en modelos de Aprendizaje Profundo, por ejemplo, para detectar rostros.

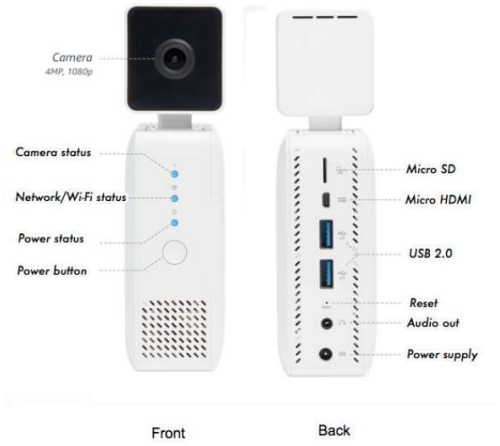

Figura IV.4 AWS DeepLens.

En la estación Amazon AWS DeepLens se logró implementar el mismo modelo CNN de las demás estaciones (RPi e Intel NUC), pero se presentaron dos dificultades importantes que impidieron poder aplicar este equipo al proyecto. El mayor obstáculo fue que la estación DeepLens no detectó la Movidius Compute Stick, debido a las reglas de seguridad del sistema operativo, una versión modificada de Ubuntu 16.04. Se realizaron varios intentos, pero no hubo resultados exitosos. El segundo obstáculo fue la librería que controla la cámara de la DeepLens que es para Python 2 y no existe una versión para Python 3, esta última es la requerida para ejecutar el modelo CNN. Lo que se ha logrado realizar en la DeepLens son proyectos de reconocimiento de rostros y objetos que son proyectos de ejemplo que proporciona AWS (Amazon Web Services) para observar lo que se puede lograr con el dispositivo. Conseguir que estos problemas planteados se solucionen llevará más tiempo del planteado para el proyecto de tesis, por lo tanto e igual que la RockPro 64 no se llevarán a cabo más pruebas.

La computadora PC Intel NUC es un ordenador en miniatura de alto/medio rendimiento, el modelo utilizado fue la NUC7- i5BNHXF Mini-PC con un CPU Intel dual-core i5-7260U de 64-bits @ 2.2 GHz y 16 GB RAM (15W) en el cual se instaló el sistema operativo Ubuntu GNU/Linux 16.04 LTS (figura 4.5). La MiniPC NUC es compatible con el *framework* OpenVINO gracias a su arquitectura x86. En este equipo se realizaron las primeras pruebas para determinar cuál sería en modelo CNN más conviene detectar personas a partir de un flujo continuo de imágenes de video. Además, en este equipo se experimentó con los primeros programas de interconexión y control de dispositivos IoT.

Figura IV.5 Computadora Intel NUC-i5BNHXF.

4.1.4 Nodo CC para los Servicios en la Nube

Para los servicios en la Nube existen diferentes proveedores como Google, Amazon, IBM y Microsoft por mencionar algunos. Estos servicios permiten el desarrollo de proyectos de *back-end* para aplicaciones en dispositivos móviles o equipos conectados en una red. Entre los proveedores evaluados se encuentran las plataformas de *Amazon Web Services* (AWS) y *Google Cloud Platform* (GCP).

La plataforma en la nube AWS de Amazon proporciona varias herramientas para el almacenamiento, habilitar servidores virtuales con tecnología *Elastic Container Service* (ECS), ofrece servicios de desarrollo y despliegue de código, entre otros. La plataforma ofrece algunos servicios gratuitos con un límite de uso mensual o diario según el tipo de servicio (Amazon Web Services, 2018). Por otro lado, la plataforma de Google Cloud (GCP) ofrece varias herramientas para realizar computación en la Nube, almacenamiento de datos, análisis de datos, aprendizaje automático, entre otros servicios. Además, soporta herramientas como Kubernetes que sirve para administrar el uso de contenedores para crear microservicios en los servicios de la Nube (Google, 2018b). Para la implementación de los servicios en la Nube se optó por la plataforma GCP por las ventajas que ofrece de usar contenedores para crear microservicios, la extensa documentación y la oportunidad de usar todas las herramientas sin costo por un período de prueba de 12 meses.

4.2 Selección de modelo de CNN para la detección de personas

Para la selección de los modelos CNN de detección de objetos se implementó un *script* en Python 3 para realizar las pruebas utilizando el *framework* de OpenVINO (Intel Corp., 2019) y Tensorflow (Google Brain, 2019). Para dichos modelos CNN es importante recordar que existe el requerimiento de poder implementarlos y ejecutarlos en todos los equipos seleccionados para cada capa de la arquitectura osmótica de IoT. A partir de las imágenes capturadas por la videocámara se transmiten al nodo correspondiente que sea capaz de procesarlas usando el modelo CNN para lograr la detección de personas en cada imagen. Las primeras versiones de los *scripts* implementados en Python 3 para ejecutar los modelos CNN en OpenVINO se basaron en los repositorios GitHub (Hyodo, 2018; Hyodo, 2019). Para correr modelos CNN específicamente en la RPi se usó el *framework* Tensorflow debido a que OpenVINO no soporta arquitecturas de CPU tipo ARM, tomando como punto de partida el repositorio de GitHub (Simonelli, 2017) donde proporciona la manera de crear los modelos en el formato correspondiente de OpenVINO para los modelos Tiny YOLO v2 y Tiny YOLO v3. Con la finalidad de evaluar el rendimiento de las distintas alternativas de modelos CNN para detección de objetos se implementaron y probaron los modelos YOLO v3, tiny YOLO v2, tiny YOLO v3, SDD+Mobilenet y SSDLite + Mobilenet v2 obteniendo las mediciones del tiempo promedio de inferencia en segundos para 200 cuadros de video, estas mediciones se muestran en la tabla 4.2.

Tabla IV.2 Rendimiento de modelos CNN para detección de objetos.

		Tiempo promedio de inferencia (*fps*)				
Equipos	Procesador	YOLO v3	Tiny YOLO v2	Tiny YOLO v3	SDD + MobileNet	SDDLite + MobileNet v2
RPi	NCS	0.515	2.5	3.99	9.55	6
	NCS2	3.97	3.2	3.97	11.88	6.5
	2 NCS	-	5.8	-	13.43	-
	CPU*	-	0.5*	0.5*	-	-
NUC	NCS	0.67	4.7	6.48	11.33	9
	NCS2	2.67	10.2	29.88	29.95	10.4
	2 NCS	1.29	12.4	12.97	18.83	20.2
	CPU	1.8	5.5	14.66	-	54

*Con Tensorflow 1.0

Para los modelos CNN implementados se eligió el formato de OpenVINO que almacena en un archivo binario la arquitectura del modelo CNN (el número y tipo de capas que contiene la CNN) y en otro archivo almacena los valores de los pesos de cada nodo del modelo CNN, cuyas extensiones son .xml y .bin respectivamente. En el caso de la RPi, se produjo una caída de voltaje en los puertos USB, mismo que impidió realizar las pruebas basadas en dos aceleradores Movidius. El formato de modelos CNN de Tensorflow en algunos casos es incompatible con el formato de OpenVINO al momento de requerir la ejecución de modelos CNN dentro del CPU de la RPi (solo soportó las versiones Tiny YOLO v2 y Tiny YOLO v3). En cuanto la NUC solo soportó el modelo CNN SSD+MobileNet con punto flotante de 16 bits que solo corre en las dos versiones de NCS y para correr modelos CNN en el CPU es necesario que el modelo soporte datos de punto flotante de 32 bits.

.

Además, la arquitectura de los modelos YOLO incorporan otros procesos después de haber obtenido los resultados de la inferencia de la red neuronal, para posteriormente obtener las coordenadas de los objetos detectados en la imagen, a diferencia del modelo MobileNet que reporta los resultados de forma directa. Para la implementación del sistema final del aula inteligente se eligió por estos motivos trabajar con el modelo CNN SSDLite + MobileNet v2 debido a que además de presentar resultados de rendimiento muy satisfactorio cuenta con los tipos de punto flotante apropiados para realizar procesos de inferencia tanto a nivel de CPU como en aceleradores externos tipo NPU.

.

4.3 Control de dispositivos IoT del Aula

Los dispositivos que se pretenden controlar en el aula son la iluminación, el termostato de la climatización del aula y un proyector multimedia convencional. El sistema de iluminación está compuesto por cuatro luces LED colocadas en una fila (zona de interés) mismas que pretenden encender según sea el área en que se detecte a una persona mediante el sistema de visión (videocámara y modelos CNN). El termostato únicamente encenderá el AC en el caso de que en el área se encuentre alguna persona y de no ser así se apagará. Y el proyector se encenderá cuando detecte a una persona acercándose a la zona de proyección y se apagará cuando la

persona abandone dicha zona. El sistema de aula inteligente, como se observa en el diagrama de estados de la figura 4.6, consta de una videocámara fija que captura imágenes de la zona de interés del aula, dichas imágenes son alimentadas y procesadas por el modelo CNN para la detección de personas dentro del aula. Los resultados de cada inferencia del modelo CNN se analizan para obtener el número de personas detectadas y en qué ubicación se encuentran dentro de las zonas de interés, para con esta información el sistema de control envíe los comandos a los dispositivos IoT para realizar la acción correspondiente y este proceso se repetirá hasta que la cámara deje de capturar imágenes.

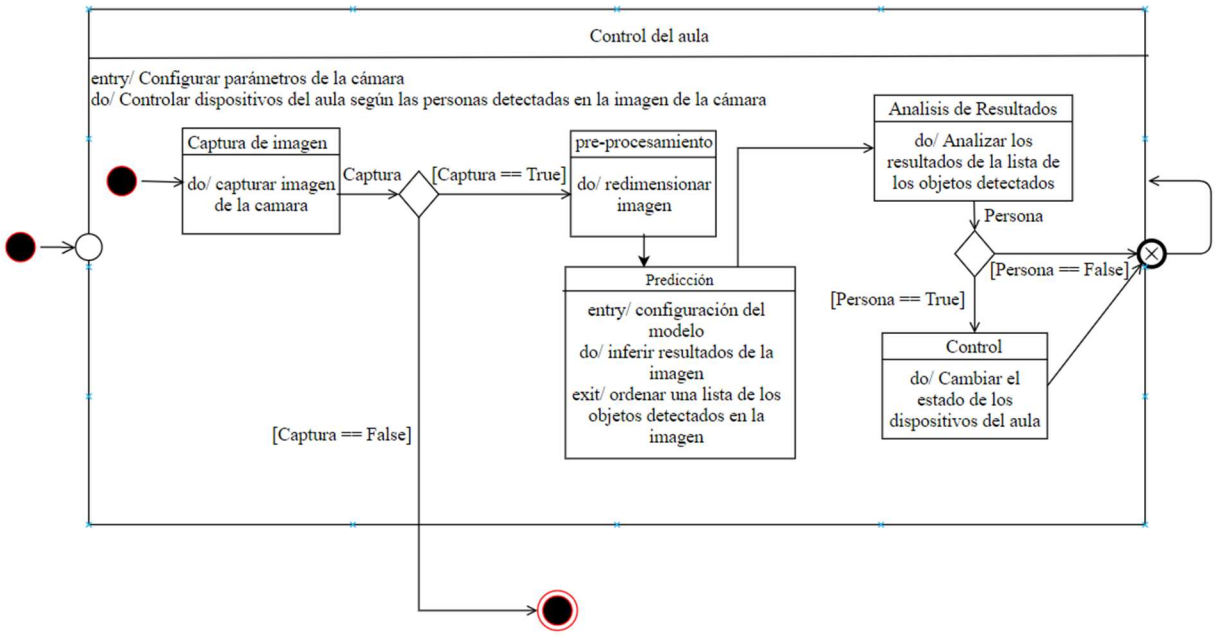

Figura IV.6 Diagrama de estados del sistema de visión y control del aula inteligente.

4.3.1 Iluminación Inteligente.

Para el sistema de iluminación inteligente se usaron cuatro focos LED modelo Philips Hue y un puente Wi-Fi Hue (*Smart bridge*) del mismo proveedor para intercomunicar por ZigBee los focos para controlar su estado (figura 4.7). Como se describió anteriormente (§1.4.2) los focos se colocan en una fila y según el área en el que se encuentre una persona estos encenderán. Para controlar los focos LED se usó una librería de Python llamada *beautifulhue* (Allanbunch,2013), pero al ir avanzando en la programación resultó insuficiente ya que no proporciona todas las funciones requeridas para controlar los focos. Indagando más a fondo

acerca de los dispositivos Philips Hue se encontró una API proporcionado por el proveedor y al tener la información de cómo se comunica el puente Hue con las luces, se creó un programa para controlar los focos a través de su API con arquitectura RESTful habilitada como un servicio web.

Figura IV.7 Paquete Philips Hue puente más 4 focos de colores.

El envío de los parámetros de las luces es en formato JSON y dependiendo de lo que se quiere hacer se usan los métodos POST, GET, PUT y DELETE para realizar los cambios en los estados de las luces a través de solicitudes HTTP. Los atributos de las luces se modifican al ingresar los valores que se desean cambiar y enviarlos en el formato JSON, algunos de estos atributos se muestran en la tabla 4.3.

Tabla IV.3 Atributos de las luces Philips Hue.

Llave	Valor	Descripción
on	True, False	Encendido o Apagado del foco
ct	[153...500]	Color de temperatura del foco desde un valor 153 (6500K) a 500 (2000K)
xy	[0… 0.9999, 0… 0.9999]	Las coordenadas "*x*" y "*y*" de un color dentro del espacio de color *CIE (Commision Internationale de L'Eclairage)*
bri	[1...254]	Intensidad o brillo de la luz con valores de 1 a 254
transitiontime	Numero entero positivo	Duración de la transición del estado actual del foco al nuevo en múltiplos de 100 ms, el valor determinado es 4 (400 ms)

Para la realización del programa que controla las luces fue necesario fijar algunas normas que deben seguir las luminarias. Estas deben encender exclusivamente en el área donde se detecten personas y tomar en cuenta las restricciones del puente Hue, según la documentación solamente puede atender 10 solicitudes por segundo para la modificación de los atributos de un foco y para un grupo de luces solo una solicitud por segundo. Tomando en cuenta lo anterior se definió una variable que contendrá el estado actual de las luces, representada con el símbolo H en la figura 4.8, y si los nuevos resultados de las inferencias detectan a una persona en una zona que no se encontraba en el estado anterior, entonces cambia el estado del foco. En otras palabras, únicamente cuando existe un cambio en el área que cubre la cámara (resultados obtenidos de la inferencia de los modelos CNN para las imágenes capturadas) se envían los comandos para cambiar el estado de los focos, como se puede observar en el diagrama de estados en la figura 4.8. Esto optimiza y evita el envío de solicitudes innecesarias para efectuar el cambio de estado de los focos, lo cual previene la posibilidad de saturar el puente Hue con un número excesivo de solicitudes.

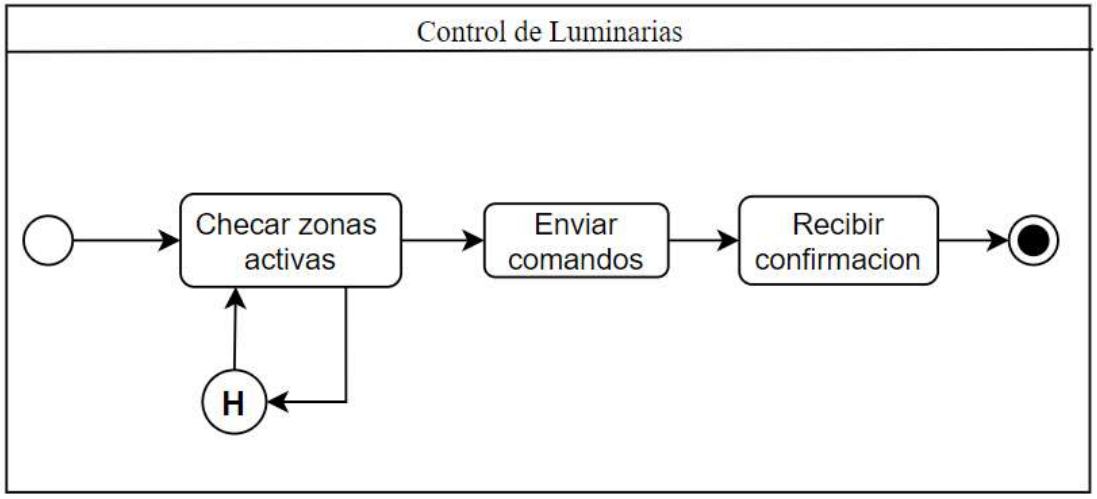

Figura IV.8 Diagrama de estado para el control de las luces.

4.3.2 Termostato Nest

Para el control del aire acondicionado se usó un termostato de la marca Nest (figura 4.8) ya que es famosa por conectarse a internet y poder controlarlo desde cualquier lugar. Todos los dispositivos Nest (termostatos, protecciones y cámaras) y sus aplicaciones se conectan al servicio en la nube que proporciona el fabricante Nest. Dichos servicios en la nube tienen, dentro de su protocolo de comunicación REST, un modelo de intercambio de datos en formato JSON para interactuar con los dispositivos Nest.

Figura IV.9 Termostato Nest.

El fabricante Nest proporciona soporte para desarrolladores de aplicaciones a través del portal "*Works with Nest"*. El termostato Nest cuenta con una API basada en arquitectura REST. Para poder conectarse a los servicios en la Nube de Nest primero se debe crear un cliente OAuth para definir la creación de un nuevo servicio por parte del desarrollador de aplicaciones que pueda ser autorizado para interactuar con un dispositivo Nest. Una vez realizada la solicitud, se genera el identificador de aplicación *client_Id* debidamente autorizado para acceder a los servidores Nest y posterior a la confirmación de dicho cliente se recibe un *token* que autoriza al desarrollador la posibilidad de acceder a una ruta en el modelo de datos del dispositivo para obtener acceso al mismo. El algoritmo desarrollado del aula inteligente activa el clima artificial a una temperatura preconfigurada del sistema, al detectar por un plazo de tiempo a un grupo de personas dentro de las zonas de interés monitoreadas por la videocámara. Del mismo modo, el algoritmo de control apaga el clima artificial por medio del termostato luego de no detectar la presencia de personas para un período de tiempo preestablecido.

Esta API es funcional para los servicios de Nest, pero con la reciente adquisición de Nest por parte de Google en agosto de 2019, los servicios de Nest se darán de baja y las aplicaciones que ya se contaban seguirán funcionando con la excepción de que no se podrán registrar nuevos usuarios a estas aplicaciones. Google ofrecerá acceso a través de su asistente virtual (Google Assistant) y proporcionará a los desarrolladores un nuevo API que estará disponible hasta finales del 2020 (*Nest Developers*, 2019).

4.3.3 Proyector Multimedia

Para controlar el proyector multimedia se desarrolló un módulo basado en una tarjeta ESP8266 como un *brigde* WiFi básico que recibe comandos por WiFi por parte de algún nodo FEC y la tarjeta emula un control remoto infrarrojo capaz de encender o apagar el proyector multimedia por medio de un led infrarrojo conectado a la tarjeta. Este sistema fue desarrollado por estudiantes de servicio social del Laboratorio de Aprendizaje Móvil y Sistema Embebidos Inteligentes y el desarrollo del trabajo está en un repositorio Github (Thenas,2018). A través de la codificación NEC (*National Electrical Code*) se envía el comando de encender o apagar el proyector por luz infrarroja. Este dispositivo se encarga de decodificar y simular un control remoto convencional. Tanto la decodificación se basa en la lectura o escritura de pulsos creados por un led infrarrojo con un tiempo específico de duración, estas pulsaciones contienen la información necesaria para poder controlar el dispositivo deseado por infrarrojo, como si fuera un control remoto convencional siguiendo el esquema de codificación NEC (tabla 4.4) para que el microcontrolador ESP8266 transmita las pulsaciones del led infrarrojo correspondientes al comando solicitado por un nodo FEC.

Tabla IV.4 Código de la codificación NEC para el proyector

Comando	Código	Comando	Código
Apagar	0xE172E817	Flecha derecha	0xE1724887
Menú	0xE17250AF	Flecha abajo	0xE17228D7
Flecha arriba	0xE172C837	Enter	0xE1724CB3
Flecha izquierda	0xE1728877	PC	0xE1729867

Para la comunicación por WiFi se utilizó una comunicación basado en el protocolo MQTT que es una arquitectura *Publisher/Subscriber* para manejar filas de eventos asíncronos, de esta forma al suscribirse a un tema (*topic*) se pueden recibir mensajes desde cualquier dispositivo suscrito al mismo tema. El protocolo MQTT necesita de un servidor principal llamado *broker* y para inicializarlo se utiliza un programa llamado Mosquitto (*Eclipse Mosquitto*, 2018) que debe estar instalado en alguno de los nodos FEC. Una vez iniciado el servidor MQTT/Mosquittose configura la tarjeta ESP8266 para que se conecte a la red del laboratorio e introducir la IP del *broker* y el tema al que se va a subscribir. La comunicación por MQTT con la tarjeta ESP8266 del proyector se implementó con la asistencia de la librería de Python de paho-mqtt (Light, s/f). El envío de comandos se realiza por medio de cadenas de caracteres codificados de acuerdo con los códigos de la tabla 4.3. Se implementó el control para un proyector multimedia Infocus (figura 4.10) de acuerdo con las especificaciones establecidas y descritas anteriormente (§1.4.2), encendiéndolo al detectar una persona (normalmente el docente o la persona bajo el rol de presentador en el curso) entrando a la zona 4 y apagándose hasta que la persona abandona dicha zona.

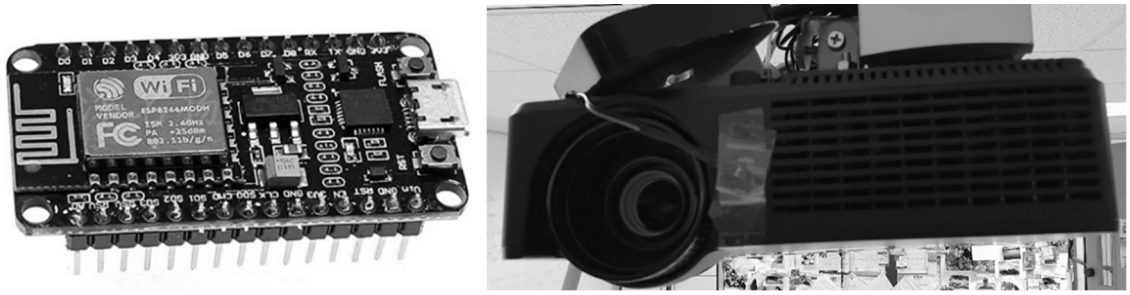

Figura IV.10 ESP8266 y proyector multimedia InFocus.

V. API PARA EL AULA INTELIGENTE

Para diseñar una API que englobe la funcionalidad del sistema del aula inteligente y sea escalable a futuro, se tomaron como punto de partida los requerimientos que buscamos para que el sistema sea lo más interoperable posible, que podemos clasificar en dos tipos diferentes de problemas. El primer problema es cómo comunicar un nodo FEC con otro para poder compartir información y el segundo es cómo abstraer las particularidades de las funciones de los distintos *frameworks* utilizados, tales como OpenVINO y Tensorflow, estableciendo bloques de construcción más genéricos y simples para que los futuros desarrolladores puedan entender y usar fácilmente dichos bloques. Primero se comenzó con la comunicación entre nodos que se encarga de compartir los datos (imágenes de video) para procesarlos posteriormente por medio de los modelos CNN para detección de personas. Una vez definidas las prestaciones para la comunicación se procedió a definir e implementar las funciones que manipulan los modelos CNN para cada uno de los *frameworks*. Al momento de diseñar un API se recomienda aclarar cuál es su objetivo, en nuestro caso es facilitar la tarea de implementación para nodo FEC en su respectiva capa tratando de abstraer su plataforma y arquitectura de hardware para ofrecer un mismo servicio capaz de recibir y procesar imágenes a través de modelos de CNN y transferir los resultados (personas detectadas) a los nodos FEC de control de los distintos dispositivos IoT que forman parte del aula inteligente, siguiendo un esquema cliente/servidor tal y como se observa en la figura 5.1. Gracias al API del aula inteligente es posible migrar buena parte del software de comunicación, procesamiento y control a equipos con diferentes arquitecturas y capacidades de procesamiento y hardware sin la necesidad de instalarlos, por lo que interconectar nuevos equipos siguiendo la analogía de *plug and play*. Al final del proceso, todos los programas resultantes y los archivos necesarios, que integran todos los recursos de un nodo FEC se virtualiza creando una sola imagen que puede moverse a cada uno de los equipos que serán habilitados como nodos FEC sin importar las particularidades de la arquitectura de hardware específica de cada equipo.

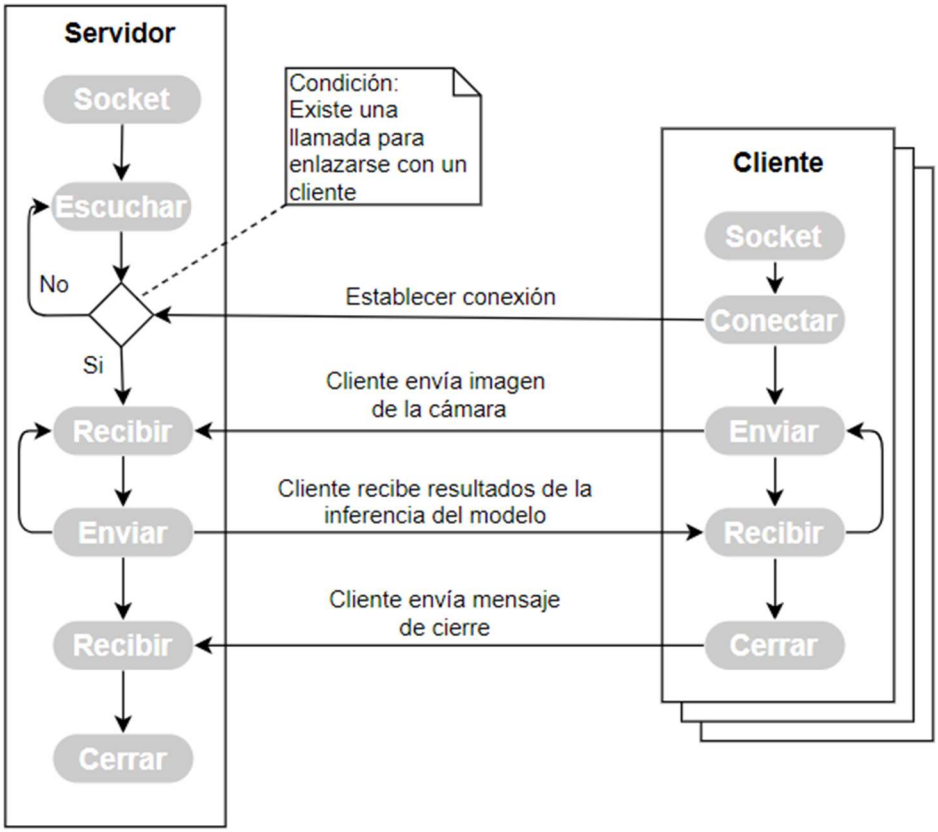

Figura V.1 Modelo de comunicación cliente-servidor.

5.1 El Protocolo de Comunicación

La comunicación entre los distintos dispositivos IoT es la parte principal del API desarrollada que de manera interna soporta tres tipos diferentes de protocolos de comunicación HTTP, MQTT y ZeroMQ. Para empezar a entender cómo funciona el intercambio de datos entre dos programas se debe comenzar por los sockets HTTP. El socket es una abstracción de bajo nivel para la capa de datos de la red que se encarga de conectarse con otro socket para intercambiar datos. Al igual que su nombre lo indica, socket se refiere a dos puntos u objetos que se pueden conectar de forma fiable, siendo en este caso un servidor y un cliente. Tanto HTTP, MQTT y ZeroMQ utilizan sockets para su funcionamiento, pero su desarrollo es de más alto nivel donde los usuarios no pueden modificarlo ya que deben cumplir ciertas normas para que sea interoperable y no pierda su compatibilidad con sus versiones anteriores. Se comenzó con HTTP que es un protocolo de transferencia usado para comunicarse con los recursos HTTP, como una página web, estos recursos usan una interfaz uniforme que se componen de los

métodos crear, leer, actualizar y borrar, para modificar los recursos de un servidor web (§2.2). Es muy utilizada en aplicaciones web y sobre todo aplicaciones IoT. Los problemas que surgieron con este protocolo fueron su latencia y la necesidad de convertir un arreglo de datos tipo *numpy* a una cadena de caracteres para poder enviar la imagen lo que consume aún más tiempo, teniendo los resultados de inferencia de una imagen en aproximadamente 6 segundos.

MQTT es un protocolo rápido, ligero, eficiente en consumo de energía e implementa varios niveles de calidad de servicio. En cuestión de latencia y consumo energía es más positivo que el protocolo http, sin embargo, posee algunos inconvenientes para el sistema que se quiere desarrollar el cual propone enviar una imagen como dato y el protocolo MQTT solo envía cadenas de caracteres. Por lo que al igual que HTTP se necesita convertir un arreglo de datos en una cadena de caracteres y volverlo a reconstruir al llegar a su destino que consume mucho tiempo. Otro problema es el de tener un servidor dedicado que se encargue de estos mensajes ya que en Python solo se puede ingresar como cliente. El tiempo promedio para realizar laa inferencia de una imagen es de 3.2 segundos. Pero este protocolo si es utilizado para comunicarse con la ESP8266 para encender o apagar el proyector del aula, ya que consiste en enviar una cadena de caracteres para que se pueda realizar.

ZeroMQ es otro protocolo de transferencia de datos cuya misión es obtener cero latencia o la menor posible. Usando una comunicación mucho más versátil y con funciones más fáciles de utilizar, fue la mejor opción en cuestión de enviar las imágenes a un servidor ya que cuenta con una función donde convierte un objeto Python a una serie de cadena de caracteres y lo vuelve a convertir en una imagen de forma eficiente. Los tiempos para obtener los resultados de la inferencia realizados en un servidor rondaban los 0.66 segundos, que es 5 veces más rápido que con el protocolo MQTT y 10 veces más que HTTP. La característica que ayudó es el par de conectores REQUEST-REPLY (solicitud-repuesta) que son las funciones que se encargan de enviar y recibir la información. El servidor enlaza un socket REP (respuesta) al puerto 5555, entra en un lazo infinito esperando una solicitud y responde cada vez con una respuesta. El cliente se enlaza con el socket REQ (solicitud) al mismo puerto que el servidor y luego se envía una solicitud y lee la respuesta del servidor. Este es el patrón de solicitud-respuesta y

probablemente la forma más sencilla de usar ZeroMQ ya que recurre al modelo clásico de cliente/servidor.

5.2 Framework de modelos CNN

Los *framework*s que se utilizaron para esta API son OpenVINO y Tensorflow con Keras. Se implementó Keras para el trabajo en conjunto de un compañero de laboratorio que consiste en probar modelos pre-entrenados y someterlos a técnicas de adaptación de modelos para hacerlos más ligeros y puedan correr en dispositivos con recursos de cómputo limitados. Además, tener varios *frameworks* posibilita aún más el uso de esta API. OpenVINO se utilizó para su uso en las NPU Movidius, con su ayuda se pueden correr modelos en dispositivos como la RPi. La API se enfocó en dos funciones principales en la configuración del *framework* para correr un modelo CNN y la predicción de este mismo a partir de los datos proporcionados.

5.3 Diseño del API

Una vez aclarado la comunicación y los *frameworks* de modelos de IA, empezamos con el diseño del API que se realizó a partir de 4 clases cada una con una función específica. Las clases contienen los métodos necesarios para comunicarse con los dispositivos del aula y con otros equipos para realizar la inferencia del modelo CNN de detección de objetos. La primera es la clase `fogClassroom`() que se encarga de inicializar el *framework* para correr modelos CNN los cuales son OpenVINO y Keras. Esta clase revisa el tipo de modelo CNN contenido en un archivo se le proporciona y dependiendo del tipo de extensión del archivo, h5 o xml, se inicializa el *framework* correspondiente. Para inicializar Keras para la RPi es diferente que en el sistema Ubuntu por lo que se revisa en qué procesador está corriendo el programa, si es un procesador ARM se busca importar la librería de Keras, de lo contrario busca importar la librería de Tensorflow, ya que en su nueva versión (Versión 2) este incorpora los comandos de Keras. OpenVINO tiene otros parámetros que se pueden modificar como el dispositivo en el que puede correr el modelo CNN ya sea en el CPU o en un dispositivo Myriad y la dirección al archivo con la librería para la compatibilidad de capas del modelo CNN. Keras únicamente necesita la dirección del modelo CNN para poder ejecutarlo con el CPU.

Una vez inicializado el *framework,* con el modelo CNN, en la instancia de la clase `fogClassroom`(), se puede utilizar el método `predict()` que se encarga de realizar la inferencia del modelo CNN con las entradas que se le suministre al modelo CNN y devuelve los resultados. Para ello es necesario que las dimensiones de la imagen a analizar sea del mismo tamaño que las dimensiones de entrada del modelo CNN y se creó una función que redimensiona la imagen para que se adecue a la dimensión de la entrada del modelo CNN. La instancia creada proporciona el tamaño de la entrada del modelo CNN por medio de la variable `in_size` y estas dimensiones se especifican en la función `convImgtoInputModel()` para redimensionar la imagen, pero hay que tener en cuenta que si el modelo CNN cuenta con más de una entrada esta función elegirá las dimensiones de la primera entrada del modelo CNN. Con la imagen ya redimensionada se realiza la operación de predicción del modelo CNN y la función devuelve el resultado para analizarlo y realizar las operaciones correspondientes.

Otra clase es la de `MessagePrtotocol`() que se encarga de enviar información a otro equipo o programa a través del protocolo de internet TCP. Los protocolos por utilizar son el de MQTT y ZeroMQ con lo que se puede iniciar una conexión y enviar o recibir algún mensaje. El protocolo ZeroMQ se usará para enviar las imágenes de la videocámara para analizarlas en el servidor y para regresar los resultados de la inferencia. Este protocolo viene con su librería en Python que envía y recibe cualquier objeto de Python y esto es posible por otra librería llamada pickle que convierte cualquier objeto Python en una secuencia de números binarios para enviarlos por TCP a la dirección IP especificada, como se observa en la figura 5.2, esta misma librería los recibe y puede revertir la secuencia de números binarios en el objeto Python original y restaurar la información original. El protocolo MQTT se encarga de enviar los mensajes para controlar un proyector y hacer que encienda o se apague según sea lo conveniente en el programa ya descrito anteriormente (§4.3.3).

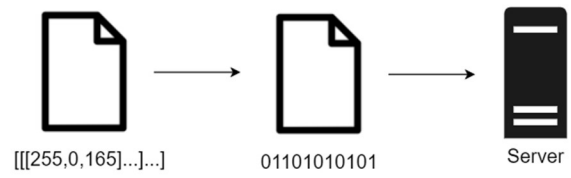

Figura V.2 Serialización de la imagen

La clase Hue se encarga del control de las luces de Phillips Hue, donde para poder comunicarse con el puente Hue se utiliza el protocolo HTTP, y así modificar los atributos de los focos. Al iniciar una instancia de la clase es necesario proporcionar la IP y el usuario del dispositivo para tener acceso al puente Hue que es el dispositivo que controla los focos, si el usuario no cuenta con dichos datos, el API se encarga de buscar la dirección IP del dispositivo dentro de la red y puede obtener el usuario dando clic en el dispositivo Hue que provoca que se genere un usuario nuevo. Una vez creada la instancia podemos usar el método de `changeState()` para cambiar el estado de las luces. También existen otros cuatro métodos, los cuales son: `get_status()`, `getAllLightsID()`, `getnewLight()` y `getIDfromName()`. El primero regresa el estado actual de los atributos de los focos seleccionados o de todos los focos conectados al puente Hue. El segundo método genera una lista de todos los identificadores de los focos conectados al puente Hue. El tercer método es para agregar un nuevo foco al puente Hue y el último método es para obtener el ID del foco a través del nombre otorgado al foco.

La última es la clase NestAPI que se encarga de conectarse a los servicios Nest para controlar y obtener el estado del termostato del aula. Al igual que la clase Hue utiliza el protocolo HTTP para conectarse a los servicios Nest. Al crear una instancia de esta clase se necesitan dos datos el *token* y el *id* del dispositivo a controlar que los proporciona los servicios Nest. Una vez creada la instancia podemos hacer uso del método `send()` para enviar los nuevos valores al termostato Nest.

5.4 Cliente, Servidor y Virtualización

Con la API diseñada podemos seguir con el desarrollo de los programas cliente/servidor y definir las funciones necesarias para crear un servidor capaz de atender las solicitudes de los clientes que envían imágenes para obtener la inferencia y obtener los objetos detectados en esa imagen. Se requiere un programa capaz de obtener las imágenes de una videocámara mientras se realiza la inferencia de la imagen, debe analizar los resultados obtenidos y ver si se detectaron personas para realizar las operaciones correspondientes en los dispositivos del aula. Para la interoperabilidad de estos programas se encontró la virtualización como una solución para poder correr los modelos CNN de detección de objetos en un dispositivo que no tiene las librerías o

programas necesarios para ejecutarlos, pero con la virtualización solo es necesario tener una plataforma para correr un contenedor que tiene todo lo necesario para correr los programas y así evitar la instalación de varios programas y librerías necesarias para ejecutar dichos modelos CNN.

5.4.1 Servidor

Al comienzo del desarrollo del servidor se implementaron varias formas de comunicación para enviar y recibir los datos necesarios, como se describió en la sección §5.1. Una vez encontrado el protocolo de comunicación adecuado se prosiguió con el desarrollo del servidor. El programa que se realizó para crear el servidor consiste en dos funciones: `detection()` y `main()`. En la función de detección se utiliza para redimensionar la imagen a la dimensión de entrada del modelo CNN y una vez con las dimensiones correctas realizar la inferencia de la imagen dentro del modelo CNN para que regrese los resultados. En la función `main()` se definen las variables a utilizar como lo son el lugar donde se encuentra el modelo CNN y la instancia de la clase `FogClassroom()` para iniciar el *framework* para el modelo CNN seleccionado. También la inicialización del protocolo de comunicación, en este caso ZeroMQ, se define en `main()` para que luego, dentro de un ciclo, el programa está a la espera de recibir un mensaje para empezar analizarlo con la función `detection()` y enviar los resultados una vez obtenidos a través de la misma función, al final el ciclo se repite una y otra vez.

5.4.2 Cliente

Al iniciar el desarrollo del programa se estableció que todo el proceso fuera secuencial, esto fue muy útil en las pruebas preliminares para enviar una imagen al servidor y obtener los resultados del modelo CNN de detección de objetos y así escoger el protocolo de comunicación entre cliente/servidor. Sin embargo, para obtener los resultados de analizar un video en tiempo real declarado en los requisitos del sistema (§1.4), se desarrolló un programa basado en paralelismo para que la espera de obtener los resultados del servidor no interfiera con la visualización de las imágenes en pantalla y observar el video en tiempo real.

El programa para el cliente consistió en crear dos funciones que corren en paralelo con dos colas o filas de datos que tienen en común para compartir las imágenes y los resultados de la inferencia. En la figura 5.3 se puede observar los estados que cada función debe realizar. La función `cam()` se encarga de obtener las imágenes de la videocámara mientras que la función `inference()` se encarga de conectarse al servidor, enviar las imágenes y recibir los resultados del servidor. En la función `cam()` también llama a otras funciones como `draw_box()` que se encarga de dibujar un recuadro en las personas detectadas en la imagen y excluyendo a los demás objetos. Además, con los puntos que se utilizan para dibujar el recuadro se obtiene en qué área de la imagen se encuentra la persona detectada y esa información se envía de vuelta a otra función llamada `dispIoT()` para cambiar los estados de los dispositivos IoT según sea el caso, como se describió en el tema de los dispositivos IoT (§4.3).

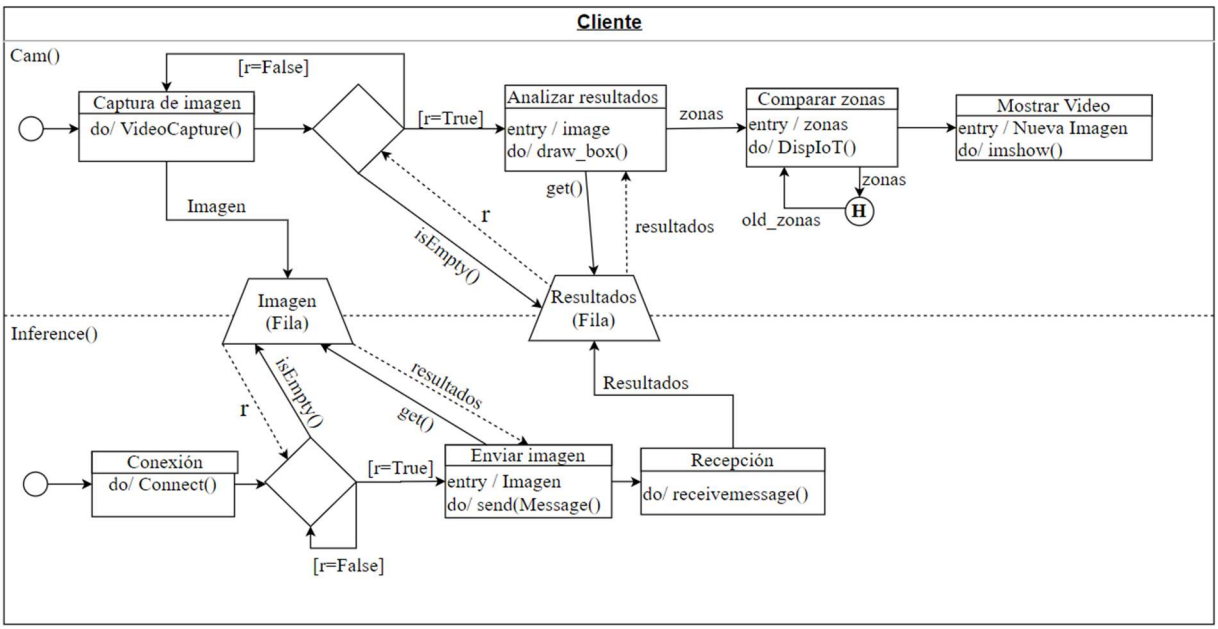

Figura V.3 Diagrama de estado del programa cliente.

Para entender más, cómo funciona el intercambio de datos entre los procesos, podemos observar la figura 5.4, donde los procesos de `cam()` e `inference()` revisan si la fila está vacía, de ser así se saltean sus respectivos procesos y vuelven al principio del proceso para volver a revisar la fila, si se detecta que no está vacía, se remueve el dato de la fila para hacer uso de ella. En el caso de la función que se encarga de la videocámara, esta remueve los datos que contienen los resultados de la inferencia del modelo CNN de detección de personas, proporcionados por la función `inference()`, y esta función remueve de la fila la imagen que proporciona la función

cam(), que utiliza para enviarla a un servidor y obtener los resultados de la inferencia. Al iniciar, revisa que la lista de las imágenes no esté llena, si se cumple esta condición, se remueve la primera imagen que está en la fila y luego la sustituye por una más reciente. Cuando la fila no está llena se van agregando las imágenes hasta que esta se llena, la cantidad depende del programa. En este caso se configuró una capacidad de hasta cinco datos en la fila. Como se puede percibir las funciones trabajan de manera asíncrona, el envío de la imagen y la obtención de los resultados del servidor no alcanzan la velocidad en la que la videocámara obtiene las imágenes. Una vez que el usuario decida romper el ciclo de la función que se encarga de obtener las imágenes la videocámara, también se detiene la función que se encarga de la inferencia y se termina el programa.

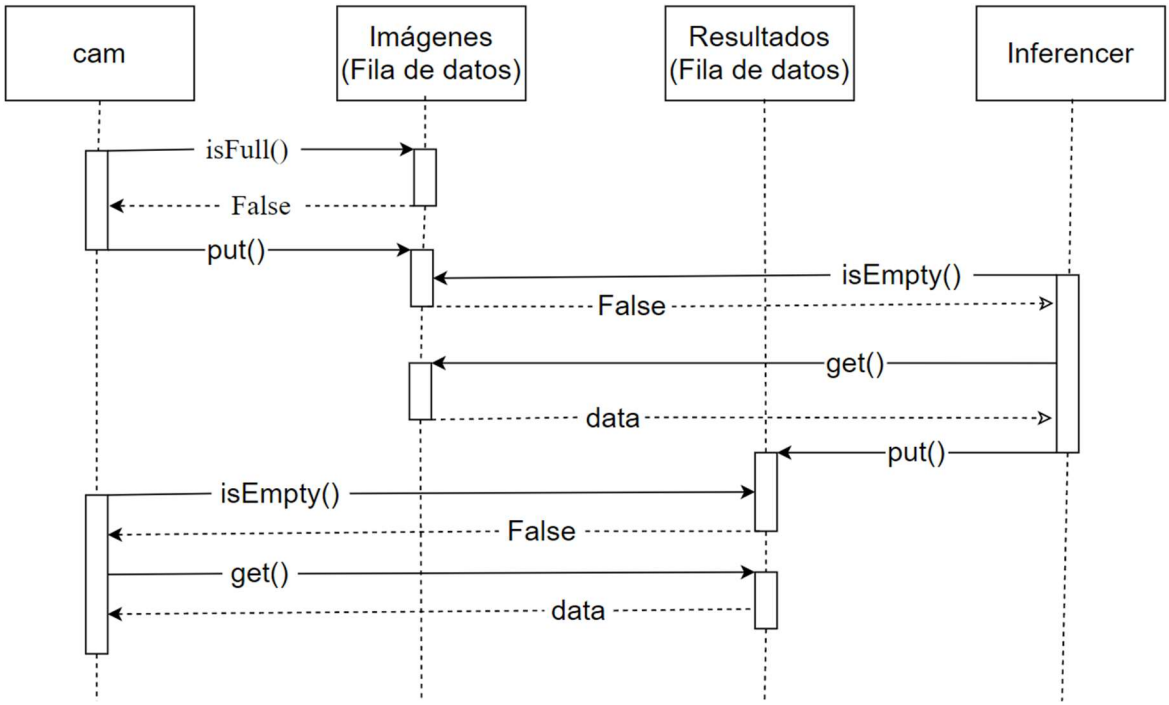

Figura V.4 Diagrama de compartición de datos entre procesos paralelos.

5.4.3 Virtualización

Para la parte de interoperabilidad se decidió una virtualización del sistema para poder transportar el código a cualquier equipo. Docker es una plataforma donde se pueden crear y correr pequeños programas virtualizados que están almacenados en una imagen Docker. al paquete de archivos de estos programas se le denomina imagen. Se decidió virtualizar la parte

del servidor porque la instalación de los *frameworks* de los modelos de IA es muy compleja y en ocasiones interfieren con otras aplicaciones que se tienen instaladas en el equipo. Existe un archivo denominado Dockerfile, es un archivo de texto que contiene los comandos de línea que el usuario puede usar para construir una imagen. El constructor de Docker se encarga de crear la imagen Docker almacenando los archivos que se instalaron con los comandos de línea en el Dockerfile.

La imagen Docker para la computadora Intel NUC fue creada a partir de una imagen base de Ubuntu 18.04 en el que se instala el *framework* de OpenVINO, sin embargo, primero se deben instalar y actualizar algunas librerías del sistema, que están agrupadas y separadas por una diagonal invertida, en la variable `DEPENDECIES`. Luego con la instrucción RUN se pueden correr programas desde la línea de comandos en una terminal que depende de la imagen base, en este caso una terminal del sistema Ubuntu, para instalar todas las dependencias que sean necesarias. Todo esto se realiza con la siguiente secuencia de comandos.

```
FROM ubuntu:18.04
ARG DEPENDENCIES="cpio\
                  build-essential \
                  python3-pip \
                  libgtk-3-0 \
                  libpython3.6-dev \
                  udev \
                  libusb-1.0"
RUN apt-get update && \
    apt-get install -y --no-install-recommends ${DEPENDENCIES} && \
    rm -rf /var/lib/apt/lists/*
```

Una vez instaladas y actualizadas las librerías se empieza con la instalación de OpenVINO y se realiza una copia de los archivos de instalación del *framework* en la imagen que se va a crear y luego se realiza su instalación con los siguiente dos comandos.

```
COPY /l_openvino_toolkit_p_[version] /l_openvino_toolkit_p_[version]
RUN cd l_openvino_toolkit* && \
    sed -i 's/decline/accept/g' silent.cfg && \
    ./install.sh -s silent.cfg && \
    rm -rf /tmp/*
```

Para evitar que la imagen se vuelva pesada se eliminaron los archivos de instalación y se procedió a instalar las librerías de Python necesarias para ejecutar el servidor, para ello se copió un archivo de texto llamado requirements.txt que contiene la lista de las librerías de Python necesarias para correr el programa, las cuales son Tensorflow 2.0, Pillow y pyzmq. Una vez agregada la lista a la imagen se procede a actualizar el paquete de instalación *pip* e instalar la librería *setuptools* que es un requisito para poder instalar Tensorflow y al final se instalan las librerías de la lista agregada recientemente. Todos estos pasos se realizan con los siguientes comandos.

```
ADD requirements.txt requirements.txt
RUN python3 -m pip install --upgrade pip
RUN python3 -m pip install setuptools
RUN python3 -m pip install -r requirements.txt
```

Con todos los programas y librerías instalados solo queda agregar la carpeta donde se guarda el programa y se deja la dirección de esa carpeta como inicio de la terminal, ya que dentro de la carpeta se encuentran los archivos que ejecutan el programa. Se realizaron esos cambios con los siguientes dos comandos.

```
COPY server /server
WORKDIR /server
```

Al final se agrega el comando que se encargará de iniciar el programa Python que se desarrolló, y para ello es necesario poner el comando de inicialización de OpenVINO y la dirección donde se encuentra el programa de Python, como se muestra en el siguiente comando.

```
CMD /bin/bash -c “source /opt/intel/openvino/bin/setupvars.sh \
                 && python3 server.py”
```

La imagen que se hizo con Ubuntu no se puede usar en cualquier dispositivo, como es el caso de la RPi, debido por la arquitectura del CPU en la que se hizo la imagen es amd64 y la RPi tiene un procesador ARM, por lo que no son compatibles. Se realizó otra imagen para la RPi donde se usa como base una imagen del sistema operativo Debian. Este sistema no es compatible con la versión 2.0.1 de Tensorflow, por lo tanto, usaremos la librería externa de Keras en el *api* y la librería Tensorflow con la versión 1.14.0. En la instalación de OpenVINO

solo se necesitó copiar los archivos de instalación a una locación del sistema para poder ser utilizados para ejecutar modelos de IA. Esta locación por defecto se almacena en una dirección en la raíz del sistema la cual es `opt/intel/openvino`. Una vez aclarado las incompatibilidades para la RPi se realizaron los mismos pasos para crear la imagen Docker para la RPi. Sin embargo, la RPi necesita de los aceleradores neuronales y para poder utilizarlos es necesario una configuración adicional, se requiere configurar las reglas de los puertos USB. Esto permite que al momento de conectar la NPU Movidius, el sistema pueda reconocer estos dispositivos. Los siguientes comandos se utilizan para para poder otorgar permiso al uso de las NPU Movidius.

```
Source /opt/intel/openvino/bin/setupvars.sh
sh /opt/intel/openvino/install_dependencies/install_NCS_udev_rules.sh
```

Se debe destacar que al momento de correr esta imagen, se necesita correr el parámetro `device` para usar las NPU Movidius, este comando sirve para compartir dispositivos externos con Docker, como se muestra en el siguiente comando.

```
docker run --device=/dev/bus/usb/001/XXX
```

Donde las XXX representan el número con el que se representa el dispositivo USB que se quiere conectar con la imagen Docker. Con las imágenes Docker listas podemos continuar con la implementación y pruebas del sistema creado para el aula inteligente en los 3 niveles de la arquitectura osmótica que se verán en los capítulos posteriores.

VI. IMPLEMENTACIÓN Y PRUEBAS

La implementación del sistema del aula inteligente se probó en las tres capas de la arquitectura IoT osmótica y con la ayuda del API fue más sencillo desarrollar un sólo programa para que ejecutar el modelo CNN para detección de personas y con los resultados obtenidos controlar los dispositivos IoT del aula según convenga. El programa que se ejecutó en cada capa de la arquitectura es el mismo, pero se crearon dos imágenes Docker diferentes una para sistemas AMD64 y la otra para sistemas ARM (§5.4.3). Ambas imágenes Docker generadas contiene los mismos programas y protocolos, detallados en la figura 6.1.

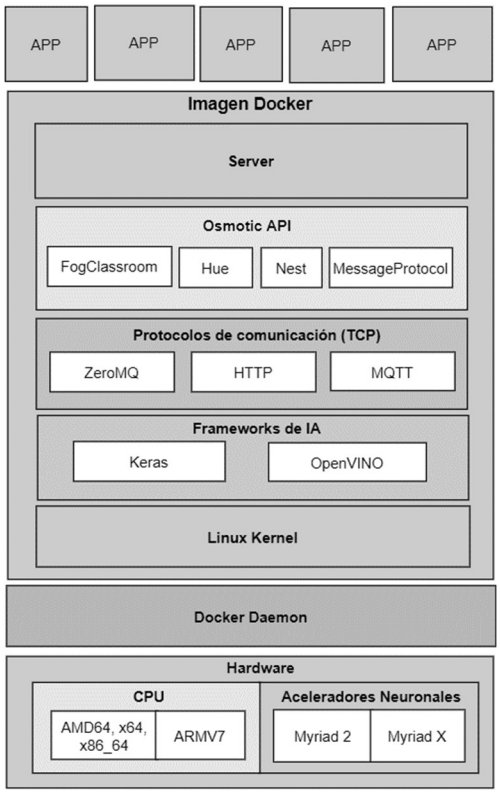

Figura VI.1 Composición de la imagen Docker

Para realizar las pruebas en CC y en FC se necesita establecer un servidor que se encargue de recibir las imágenes de video y procesarlas con el modelo CNN de detección de objetos para enviar las predicciones resultantes. Al programa que recibe estos resultados se le denominará cliente, este se encarga de enviar las imágenes de la videocámara al servidor. Con los resultados del servidor se controlarán los dispositivos IoT del aula.

6.1 Terminología para documentar las pruebas realizadas

Las pruebas en CC y FC consistieron en enviar imágenes de video a distintas resoluciones y con distintos tipos de conexión de red, simbolizadas como Wifi (W) y Ethernet (E). En el caso de FC, tanto el servidor como el cliente pueden conectarse por W o E, por lo tanto, se realizaron las pruebas para cada una de sus configuraciones W-W, W-E y E-E, donde el primero concierne al tipo de conexión del servidor y el segundo al cliente. Las resoluciones utilizadas fueron tres, estas se muestran en la tabla 6.1. Cada resolución de la imagen ocupa cierto espacio de memoria por lo que su tiempo de envío y de procesamiento difiere en relación con los tres tipos resolución usados: alta definición (HD), definición estándar (SD) y resolución del modelo (MR). Para las pruebas realizadas en la RPi se utilizaron los aceleradores neuronales para correr los modelos CNN de detección de objetos, los cuales se denominaron como NCS1 y NCS2, donde el primero es el que contiene la Myriad 2 y el segundo la Myriad X, que es una versión mejorada.

Tabla VI.1 Resoluciones de las imágenes capturadas

Nombre	Resolución	Tamaño
High Definition (HD)	1920x1080	5.9 MB
Standard Definition (SD)	640x480	900 KB
Model Resolution (MR)	300x300	263.6 KB

6.2 Cómputo en la Nube

La implementación en CC consistió en desarrollar un proyecto con los servicios que ofrece *Google Cloud Platform* (GCP). En el diagrama de la figura 6.2 se observa las herramientas de GCP que se utilizaron para crear el servicio en la Nube y ejecutar un servidor capaz de analizar imágenes con modelos CNN de detección de objetos y enviar los resultados al cliente. Primero se debe crear una máquina virtual y luego ejecutar la imagen Docker, pero primero debemos subir la imagen Docker a los repositorios del proyecto de Google que se había creado anteriormente. Con los comandos de Docker se cambió el nombre de la etiqueta ligada a la imagen creada con el siguiente patrón `gcr.io/[nombre del proyecto]/[nombre de la imagen]:[etiqueta]`

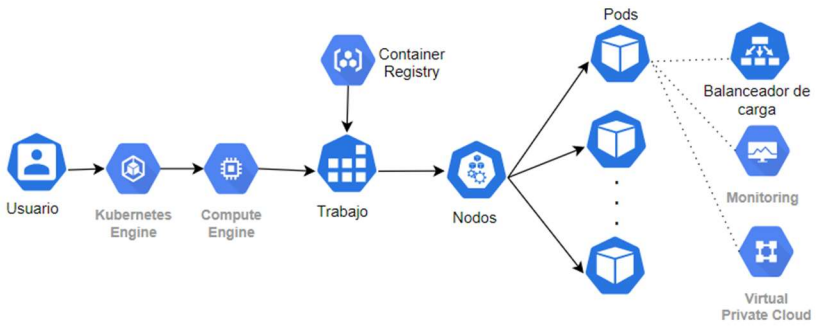

Figura VI.2 Diagrama de los servicios de GCP.

Una vez cambiado el nombre de la etiqueta a la imagen, usamos el comando `Push` de Docker para que empiece a subir la imagen a los servicios de GCP. Luego de haber subido la imagen podemos iniciar un servicio de Kubernetes para la ejecución del programa. Existen 2 formas de inicializar el servicio a través de comandos de consola o por el interfaz de la página web. Para configurar el proyecto a través de comandos de consola se necesita instalar la herramienta para la línea de comandos denominada `gcloud`, luego usamos los siguientes comandos.

```
1  gcloud auth login
2  gcloud config set project [Project_ID]
3  gcloud config set compute/zone [COMPUTE_ZONE]
4  gcloud config set compute/region [COMPUTE_REGION]
5  gcloud components update
6  gcloud auth configure-docker
7  docker push gcr.io/[Project_ID]/[nombre de la imagen]:[etiqueta]
8  gcloud container clusters create [CLUSTER_NAME] \
   --zone [COMPUTE_ZONE]
9  gcloud container clusters get-credentials [CLUSTER_NAME]
10 kubectl run [nombre del servicio] --image [ruta a la imagen] \
   --cluster [nombre del cluster]\
11 kubectl expose deployment [nombre del servicio] \
   --type LoadBalancer --port 80 –target-port 5555
```

Los comandos del 1 al 5 son las configuraciones iniciales de la herramienta `gcloud` para obtener acceso a los recursos de GCP y establecer algunos valores por defecto. La primera línea es la que se encarga de autentificar la cuenta de GCP y obtener acceso a ella. La segunda línea

permite elegir un proyecto de la plataforma y así crear y administrar los servicios que este puede proporcionar. Las siguientes dos líneas permiten seleccionar algunos datos por defecto como la zona y la región donde se prefiere que se creen los servicios. La selección de estos dependerá de la aplicación que se quiere utilizar y de qué tipo de clientes va dirigido, si se busca ofrecer el servicio en Europa o América del Norte se debe elegir la región más cercana a estos lugares y en algunas regiones existen más zonas con el fin de no saturar los servidores y reduciendo aún más la latencia. Todo esto también se podría configurar con el comando `gcloud init` que llama a un programa para configurar estos parámetros de la herramienta de `gcloud`.

El quinto comando es para actualizar la herramienta de comandos de línea de `gcloud` con el fin de tener las funciones más actualizadas y no tener complicaciones al momento de crear o administrar un servicio. Los siguientes dos comandos son para subir una imagen Docker al repositorio de GCP, primero configuramos Docker a fin de autenticar las solicitudes a *Container Registry* y poder subir la imagen, para ello es necesario que la imagen tenga el nombre de registro hacia dónde se quiere subir y se divide en tres partes el primero es el host que es `gcr.io`, luego está la identificación del proyecto y al final el nombre y versión de la imagen. El segundo comando es para enviar la imagen al repositorio de GCP.

El octavo comando crea un servicio o trabajo en el clúster y al no definir los atributos del comando `cluster create` se usan los parámetros predeterminados que se configuraron al inicio. Una vez creado el trabajo en el clúster se le puede agregar un nodo de trabajo con Kubernetes, pero primero se deben bajar las credenciales del clúster con el comando 9 para que los comandos de Kubernetes puedan tener acceso a la administración del clúster y después con el comando 10 creamos un nuevo trabajo para ejecutar el contenedor Docker donde está el servidor desarrollado en Python. En este paso se debe tomar cautela de que la imagen de Docker se ejecute correctamente en el nodo o los nodos del clúster que se están corriendo, ya que el sistema busca un archivo ejecutable que empiece a correr. Al final se exponen los servicios en una red pública con el comando 11, para que distintos clientes puedan conectarse y comunicarse con el servidor desde cualquier lugar con acceso a internet.

Para configurar el proyecto a través de la interfaz de la página web de GCP se realizan los mismos pasos anteriores, pero aquí son más sencillos ya que no es necesario instalar un programa en la computadora, solo se necesita entrar a la página de GCP e iniciar sesión con una cuenta de Google. En la figura 6.3 se observan las opciones a configurar del servicio en la Nube, donde la opción número 1 se utiliza para escoger qué proyecto se va a utilizar para realizar los servicios de la Nube, de no tener uno o sí se quiere empezar de cero se puede crear uno nuevo. Como se requiere un servicio para administrar contenedores, entonces se usa la herramienta de Kubernetes. Dentro de esta herramienta se utilizan principalmente tres opciones que son las opciones 3, 4 y 5 que se muestra en la figura 6.3.

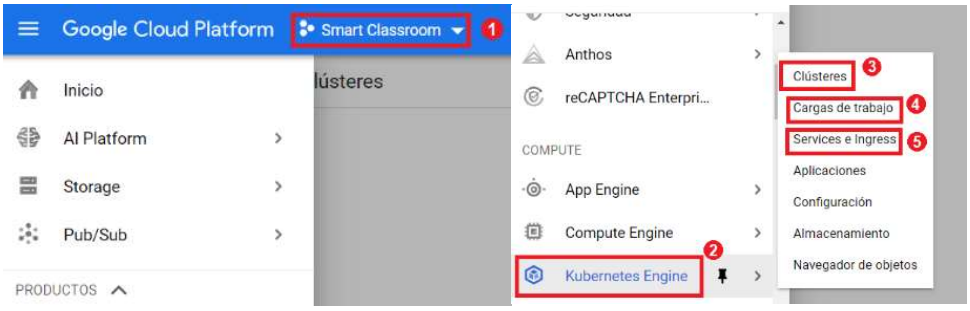

Figura VI.3 Interfaz web de GCP.

Para empezar a configurar un servicio de Kubernetes primero se debe crear un clúster, al entrar en las opciones se pueden observar las características que se pueden modificar como se muestra en la figura 6.4. En el recuadro marcado con el número 1 se debe poner un nombre al clúster, que por defecto tiene el nombre de cluster-1. En el segundo recuadro se selecciona la zona o región en el que se busca que se encuentre corriendo ese clúster, aquí la selección dependerá del área en que los usuarios se pueden conectar. El tercer recuadro son opciones para configurar la seguridad o poder conectar otros servicios con el clúster, por el momento no es necesario cambiar sus valores por defecto. Sin embargo, algo que se debe destacar es que la configuración del hardware del clúster se puede modificar en la opción de nodos, se pueden usar hasta 160 vCPU y 3.75 TB de memoria RAM. Para este proyecto, la opción estándar es más que suficiente con 1 vCPU y 3.75 GB de memoria. Una vez ajustados los valores de la configuración del clúster se puede crear al hacer clic en el botón que se muestra en el cuarto recuadro de la figura 6.4.

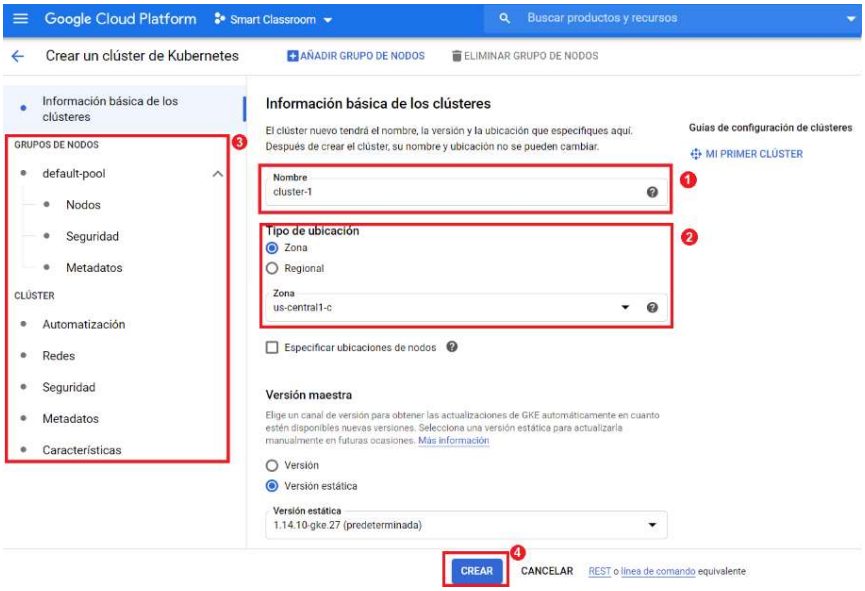

Figura VI.4 Interfaz web para crear un clúster en GCP:

El siguiente paso es desplegar un trabajo y, como se observa en figura 6.5, se escoge la imagen Docker que previamente se subió al *Container Registry* con los mismos comandos que se mostraron anteriormente. Es posible utilizar más de un contenedor y poder comunicarse entre ellos, esto suele ser conveniente para programas grandes y compartir el proceso de cómputo o particionar el programa por módulos, esto hace más fácil el mantenimiento del programa y en la mayoría de los casos ocupa menos almacenamiento en comparación a tener todo el programa en un solo contenedor.

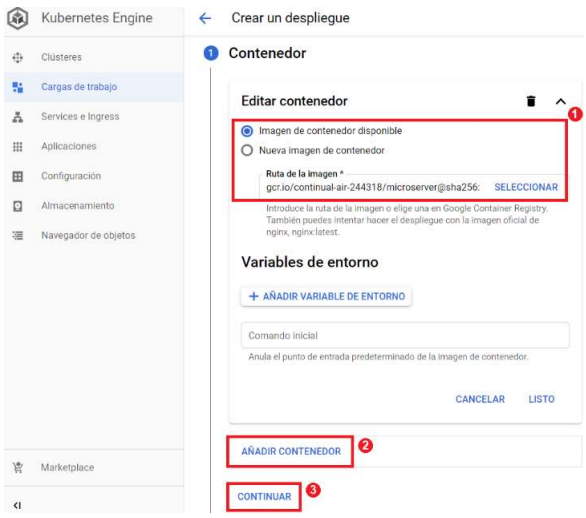

Figura VI.5 Primer paso para desplegar un trabajo en Kubernetes de GCP.

La segunda parte para desplegar un trabajo consiste en darle un nombre a la aplicación para poder diferenciar de otras aplicaciones o servicios activados en el proyecto de GCP que se llena en el primer recuadro de la figura 6.6. El segundo recuadro es para configurar las reglas con las que el contenedor podrá interactuar con otros contenedores y con la plataforma de Kubernetes, esto lo dejamos con sus valores por defecto. En el tercer recuadro seleccionamos el clúster que se creó anteriormente y, por último, se despliega el trabajo haciendo clic en el botón que se muestra en el cuarto recuadro.

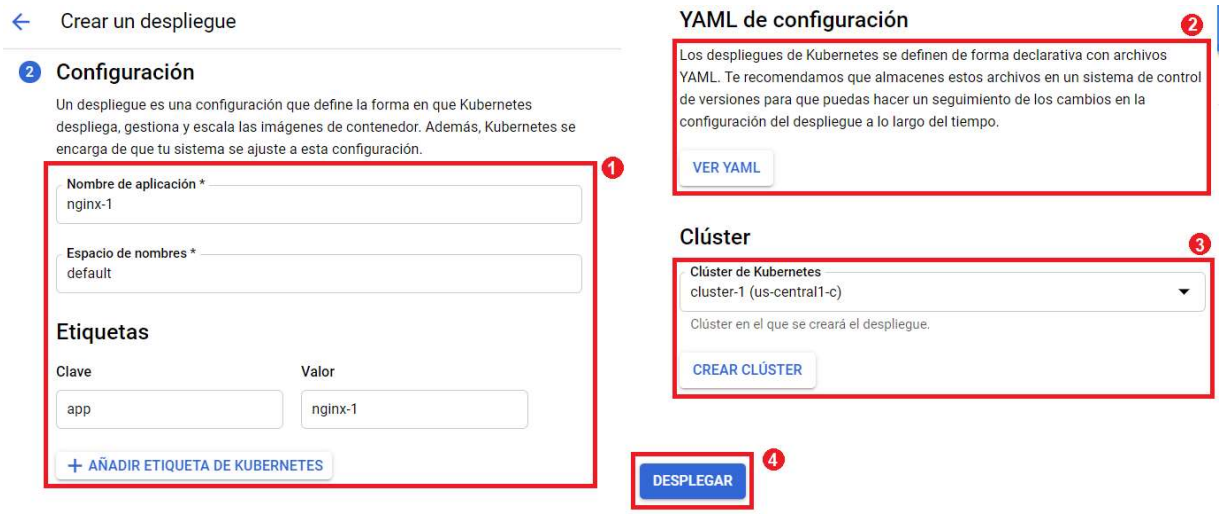

Figura VI.6 Segundo paso para desplegar un trabajo en Kubernetes.

Una vez desplegado el trabajo y no tener errores al correr el contenedor Docker se puede exponer en la red pública. Para exponer el trabajo desplegado es necesario ir a las opciones y características del trabajo y buscar la opción de Mostrando servicios y dar clic en el botón de exponer como se muestra en el recuadro rojo de la figura 6.7a. Una vez expuesto el trabajo se obtiene la dirección IP pública como se muestra en la figura 6.7b. Con esta dirección se podrá comunicar el cliente y al contar con un balanceador de carga podrá distribuir el trabajo en las distintas réplicas del servidor.

a) b)

Figura VI.7 Exposición del trabajo desplegado en Kubernetes

Al momento de exponer el trabajo es necesario indicar que puertos se van a abrir para la comunicación entre el servidor y cliente, como se observa en la figura 6.8. El primer puerto que se abre es para la IP pública y el puerto de destino es el puerto que la imagen Docker tiene expuesto. Normalmente, para la comunicación con servidores web se usa el puerto 80, pero si la imagen Docker expone uno diferente es necesario especificarlo para evitar problemas de comunicación entre cliente y servidor. El servidor que se desarrolló utiliza el puerto 5555.

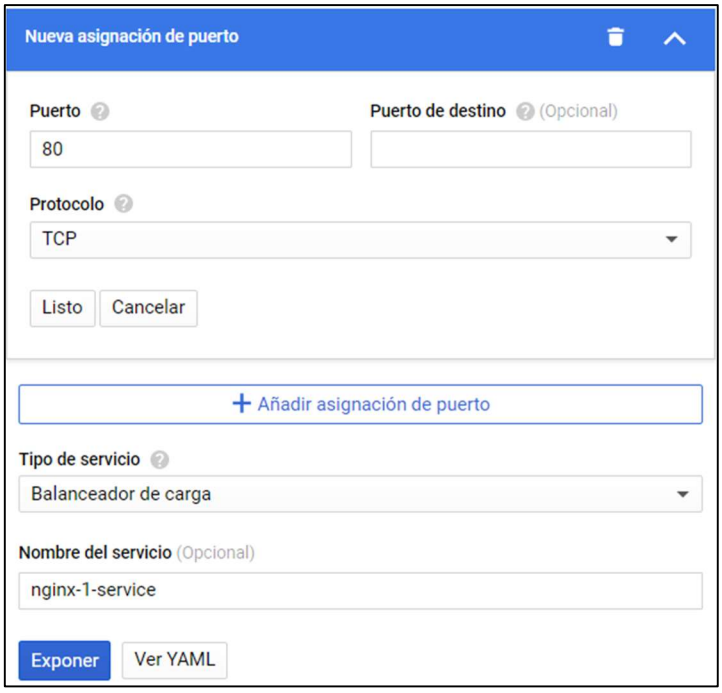

Figura VI.8 Asignación de puertos para exposición del trabajo.

Las pruebas realizadas en la Nube se enfocaron en el servidor, se obtuvieron los tiempos que tardan en realizar tres de las operaciones de interés, estas son: la inferencia del modelo CNN de detección de objetos, la redimensión de la imagen y el envío-recepción de datos. Los tiempos son medidos al momento de correr el programa cliente y se guardan en un archivo de texto, como se observa en la figura 6.9. Se hablará más a detalle de estas pruebas en el capítulo de análisis de resultados.

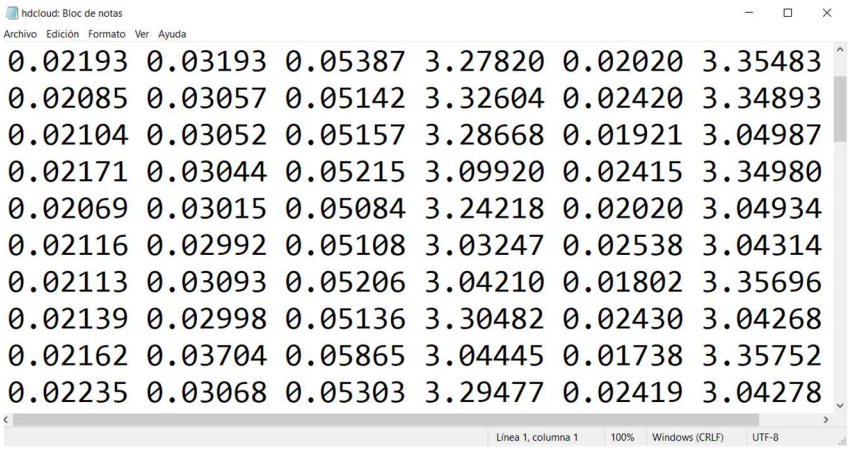

Figura VI.9 Archivo de texto con los tiempos obtenidos de una prueba en CC.

6.3 Cómputo en la Niebla

Para la capa de FC se hicieron pruebas en los dispositivos de la Intel NUC y RPi donde cada uno fungió como un microservidor que analiza las imágenes que se le envían a través del protocolo ZMQ. Las diferencias que se encontraron para crear un contenedor Docker para estos equipos fueron las arquitecturas de la CPU, así que se hicieron dos contenedores para cada dispositivo, pero utilizando el mismo programa en ambas imágenes. Para la computador Intel NUC se corrió el contenedor, el cual se mencionó en el capítulo pasado (§5.4.3), usando los comandos de línea de Docker y dependiendo a qué procesador se enviaban los datos para hacer la inferencia del modelo CNN eran los parámetros del comando de ejecución que se utilizaban. Para el CPU se ingresaba el siguiente comando.

```
Docker run --rm -p 5555:5555 [nombre del contenedor]
```

Donde --**rm** significa que el contenedor se elimina del sistema una vez haya finalizado la ejecución del contenedor y el parámetro P especifica los puertos que se abrirán en el sistema para que se pueda comunicar con el contenedor a través de la red, el primer puerto es el del sistema al que los clientes se conectaran para enviar las solicitudes de comunicación y el segundo puerto es el que está configurado en el servidor que se desarrolló en Python y se transfirió al contenedor.

Las pruebas que se realizaron para la computadora Intel NUC consisten en el uso de este dispositivo como servidor y que esté conectado a la red local para observar las diferencias de lo que se tarda en realizar el proceso de enviar las imágenes y recibir los resultados de la inferencia. Al igual que las pruebas en CC, se hicieron pruebas con distintas resoluciones para la cámara de video. Al tener la posibilidad de cambiar el medio de conexión entre cliente/servidor se usaron distintas configuraciones para tener una conexión por Wifi o Ethernet y sus combinaciones. En la figura 6.10 se observa uno de los archivos de texto generados con los tiempos para obtener los resultados de la inferencia del programa cliente con la computadora Intel NUC como servidor.

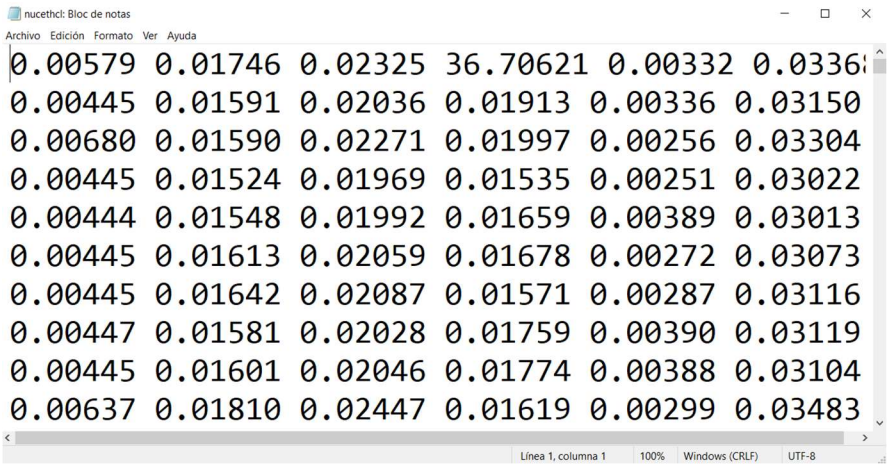

Figura VI.10 Archivo de texto con los tiempos obtenidos de una prueba en la NUC.

Las pruebas realizadas en la RPi son similares a la computadora Intel NUC, sin embargo, al no contar con una CPU capaz de correr los modelos CNN con OpenVINO se usaron los aceleradores neuronales NCS 1 y NCS 2. Otra diferencia fue la de no correr todas las pruebas con Docker, solamente una se corrió con Docker, con el fin de obtener los tiempos más bajos y

esta prueba con Docker es para observar que tanto retarda la obtención de los resultados. Para usar las Movidius con Docker se otorgaron permisos para el acceso a los periféricos USB, como se mencionó en un capítulo anterior §5.4.3. El comando que se utilizó es el que se muestra a continuación.

```
Docker run --rm -p 5555:5555 --priveleged
--device=/dev/bus/usb/001/XXX [nombre del contenedor]
```

En la figura 6.11 se puede visualizar la terminal con el comando mencionado anteriormente, para correr la imagen Docker en la RPi, que no difiere a lo utilizado en la computadora Intel NUC, excepto por el nombre de la imagen y los permisos para los periféricos USB. Una vez ejecutado la imagen Docker se empieza a correr el servidor dentro del contenedor y con ello se puede correr el programa cliente desde otro equipo para enviar las imágenes y recibir los resultados de la RPi.

Figura VI.11 Terminal para ejecutar Docker en la RPi.

Una vez iniciada cada una de las pruebas se genera un archivo de texto con los tiempos para ser analizarlos y obtener algunas conclusiones de cómo usar el programa en FC y compararla con las demás capas de la arquitectura. Con esto completamos las pruebas para FC de un sistema de detección y localización de personas con un flujo continuo de video en tiempo real del aula inteligente. Se hablará con más detalle sobre las pruebas en el capítulo 7.

6.4 Cómputo en la Frontera

Las pruebas en la frontera se realizaron en la misma RPi, la diferencia es que no se desarrolló un servidor para realizar las inferencias del modelo CNN de detección de objetos, sino que tanto la obtención de las imágenes como el procesamiento de estas se generaron en el mismo programa. Para que pueda haber similitud en las pruebas realizadas con las otras arquitecturas se implementó el mismo esqueleto del programa anterior, sin embargo, la función que se encarga de la inferencia no manda la imagen a un servidor, sino que lo procesa ahí mismo.

La prueba consiste en tomar una muestra de 500 imágenes de un video adquirido en vivo y obtener los tiempos que toma realizar la inferencia. De esa muestra se toman 100 datos para realizar la gráfica y obtener promedios. Para obtener los tiempos de inferencia se agregaron funciones para mostrar en pantalla el tiempo de ejecución del programa y redirigir la salida a la consola hacia un archivo de texto, como se puede observar en la figura 6.12.

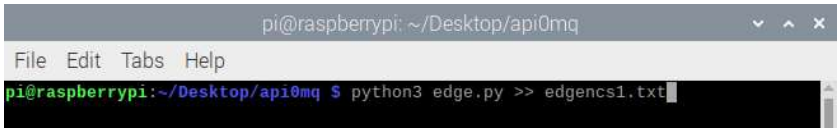

Figura VI.12 Terminal para ejecutar el programa para EC en la RPi.

Una vez terminada la prueba podemos observar el archivo de texto con los tiempos de inferencia. En este caso se realizó con la NCS 1, como se ve en la figura 6.13. Esto se repite para correr el modelo CNN con la NCS 2. Para ver las diferencias al usar la RPi como un servidor (en FC) o como un dispositivo que procesa los datos en el mismo lugar en que se generaron (en EC).

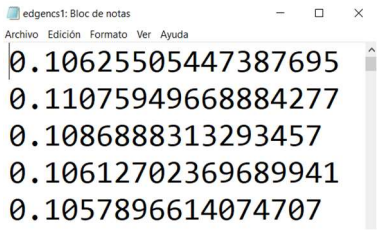

Figura VI.13 Archivo de texto de las pruebas en EC.

VII. ANÁLISIS DE RESULTADOS

Los resultados obtenidos de las pruebas en cada una de las plataformas de hardware de cada uno de los nodos FEC nos permiten determinar en qué capa de la arquitectura osmótica es más conveniente realizar cada uno de los procesos para obtener las inferencias del modelo CNN de detección de objetos con la menor latencia posible. Las pruebas realizadas se enfocaron principalmente en los servidores que se encargaron de realizar la inferencia del modelo CNN y de redimensionar las imágenes de video capturadas. En las pruebas realizadas para CC y FC se midieron seis tiempos diferentes, el primero es la redimensión, el segundo la inferencia, el tercero el total de las dos anteriores, el cuarto es el tiempo en el que se tarda el servidor en ejecutar el comando de recibir una nueva imagen, el quinto es el tiempo en que se tarda en ejecutar el comando para enviar una imagen y el sexto es el tiempo total en el que se tardará en obtener un resultado del modelo CNN, desde que se envía la imagen. El cuarto y el quinto no se utilizaron para realizar un análisis por el tipo de protocolo de comunicación en el cual no tiene una comunicación de confirmación de los datos recibidos o enviados a su destino, por lo que el tiempo no es exacto. Se utilizaron los otros cuatro tiempos para analizar el tiempo de los procesos dentro del servidor y el tiempo total en que el cliente obtiene los resultados de la inferencia del servidor. El proceso que se realizó para la toma de los tiempos fue por medio del servidor que envía los tiempos que tomó realizar la redimensión de la imagen y la inferencia al cliente, el cliente registra la latencia obtenida en realizar todo el procesamiento del servidor y así obtener el tiempo de envío y recepción de datos con la diferencia entre el tiempo total en que se realiza el proceso de la obtención de datos y la suma de los tiempos de inferencia y redimensión en el servidor. Hay que destacar que en el proceso del envío y recepción de datos está un proceso en el que serializa la imagen para enviarla y realiza el proceso inverso al recibirla, como se explicó anteriormente (§5.3).

Antes de hablar de las pruebas se debe tomar en cuenta las restricciones que se presentaron en estas mismas. Con los tiempos que se mostraron anteriormente se usaron muestras de 100 datos para cada una de las pruebas para realizar las gráficas y los promedios correspondientes. Se acondicionaron estos datos para evitar las anormalidades o transitorios que pudieran entorpecer la estadística de estas pruebas, uno de los condicionantes es evitar los datos que se generan al principio, es de esperarse que se tarde más tiempo durante la inicialización del

programa, como se puede apreciar en los puntos encerrados con círculos de la figura 7.1. Otro de los condicionantes es evitar los picos anormales de las gráficas, que por motivos de la conexión a Internet por Wifi o Ethernet se puede encontrar valores efímeros que pueden aletargar el envío y recepción de datos y producen picos anormales en la gráfica como se observa en los puntos encerrados en triángulos de la figura 7.1. Esto también se puede deber a recursos utilizados por el equipo que está realizando procesos ajenos al programa desarrollado.

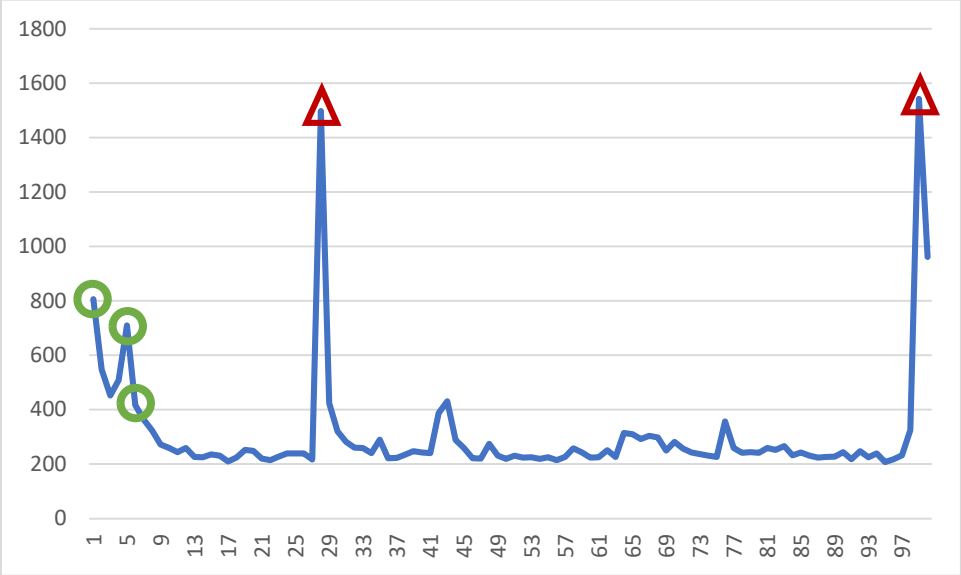

Figura VII.1 Gráfica con datos sin acondicionar.

7.1 Análisis y discusión de los resultados de Cómputo en la Nube

Para obtener las gráficas se usaron 100 muestras del tiempo en que se tarda en obtener los resultados de la inferencia desde el momento en que el cliente envía la imagen de la videocámara. La videocámara hace una captura en vivo dentro del aula donde se encuentra una persona caminando de un lado a otro. Estas pruebas se realizaron varias veces para observar algún patrón o particularidad del tiempo de procesamiento o para poder observar las diferencias entre los tipos de conexión como lo son el Wifi y el Ethernet. En la figura 7.2 se observa la gráfica de los tiempos en que se tarda en obtener los resultados de una inferencia obtenidos por cada una de las pruebas en CC y en el caso del uso de Wifi existe más inestabilidad en comparación con una conexión Ethernet.

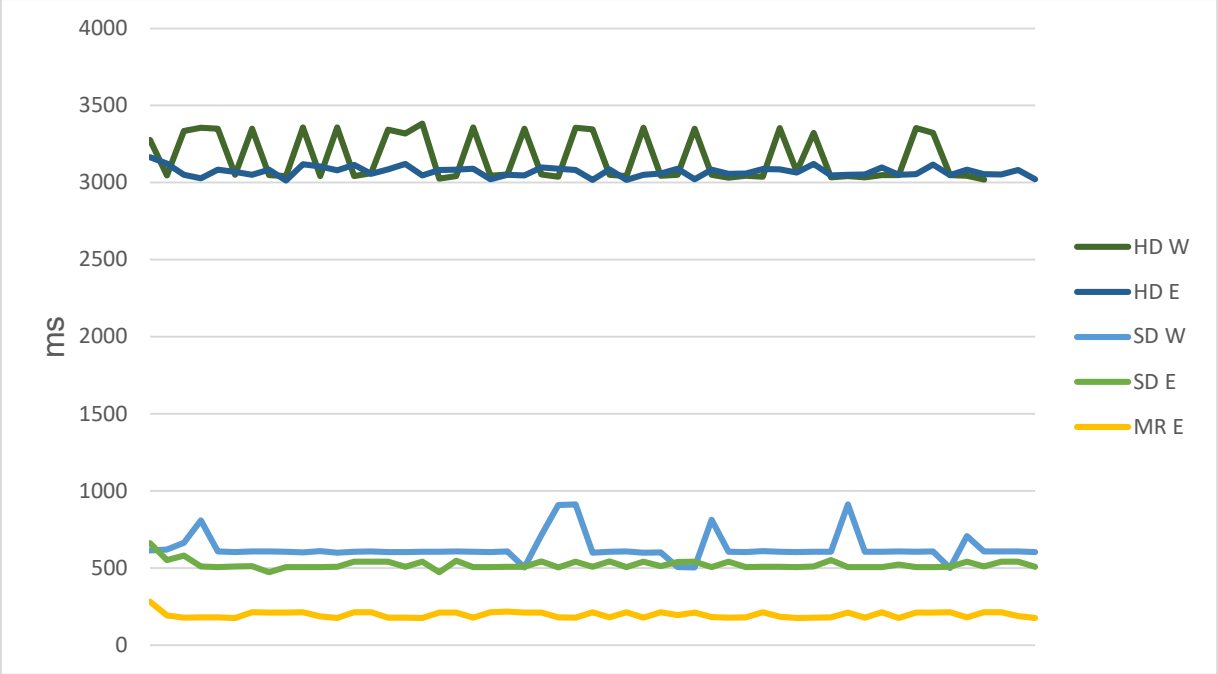

Figura VII.2 Gráfica de tiempos obtenidos de las pruebas para CC.

Los promedios de cada una de las pruebas se muestran en la tabla 7.1, la terminología de las pruebas se vieron en la sección §6.1. Se puede observar que el envío de datos con más peso tiende a tomar más tiempo para obtener los resultados de la inferencia del modelo CNN y entre más ligero es la imagen más acelerado es el proceso de obtener los resultados de la detección de objetos.

Tabla VII.1 Promedios de las pruebas de Cómputo en la Nube.

Prueba	Promedio (ms)	Promedio (*fps*)
HD W	3164.4	0.316
HD E	3064.9	0.326
SD W	626.32	1.59
SD E	522.04	1.92
MR E	199.92	5.02

Aparte del tiempo en que se tarda en enviar la imagen y recibir los resultados, también se tomó el tiempo que toma realizar los procesos internos del servidor que son la inferencia y la redimensión de la imagen, dando así el tiempo del envío y recepción de datos con la diferencia de estos tiempos y el tiempo en que se tarda obtener los resultados de la inferencia en el cliente

que se mencionó anteriormente. Con estos datos se graficó el porcentaje que consumen los procesos del servidor y el envío-recepción de datos, como se muestra en la figura 6.9. En la gráfica también se puede observar los tiempos promedio de cada uno de los procesos que se llevan a cabo para obtener los resultados del modelo CNN de detección de objetos. La suma de estos tiempos nos da el tiempo de su respectiva prueba que se muestra en la tabla 7.1.

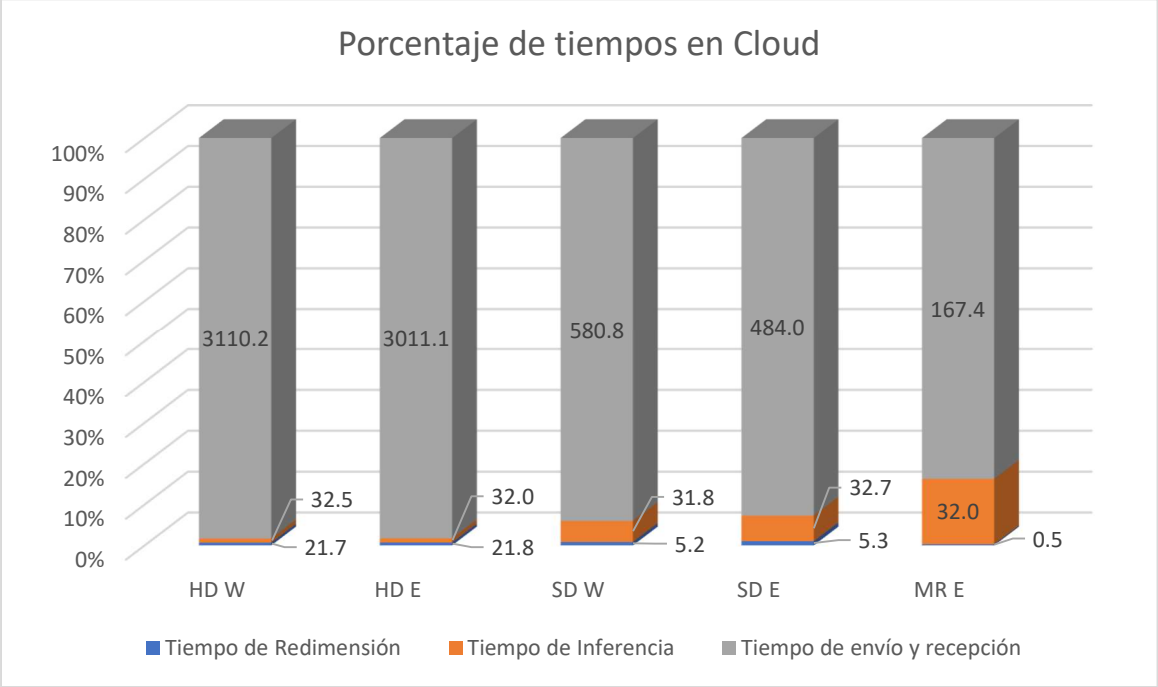

Figura VII.3 Gráfica porcentual de los tiempos en CC. (tiempos en milisegundos)

Como lo vimos anteriormente la resolución de la imagen es un factor determinante en el tiempo para obtener la inferencia. En este caso, el tiempo que toma procesar una imagen HD es entre 5 y 6 veces más tiempo que para una imagen SD y la imagen MR se procesa entre 15 y 16 veces más rápido comparado con la imagen HD. Con la resolución MR incluso sobrepasamos el límite mínimo de 2 *fps* que se buscan para este trabajo (§1.4.3). Durante las pruebas se encontraron algunos problemas con la conexión del servidor en los servicios de GCP debido a los bloqueos del cortafuegos del sistema de internet del instituto que bloquean las IP desconocidas y algunos puertos. Esto se arregló permitiendo que el cortafuegos dejara pasar las conexiones de IP desconocidas, sin embargo, esto abre un problema de seguridad del sistema de internet por lo que también se pudo corregir aceptando los riesgos de conexión desde el equipo.

7.2 Análisis de resultados de Cómputo en la Niebla

Las pruebas en FC se realizaron en la computadora Intel NUC y la RPi, con la diferencia de que en la computadora Intel NUC se ejecutó el modelo CNN de detección de objetos en la CPU y en la RPi se ejecutó en los procesadores neuronales NCS1 y NCS2. Al momento de analizar los tiempos se encontraron algunas discrepancias, ya que al correr el modelo CNN en los dispositivos como la computadora Intel NUC o la RPi, el tiempo para completar los cálculos de la inferencia del modelo CNN se ve afectado por otros procesos ajenos al programa en la CPU del dispositivo como el proceso que se lleva acabo para el funcionamiento del monitor o el del sistema de archivos. Otra de las causas de estos picos en las gráficas es la conexión del internet por medio del Wifi, ya que existe una gran variación a diferencia de las pruebas realizadas con Ethernet, como se puede observar en la figura 7.4.

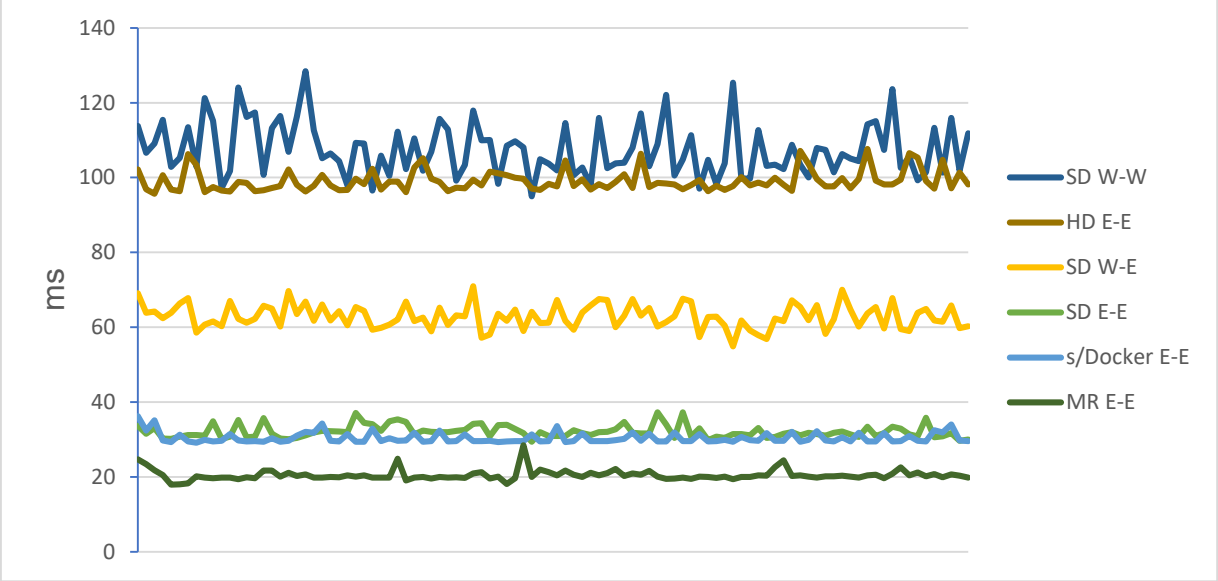

Figura VII.4 Gráfica de tiempos obtenidos de las pruebas para FC en la NUC.

En la tabla 7.2 podemos observar los promedios de los tiempos de cada una de las pruebas mostradas en la gráfica de la figura 7.4. Todas estas pruebas se realizaron con Docker a menos que se indique lo contrario. La prueba que obtuvo mayor tiempo en la obtención de los datos fue la que consistió en enviar una imagen de 640x480 pixeles a la NUC, que hace la función de servidor, y además lleva una conexión inalámbrica Wifi entre el cliente y el servidor. Sin

embargo, sigue siendo menor en comparación con el menor tiempo de las pruebas de CC que fue de 199.92 ms, dando un total de 5.02 *fps*.

Tabla VII.2 Promedios de las pruebas de la NUC en FC.

Prueba	Promedio (ms)	Promedio (*fps*)
SD W-W	111.86	8.64
HD E-E	99.10	10.09
SD W-E	62.87	15.90
SD E-E	32.08	31.17
SD s/Docker E-E	30.39	32.90
MR E-E	20.45	48.90

Al igual que con las pruebas en CC, se obtuvieron los tiempos de los procesos internos del servidor. En la gráfica 7.5 se pueden apreciar los resultados obtenidos. El tiempo de inferencia fluctúa más que en CC con un rango entre 15.7 a 20.3 ms, pero es menor a los 31-32 ms que se obtuvieron en CC. La prueba que se realiza sin Docker nos muestra cómo es que tanto la inferencia y la redimensión de la imagen se tardan menos en comparación de la prueba realizada con Docker y con el mismo tipo de conexión, debido que la inicialización de los contenedores hace uso del CPU para correr el sistema virtualizado dejando con menos recursos disponibles para realizar las operaciones necesarias del sistema del servidor.

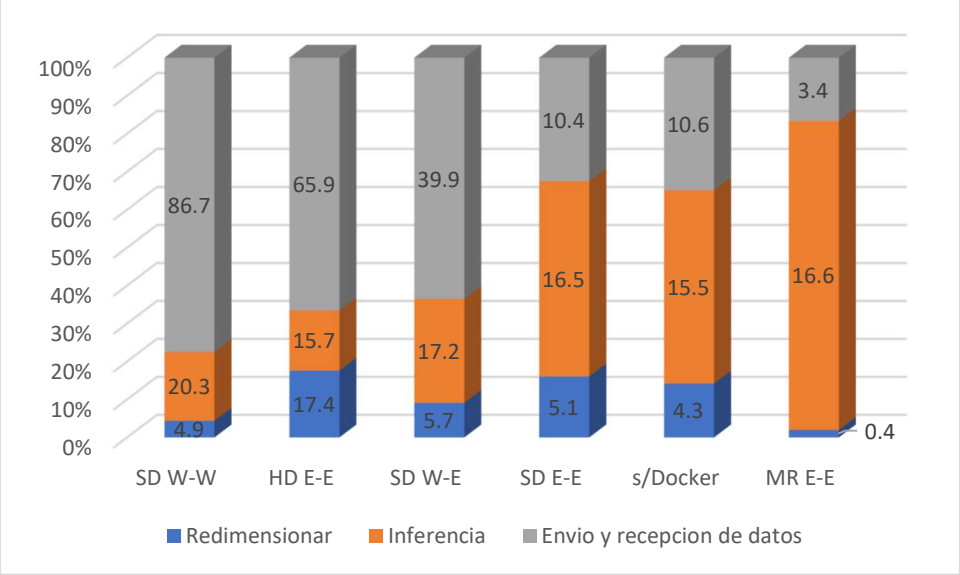

Figura VII.5 Gráfica porcentual de los tiempos en la NUC. (tiempos en milisegundos)

Las pruebas realizadas a la RPi se basaron en las anteriores realizadas en la computadora Intel NUC con la diferencia de que el modelo CNN se ejecutó en los procesadores neuronales, principalmente en la NCS2. Las pruebas se realizaron sin Docker, a menos que se indique lo contrario, esto con el propósito de obtener los tiempos más bajos posibles. Al igual que en las pruebas en las plataformas anteriores, el uso de Wifi produjo una inestabilidad e incluso más pronunciada que en los anteriores, como se puede observar en la figura 7.6, esto debido a que la RPi tiene menos control con la conexión Wifi.

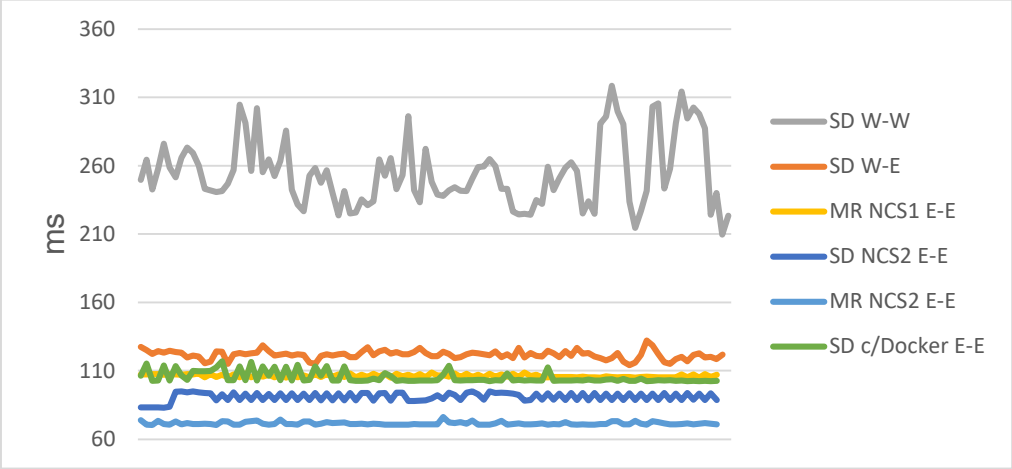

Figura VII.6 Gráfica de tiempos en FC en la RPi.

En la Tabla 7.3 se encuentran los promedios del tiempo en que se tarda en obtener los resultados del servidor y se observa que la manera más rápida en realizar este proceso es por medio de la NCS2 con las imágenes a una resolución del modelo (MR) y ambos dispositivos conectados por Ethernet, dando como resultado 13.99 *fps*. Aunque este tiempo no supera al de la computadora Intel NUC si lo hace a los obtenidos en CC.

Tabla VII.3 Promedio de las pruebas de la RPi en FC.

Prueba	Promedio (ms)	Promedio (*fps*)
SD W-W	262.57	3.80
SD W-E	121.8	8.20
SD E-E	90.74	11.02
SD c/Docker E-E	105.26	9.50
MR NCS1 E-E	106.33	9.40
MR NCS2 E-E	71.46	13.99

Dividiendo estos tiempos en los diferentes procesos que se ejecutan para obtener los resultados del servidor observamos que el proceso más tardado es la inferencia, como se observa en la figura 7.7. En la última barra podemos observar que prácticamente es solo la inferencia ocupa alrededor del 94% del tiempo.

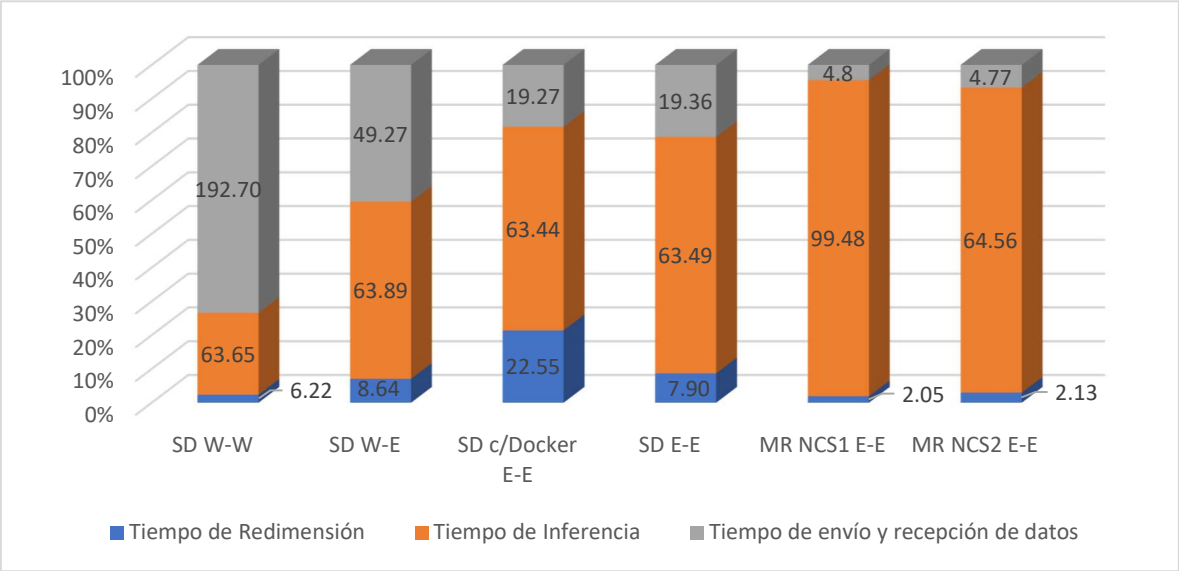

Figura VII.7 Gráfica porcentual de los tiempos de la RPi. (tiempos en milisegundos)

Algo que destacar en esta gráfica es que al utilizar Docker el tiempo de redimensión de la imagen aumenta más del doble pero el tiempo de inferencia se mantiene igual, ya que ésta se realiza en una unidad aparte mientras que la redimensión utiliza la CPU del dispositivo y puede saturarse con otros programas ajenos, incluyendo Docker, provocando que tarde más en realizar los procesos.

7.3 Resultados de Cómputo en la Frontera

En las pruebas de Cómputo en la Frontera se utilizó la RPi y se eliminó la arquitectura cliente/servidor de manera que la RPi corra todos los procesos necesarios para ejecutar el modelo CNN para la detección de objetos. Al poder correr todos los procesos en el mismo dispositivo el tiempo de redimensión ya no obstaculiza al tiempo total de la inferencia de la imagen, ya que se procesa junto con la obtención de la imagen, por lo tanto, no interfiere con la

obtención del resultado con el modelo CNN. Así que únicamente se obtuvo el tiempo de la inferencia en los dos procesadores neuronales NCS1 y NCS2. Como se muestra en la figura 7.8.

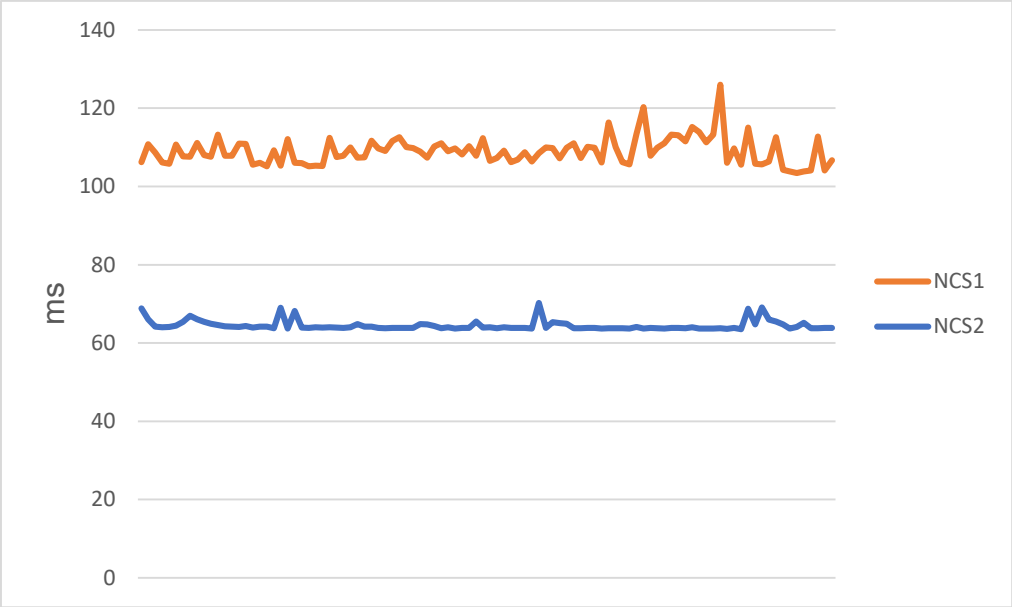

Figura VII.8 Gráfica de tiempos de Cómputo en la Frontera.

Los promedios mostrados en la tabla 7.4 se calcularon con los datos de la figura 7.8. El promedio más alto es de 15.52 *fps* y supera al más alto de la RPi en FC, pero es más bajo que el de un equipo con mayor poder de cómputo que es de 48.9 *fps*. Para el flujo de video del aula inteligente no representa gran diferencia, ya que los cambios que se producen en el aula, captados por la videocámara, son muy lentos. Solamente cuando pasan personas corriendo se puede apreciar en el flujo de video la diferencia entre los 48.9 *fps* y los 15.52 *fps*. Por lo tanto, la RPi cumplió con los requerimientos (§1.4.3) que buscamos en este trabajo que es un reconocimiento en tiempo real de las personas en un aula.

Tabla VII.4 Promedios de las pruebas de Cómputo en la Frontera.

Prueba	Promedio (ms)	Promedio (*fps*)
SD NCS1	109.03	9.17
SD NCS2	64.43	15.52

VIII.CONCLUSIONES Y TRABAJO A FUTURO

Esta tesis es un estudio de investigación y experimentación acerca de las capacidades de la arquitectura osmótica para ejecutar modelos de detección de objetos en las diferentes capas que las componen (CC, FC y el EC). Se realizaron pruebas comparativas del tiempo de inferencia para cada imagen de video capturada por una videocámara que está monitoreando en tiempo real distintas zonas de interés para controlar la iluminación del aula inteligente. A continuación, se presenta las conclusiones derivadas de los resultados obtenidos en las pruebas realizadas en las distintas capas de la arquitectura osmótica, así como lo referente al desarrollo del API para un aula inteligente, la ejecución de modelos CNN, el proceso de la virtualización en Docker y el control de dispositivos IoT.

8.1 API del Aula Inteligente

El API para dispositivos IoT del aula inteligente ejecutado en todos los nodos FEC brindó un gran apoyo en la implementación de un sólo código para efectuar la inferencia de modelos CNN para la detección de personas y apoyo en gran medida en la implementación de los programas cliente/servidor encargados de realizar el control de los dispositivos IoT, a pesar de grandes diferencias en las arquitecturas de hardware de los nodos. El API evitó la redundancia de código que fue evidente en las primeras implementaciones previas a su diseño.

Su desarrollo puede considerarse en realidad un conjunto de APIs, en un futuro cercano puede llegar a ser un *framework* más complejo, que desde sus primeras versiones fue de gran ayuda para el empaquetamiento de las funciones que se necesitan para crear un programa que ejecute modelos CNN para la detección de objetos y poder crear una conexión entre programas para tener una facilidad de comunicarse con otros clientes y nodos FEC. Las APIs, que corresponden a cada uno de los dispositivos IoT y los *frameworks* para DL utilizados para correr los modelos CNN, están contenidas en un solo archivo que contiene más de 700 líneas de código. Gracias a este API fue posible desarrollar los programas de los servidores (nodos FEC) en menos de 50 líneas de código y para el cliente en alrededor de 100 líneas. Por lo tanto, las APIs desarrolladas cumplieron con el objetivo de ayudar a crear programas reusando código, para

reducir el número de líneas de código y mejorar las interfaces entre los módulos de inferencias CNN y la comunicación en red entre los nodos y dispositivos IoT. Para mejorar esta API se pueden agregar más funciones para que sea compatible con otros *framworks* como Pytorch, MXNet, entre otros. También se puede mejorar el acoplamiento de más dispositivos IoT, además de los ya implementados en este trabajo.

8.2 Ejecución del modelo CNN en las capas de la arquitectura osmótica

Se implementó un sistema de IoT para correr un modelo CNN para la detección de objetos en una arquitectura osmótica de tres capas, validadas a través de un conjunto de pruebas para comparar su comportamiento, latencias y rendimiento medido en frames-por-segundo (*fps*). Aparte del API se desarrollaron otros programas para crear un modelo cliente/servidor y en el caso de EC solo un programa para realizar el sistema de control del aula inteligente. Estos programas se realizaron con la programación concurrente para la visualización del video, que se usó en los nodos de la arquitectura osmótica para obtener una mejor visualización de la detección de personas en tiempo real (§5.4.2). En las pruebas realizadas en CC fue evidente y muy inferior el rendimiento (*fps*) en comparación con las otras capas (FC, EC), resultando un rendimiento cercano a los 2 *fps*, lo cual resulta inadecuado para la aplicación de control del aula inteligente. También se observó que la resolución de la imagen influye mucho en el tiempo para obtener una inferencia del modelo CNN en la Nube, por lo que enviar una imagen correctamente redimensionada ofrece la mejor opción en rendimiento en cualquier capa de la arquitectura, pero tienen desde luego un mayor impacto en la capa de la Nube.

En la capa de la niebla (FC), se realizaron pruebas con la computadora Intel NUC estas fueron las que ofrecieron mejores resultados en términos de rendimiento (*fps*) para detectar personas (*fps*), debido en buena medida a la capacidad de procesamiento de su CPU Pentium multi-core de alto rendimiento llegando en el mejor de los casos, casi a los 50 *fps*. La minicomputadora Raspberry Pi obtuvo el mejor resultado usando un acelerador neuronal Movidius, logrando procesar 14 *fps*, por lo que podemos concluir que la capa en la niebla proporciona un rendimiento apropiado para una aplicación de IoT + DL capaz de detectar personas y controlar dispositivos IoT en tiempo real, como la aplicación desarrollada en esta

tesis para controlar un aula inteligente (Pacheco, 2018a). Entre las observaciones más relevantes respecto a la etapa de inferencias CNN, fue notoria la variación obtenida en los tiempos de inferencia correspondientes a la capa de la niebla (FC), al aparecer debido a las diversas tareas que ejecuta concurrentemente el sistema operativo de la computadora Intel NUC y durante las pruebas. Si el usuario ejecuta otras aplicaciones (e.g. sistema de archivos o los gráficos de la pantalla), esto afecta significativamente el rendimiento del sistema, por lo que es importante mencionar que, en la operación actual del sistema del aula inteligente, los nodos FEC habilitados deben ser equipos exclusivamente dedicados a ejecutar los programas del sistema IoT implementado para lograr un funcionamiento adecuado.

Por último, las pruebas de rendimiento realizadas dentro de la capa en la frontera (EC) ofrecieron una mejora relativamente marginal en el rendimiento respecto a la capa en la niebla, al alcanzar alrededor de 16 *fps* utilizando la misma computadora RPi + acelerador, con la diferencia de que un nodo EC trabaja todo su procesamiento de manera local, es decir, no requiere enviar las imágenes que captura a un servidor FC para realizar la detección de personas. De esta forma podemos concluir que la capa de la arquitectura IoT osmótica que ofreció el mayor rendimiento fue la capa ubicada en la frontera de la red, es decir, los nodos de *Edge Computing* (EC). Por la frecuencia baja en que suceden los cambios en las imágenes de la videocámara dentro del aula, la visualización del video en tiempo real no es perjudicada, a menos que una persona pase corriendo frente la videocámara.

Como sugerencias para trabajo a futuro, consideramos que es posible intercambiar la ejecución de diferentes modelos CNN en los mismos nodos FEC + aceleradores (i.e. *deep neural network scheduler*), donde sea posible detectar personas y a la vez, correr otros modelos CNN para determinar posturas, actividades, detectar y reconocer rostros, emociones o gestos, como se observa en la figura 8.1, que permitirán ofrecer un aula o edificio más inteligente, autónomo, sensitivo al contexto, reductor de consumo energético, más seguro y/o automatizado.

Figura VIII.1 Vectorización de los miembros del cuerpo y de características faciales.

La primera implementación del aula inteligente derivada en su conjunto del trabajo del proyecto de investigación del Laboratorio de Aprendizaje Móvil y dos tesis de maestría, la presente y (Flores, 2019), puede servir de plataforma y abrir la posibilidad para explorar a futuro diversos temas y problemas afines, tales como: la incorporación de otros dispositivos y sensores, desde detectores de presencia, temperatura, proximidad (*beacons*), hasta micrófonos y cámaras RGBD (*multimodal deep learning*); mejorar los APIs propuestas en esta tesis para integrar un *framework* de IoT genérico y extensible, donde sea posible derivar nuevas subclases para incorporar nuevos dispositivos IoT y nodos FEC minimizando el código de implementación de aplicaciones y maximizando la interoperabilidad de los mismos; incorporar múltiples modelos CNN optimizados para ser despachados en un clúster de nodos FEC desarrollando un planificador de procesos concurrentes para realizar inferencias de modelos CNN (*DL scheduler*); explorar nuevos dispositivos aceleradores (Google Coral, NVIDIA® Jetson Nano), otras arquitecturas de procesamiento (NPU, GPU, FPGA) y nuevas técnicas de aceleración de procesos de inferencia CNN en dispositivos IoT limitados, de bajo costo y bajo consumo de energía (*energy harvesting*); escalar el sistema para soportar múltiples aulas con el mínimo número de servidores o nodos FEC (*smart buildings*); implementar puentes y compuertas de red inteligentes (*smart gateways*) especializadas en IoT explorando nuevos esquemas de conectividad (Mekki et al., 2019), protocolos y bandas de radiofrecuencia (Bluetooth BLE 5, 5G, ZigBee, LoRA, SigFox, LPWA, NB-IOT); incorporar esquemas de monitoreo, analítica de datos, encriptamiento y seguridad para los dispositivos y todo el sistema IoT; desarrollar aplicaciones móviles inteligentes (*smart mobile edge computing, MEC + DL*) para supervisar y controlar un edificio inteligente, entre muchas otras interesantes y novedosas posibilidades (Pacheco, 2018). Esta implementación ha abierto las puertas para próximos trabajos con otros

tipos de pruebas para la arquitectura de FC donde con distintos tipos de modelos ejecutándose en un solo servidor pueda llegar a usar aplicaciones multimodales, con la capacidad de llegar a tener una amplia percepción del entorno con estos modelos para que el sistema reconozca la actividad que está realizando la persona, los gestos o incluso lo que dice la persona para poder dar más atención al control de los dispositivos del aula.

8.3 Virtualización: El Uso de Contenedores Docker

El uso de la tecnología de contenedores de Docker para el desarrollo del sistema de aula inteligente fue muy útil para empaquetar e integrar todos los *frameworks*, librerías y archivos a una sola imagen Docker y poder correr dicha imagen en contenedores habilitados para los diferentes nodos FEC y el servicio en la nube de Google (GCP). Sin embargo, la imagen debe ser compatible con la arquitectura del CPU y el equipo debe correr alguna distribución de Linux o Windows. Una imagen creada en un equipo con una CPU x86 no puede correr en un equipo con una CPU ARMv7 y tampoco una imagen creada en Linux no puede funcionar correctamente en un equipo con Windows. Por lo tanto, es necesario crear una imagen diferente en esos casos y el contenido de esas imágenes puede diferir en algunas librerías para poder correr los programas desarrollados para el sistema del aula inteligente.

Los tiempos de ejecución del sistema del aula inteligente fueron afectados con el uso de contenedores Docker en los dispositivos de FC, debido a los procesos qué se llevan a cabo para la ejecución de Docker que usa el CPU del dispositivo y deja menos recursos para ejecutar los procesos del aula inteligente. Aunque la diferencia de usar Docker o no es de aproximadamente 1.5 *fps*, si lo comparamos con las ventajas de poder mover el sistema a cualquier nodo de la arquitectura es un precio que vale la pena pagar.

En próximos proyectos se recomienda hacer uso de estos contenedores Docker y explorar los nuevos esquemas de virtualización para dispositivos IoT para crear un sistema aún más complejo con la habilidad de correr varios de estos contenedores y que puedan procesar distintos tipos de modelos CNN para obtener más información del entorno que se esté monitoreando y luego analizar esa información para realizar las acciones adecuadas en ese mismo entorno.

También se puede mejorar la administración de los contenedores para atender la demanda de solicitudes que esté recibiendo el servidor y así aumentar o disminuir la capacidad del sistema según sea conveniente (*load balancing)*, esto con la finalidad de que los clientes reciban una respuesta más rápida a sus peticiones cuando la carga del servidor pueda ajustarse o distribuirse a otros servidores de forma dinámica.

8.4 Control de dispositivos IoT

Durante el uso de los dispositivos IoT se puede apreciar un pequeño retraso en el tiempo que toma recolectar una imagen por el tiempo que consumen para realizar cambios en el estado de estos dispositivos, pero exclusivamente se refiere a cuando existe un cambio en el estado de las áreas en que se detectan personas y esto perjudica exclusivamente a la obtención de las imágenes de la videocámara por lo que la cantidad de cuadros procesados decae de los 60 *fps* a valores entre 58 y 60 *fps.* Este retardo no fue considerado parte de las pruebas de este trabajo y no se profundizó en su estudio, ya que no afectó en gran medida la visualización del video.

El programa que se encarga de mandar los comandos de control a los dispositivos IoT aún necesita algunos cambios, ya que para su funcionamiento es necesario que estos dispositivos ya estén configurados con sus claves de acceso y deben de estar todos en línea, si alguno llega a estar fuera de la red el programa manda una alerta de error por no encontrarse el dispositivo disponible. Se debe crear una función que sea capaz de encontrar estos dispositivos y configurarlos para poder utilizarlos en la aplicación de control como sea conveniente. Además de poder agregar más dispositivos IoT distintos a los usados en este trabajo para que puedan conectarse a la red del sistema de aula inteligente.

En versiones más avanzadas del sistema de aula inteligente, se sugiere crear más opciones de control de los dispositivos IoT para poder manipular intensidad y color de la iluminación, así como contabilizar el número de personas ubicadas en cada región, estimar la distancia de separación entre personas (e.g. contingencia COVID-19) y registrar los datos históricos del sistema, como puede ser el número de personas detectadas, eventos anormales, consumo de energía, etc.

8.5 Conclusiones Generales

Esta tesis ha explorado una arquitectura IoT de Cómputo Osmótico compuesta por tres capas (CC, FC y EC). Una vez implementado el sistema del aula inteligente mediante nodos FEC con apoyo de las APIs genéricas desarrollada, fue posible realizar diversas pruebas de rendimiento estimado en *frames*-por-segundo (fps) para la captura de imágenes de video, proceso de inferencia del modelo CNN, comunicación y control de dispositivos IoT en cada una de las distintas capas. Estas mediciones permitieron realizar un estudio comparativo de dichas capas y realizar una caracterización sistemática de las mismas. Para realizar este estudio comparativo se usaron dispositivos de un aula para controlarlos y automatizarlos, fue necesario desarrollar una API que pudiera controlar cada uno de ellos. Conforme avanzó el desarrollo del trabajo de tesis, estas APIs fueron útiles para realizar las pruebas en las tres capas de la arquitectura IoT. Aunque el problema con el API fue que se realizaron muchos cambios durante el transcurso del trabajo de tesis debido a que algunas de las librerías y *frameworks* para DL tuvieron actualizaciones y las versiones anteriores no eran compatibles. Esto es debido a que este tipo de tecnologías son muy nuevas y tienden a cambiar incluso en cuestión de meses. Por ello es probable que la vigencia de un API para IoT deba revisarse y actualizarse constantemente.

Como conclusión general del presente trabajo, fue posible demostrar que efectivamente la capa correspondiente al borde de la red (*Edge Computing, EC*) de la arquitectura IoT osmótica ofrece un mejor rendimiento para ejecutar inferencias CNN y tomar acciones de control en el menor lapso. Sin embargo, la capa del Cómputo en la Niebla ofrece ventajas adicionales con un rendimiento ligeramente inferior a la capa EC (una diferencia de 1.5 *fps*) que pueden ser de gran valor para soportar a futuro, un sistema IoT más complejo, heterogéneo, escalable, confiable y rentable.

REFERENCIAS

1. Agarwal, S., Terrail, J. O. D., & Jurie, F. (2018). Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. *ArXiv:1809.03193 [Cs]*. Recuperado de http://arxiv.org/abs/1809.03193
2. Al-Gumaei, Khaled, Kornelia Schuba, Andrej Friesen, Sascha Heymann, Carsten Pieper, Florian Pethig, y Sebastian Schriegel. «A Survey of Internet of Things and Big Data Integrated Solutions for Industrie 4.0». En 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 1417-24. Turin: IEEE, 2018. https://doi.org/10.1109/ETFA.2018.8502484.
3. Allanbunch (2013). Beautfulhue. Recuperado el 27 de marzo de 2018, de Github website: https://github.com/allanbunch/beautifulhue.
4. Al-Qamash, A., Soliman, I., Abulibdeh, R., & Saleh, M. (2018). Cloud, Fog, and Edge Computing: A Software Engineering Perspective. 2018 International Conference on Computer and Applications (ICCA), 276–284. https://doi.org/10.1109/COMAPP.2018.8460443
5. Amazon Web Services, Inc. AWS. Recuperado el 18 de septiembre de 2018, de https://aws.amazon.com/es/
6. Athmaja, S., Hanumanthappa, M., & Kavitha, V. (2017). A survey of machine learning algorithms for big data analytics. 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), 1–4. https://doi.org/10.1109/ICIIECS.2017.8276028
7. Biehl, Matthias (2016). API Design. API-University press. Primera edición.
8. Bruce, Morgan & Pereira, Paulo (2019). Microservice in Action. Manning Publications Co.
9. Buduma, N., & Locascio, N. (2017). *Fundamentals of Deep Learning* (1a ed.). Canada: O'Reilly Media Inc.
10. Bughin, J., Seong, J., Manyika, J., Chui, M., & Joshi, R. (2018). Notes from the AI frontier Modeling the impact of AI on the world economy. Recuperado de McKinsey Global Institute website: https://www.mckinsey.com/~/media/McKinsey/Featured%20Insights/Artificial%20

Intelligence/Notes%20from%20the%20frontier%20Modeling%20the%20impact%20of%20AI%20on%20the%20world%20economy/MGI-Notes-from-the-AI-frontier-Modeling-the-impact-of-AI-on-the-world-economy-September-2018.ashx

11. Burns, B. (2018). Designing Distributed Systems. O'Reilly Media, Inc. 165.
12. Buyya, R. & Srirama, S. (2019). Fog and Edge Computing: Principles and Paradigms, Hoboken,NJ,USA: John Wiley & Sons, Inc.
13. Chang, C., Narayana Srirama, S., & Buyya, R. (2017). Indie Fog: An Efficient Fog-Computing Infrastructure for the Internet of Things. Computer, 50(9), 92–98. https://doi.org/10.1109/MC.2017.3571049
14. Chen, C., Ruan, Y., & Liao, Z. (2018). iOccupancy: An Investigation of Online Occupancy-driven HVAC Control in Campus Classrooms. Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing - CitiFog'18, 25–28. https://doi.org/10.1145/3277893.3277900
15. Chiang, M., & Zhang, T. (2016). Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, *3*(6), 854–864. https://doi.org/10.1109/JIOT.2016.2584538
16. Chui, M., Manyika, J., Miremadi, M., Henke, N., Chung, R., Nel, P., & Malhotra, S. (2018). Notes from the AI frontier. Insights from hundreds of use cases. Recuperado de McKinsey Global Institute website: http://www.mckinsey.com/mgi.
17. Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper", Cisco, 2017. Disponible en: https://goo.gl/2aVofM. [Consultado: 16-mar-2018]
18. Cisco, Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. (2015). Recuperado el 26 de marzo de 2019, de Cisco website: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf
19. Dar, A. R., & Ravindran, D. D. (2019). Fog Computing: An Extended Version of Cloud Computing, (7), 7
20. De Buen, O. (2017). Presentación "Eficiencia energética en iluminación". Secretaría de Energía: Comisión Nacional para el Uso Eficiente de la Energía (CONUEE).

Recuperado el 12 de septiembre de 2018, de https://www.conuee.gob.mx/transparencia/Comenor-2017.pdf

21. Eclipse Mosquitto. (2018, enero 8). Eclipse Mosquitto. https://mosquitto.org/
22. ETSI. (2019). Multi-access Edge Computing (MEC); Study on MEC support for alternative virtualization technologies. https://www.etsi.org/deliver/etsi_gr/MEC-DEC/001_099/025/02.01.01_60/gr_MEC-DEC025v020101p.pdf
23. Fakhruddin, H. (2018). Machine Learning in 2019: Tracing The Artificial Intelligence Growth Path. Recuperado de Teks mobile Teknowledge Software website: https://teks.co.in/site/blog/machine-learning-in-2019-tracing-the-artificial-intelligence-growth-path/
24. Fakhruddin, H. (2018). Machine Learning in 2019: Tracing The Artificial Intelligence Growth Path. Recuperado de Teks mobile Teknowledge Software website: https://teks.co.in/site/blog/machine-learning-in-2019-tracing-the-artificial-intelligence-growth-path/
25. Ferreira, H. G. C., Dias Canedo, E., & de Sousa, R. T. (2013). IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and arduino. 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 53–60. https://doi.org/10.1109/WiMOB.2013.6673340
26. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., & Reschke, J. F. (2007). Hypertext Transfer Protocol—HTTP/1.1. https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html
27. Filocamo, B., Galletta, A., Fazio, M., Ruiz, J. A., Sotelo, M. A., & Villari, M. (2018). An Innovative Osmotic Computing Framework for Self Adapting City Traffic in Autonomous Vehicle Environment. 2018 IEEE Symposium on Computers and Communications (ISCC), 01267–01270. https://doi.org/10.1109/ISCC.2018.8538675
28. Flores, Ever A. (2019). Eficientización de Redes Neuronales para Internet de las Cosas sobre Cómputo en la Frontera y en la Niebla (tesis de maestría). Instituto Tecnológico de Chihuahua, Chihuahua, México.

29. G., Mujthaba, Alameen, Abdalla, & Kolhar, Manjur. (2018). *Cloud servers and fog or edge computing with their limitations & challenges,* Journal of Computer Hardware Engineering, vol. 1, núm. 1, pp. 1–7.
30. Geewax, J. J., & Hölzle, U. (2018). Google Cloud platform in action. Manning Publications Co.
31. Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
32. Google (2018a). Coral. Recuperado el 6 septiembre 2018, de https://coral.ai/
33. Google Brain. (2018). Tensorflow. Recuperado el 25 de septiembre de 2018, de Tensorflow website: https://www.tensorflow.org/
34. Google(2018b). Google Cloud Platform. Recuperado el 18 de mayo de 2018, de https://cloud.google.com/
35. H. Song, R. Srinivasan, T. Sookoor y S. Jeschke, Smart Cities: Foundations, Principles and Applications, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2017.
36. Hao, K. (2019). We analyzed 16,625 papers to figure out where AI is headed next. Recuperado de MIT Technology Review website: https://www.technologyreview.com/s/612768/we-analyzed-16625-papers-to-figure-out-where-ai-is-headed-next/
37. Hao, S., Li, D., Halfond, W. G., & Govindan, R. (2012, Junio). Estimating Android applications' CPU energy usage via bytecode profiling. En Green and Sustainable Software (GREENS), 2012 First International Workshop on (pp. 1-7). IEEE.
38. Hao, X., Zhang, G., & Ma, S. (2016). Deep Learning. *International Journal of Semantic Computing*, *10*(03), 417–439. https://doi.org/10.1142/S1793351X16500045
39. Herrera, J., Demir, M. A., Yousefi, P., Prevost, J. J., & Rad, P. (2018). Distributed Edge Cloud R-CNN for Real Time Object Detection. En 2018 World Automation Congress (WAC) (pp. 1–5). https://doi.org/10.23919/WAC.2018.8430385
40. Hunter, K. L., & Cockcroft, A. (2017). Irresistible APIs: Designing web APIs that developers will love. Manning.

41. Hyodo, K (2019). MobileNet-SSD-Realsense. Recuperado el 25 de agosto de 2019, de Github website: https://github.com/PINTO0309/MobileNet-SSD-RealSense#2openvino-ver-corresponds-to-ncs2
42. Hyodo, K. (2018). OpenVINO-YoloV3: YoloV3/tinyYoloV3+RaspberryPi3/ Ubuntu Laptop PC+NCS/NCS2+USBCamera+Python+OpenVINO. Recuperado el 21 de agosto de 2019, de Github website: https://github.com/PINTO0309/OpenVINO-YoloV3
43. Ibarra-Esquer, J., González-Navarro, F., Flores-Rios, B., Burtseva, L., & Astorga-Vargas, M. (2017). Tracking the Evolution of the Internet of Things Concept Across Different Application Domains. Sensors, 17(6), 1379. https://doi.org/10.3390/s17061379
44. IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. (2018). IEEE. https://doi.org/10.1109/IEEESTD.2018.8423800
45. Instituto Nacional de Estadística y Geografía (2014). Estadísticas sobre disponibilidad y uso de tecnología de información y comunicaciones en los hogares, 2013. México: INEGI.
46. Intel Corp. (2018a). Intel® Neural Compute Stick 2. Recuperado el 6 de octubre de 2018, de https://software.intel.com/en-us/neural-compute-stick.
47. Intel Corp. (2018b). Intel® Distribution of OpenVINOTM Toolkit. Recuperado el 25 de octubre de 2018, de Intel website: https://software.intel.com/en-us/openvino-toolkit
48. Intel Corp. (2018c). Intel Movidius Neural Compute SDK. Recuperado el 25 de octubre de 2018, de Movidius GitHub website: https://movidius.github.io/ncsdk/
49. Intel Corp.(2019). Inbtel Distribution of OpenVINO Toolkit. Recuperado el 25 de agosto de 2019, de Intel website: https://software.intel.com/en-us/openvino-toolkit
50. Ionica, M. H., & Gregg, D. (2015). The Movidius Myriad Architecture's Potential for Scientific Computing. IEEE Micro, 35(1), 6–14. doi:10.1109/mm.2015.4
51. Light, R. (s/f). paho-mqtt: MQTT version 3.1.1 client class (1.5.0) [Python; MacOS :: MacOS X, Microsoft :: Windows, POSIX]. Recuperado el 18 de agosto de 2020, de http://eclipse.org/paho

52. Mach, P., & Becvar, Z. (2017). Mobile Edge Computing: A Survey on Architecture and Computation Offloading. IEEE Communications Surveys & Tutorials, 19(3), 1628–1656. https://doi.org/10.1109/COMST.2017.2682318
53. Manic, M., Amarasinghe, K., Rodriguez-Andina, J. J., & Rieger, C. (2016). Intelligent Buildings of the Future: Cyberaware, Deep Learning Powered, and Human Interacting. IEEE Industrial Electronics Magazine, 10(4), 32–49. https://doi.org/10.1109/MIE.2016.2615575
54. Matt Morley. (2013, enero 4). JSON-RPC 2.0 Specification. https://www.jsonrpc.org/specification
55. Mekki, K., Bajic, E., Chaxel, F., & Meyer, F. (2019). A comparative study of LPWAN technologies for large-scale IoT deployment. ICT Express, 5(1), 1–7. https://doi.org/10.1016/j.icte.2017.12.005
56. Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. 7.
57. Mocanu, E., Mocanu, D. C., Nguyen, P. H., Liotta, A., Webber, M. E., Gibescu, M., & Slootweg, J. G. (2017). On-line Building Energy Optimization using Deep Reinforcement Learning. ArXiv:1707.05878 [Cs, Math]. http://arxiv.org/abs/1707.05878
58. Mohammadi, M., & Al-Fuqaha, A. (2018). Enabling Cognitive Smart Cities Using Big Data and Machine Learning: Approaches and Challenges. IEEE Communications Magazine, 56(2), 94–101. https://doi.org/10.1109/MCOM.2018.1700298
59. Morshed, A., P. P. Jayaraman, T. Sellis, D. Georgakopoulos, M. Villari, y R. Ranjan. «Deep Osmosis: Holistic Distributed Deep Learning in Osmotic Computing». IEEE Cloud Computing 4, n.o 6 (noviembre de 2017): 22-32. https://doi.org/10.1109/MCC.2018.1081070
60. Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., & Polakos, P. A. (2018). A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. IEEE Communications Surveys Tutorials, 20(1), 416–464. https://doi.org/10.1109/COMST.2017.2771153
61. N. Buduma y N. Lacascio, Fundamentals of DeepLearning, Sebastopol, CA: O'Reilly Media, 2017.

62. Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., & Ranjan, R. (2018). Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions. IEEE Access, 6, 47980–48009. https://doi.org/10.1109/ACCESS.2018.2866491
63. Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. 2017 IEEE International Systems Engineering Symposium (ISSE), 1–7. https://doi.org/10.1109/SysEng.2017.8088251
64. Nest Developers. (2019). Nest Developers. https://developers.nest.com/?hl=es
65. Nikouei, S. Y., Chen, Y., Song, S., Xu, R., Choi, B.-Y., & Faughnan, T. R. (2018). Real-Time Human Detection as an Edge Service Enabled by a Lightweight CNN. 2018 IEEE International Conference on Edge Computing (EDGE), 125–129. https://doi.org/10.1109/EDGE.2018.00025
66. Pacheco, A., Cano, P., Flores, E., Trujillo, E., & Marquez, P. (2018). A Smart Classroom Based on Deep Learning and Osmotic IoT Computing. *2018 Congreso Internacional de Innovación y Tendencias En Ingeniería (CONIITI)*, 1–5. https://doi.org/10.1109/CONIITI.2018.8587095
67. Pacheco, A., Flores, E., Sánchez, R., & Almanza-García, S. (2018). Smart Classrooms Aided by Deep Neural Networks Inference on Mobile Devices. 2018 IEEE International Conference on Electro/Information Technology (EIT), 0605–0609. https://doi.org/10.1109/EIT.2018.8500260
68. PINE64. (2018). rockpro64. Recuperado el 25 de marzo de 2018, de https://www.pine64.org/rockpro64/
69. Preden, J. S., Tammemae, K., Jantsch, A., Leier, M., Riid, A., & Calis, E. (2015). The Benefits of Self-Awareness and Attention in Fog and Mist Computing. Computer, 48(7), 37–45. https://doi.org/10.1109/MC.2015.207
70. Raschka, S., & Mirjalili, V. (2017). *Python Machine Learning, Second Edition* (Edición: 2nd). Birmingham Mumbai: Packt Publishing.
71. Ray, P. P. (2018). A survey on Internet of Things architectures. Journal of King Saud University - Computer and Information Sciences, 30(3), 291–319. https://doi.org/10.1016/j.jksuci.2016.10.003

72. S. Raschka y V. Mirjalili, Python Machine Learning, Segunda ed., Birmingham, UK: Packt Publishing Ltd., 2017.
73. Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J., Manyika, J., Niebles, J. C., … Bauer, Z. (2018). The AI Index 2018 Annual Report. AI Index Steering Committee, Human-Centered AI Initiative, Stanford University, Stanford, CA.
74. Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J., Manyika, J., Niebles, J. C., … Bauer, Z. (2018). The AI Index 2018 Annual Report. AI Index Steering Committee, Human-Centered AI Initiative, Stanford University, Stanford, CA.
75. Shukla, N. (2018). Machine learning with TensorFlow (K. Fricklas, ed.). Shelter Island, NY: Manning.
76. Simonelli, A (2017). Tiny YOLOv2 in Tensorflow made simple. Recuperado el 24 de agosto del 2018, de Github website: https://github.com/simo23/tinyYOLOv2
77. Singh, S. P., Nayyar, A., Kumar, R., & Sharma, A. (2018). Fog computing: from architecture to edge computing and big data processing. The Journal of Supercomputing. https://doi.org/10.1007/s11227-018-2701-2.
78. Suwimon Vongsingthong, & Smanchat, S. (2014). INTERNET OF THINGS: A REVIEW OF APPLICATIONS AND TECHNOLOGIES. 4, 21, Suranaree Journal of Science and Technology-. https://doi.org/10.14456/sjst.2014.38.
79. Thenas, (2018). Control-ir-mqtt, Recuperado el 16 de octubre de 2018, de Github website: https://github.com/Thenas/Control-ir-mqtt
80. Verba, N., Chao, K.-M., James, A., Goldsmith, D., Fei, X., & Stan, S.-D. (2017). Platform as a service gateway for the Fog of Things. Advanced Engineering Informatics, 33, 243–257. https://doi.org/10.1016/j.aei.2016.11.003.
81. Villari, M., M. Fazio, S. Dustdar, O. Rana, y R. Ranjan. «Osmotic Computing: A New Paradigm for Edge/Cloud Integration». IEEE Cloud Computing 3, n.o 6 (noviembre de 2016): 76-83. https://doi.org/10.1109/MCC.2016.124
82. W3C. (2004). SOAP 1.2 Attachment Feature. https://www.w3.org/TR/2004/NOTE-soap12-af-20040608/
83. Wilson, C., Hargreaves, T., & Hauxwell-Baldwin, R. (2017). Benefits and risks of smart home technologies. Energy Policy, 103, 72–83. https://doi.org/10.1016/j.enpol.2016.12.047

84. Xu, Li Da, Wu He, y Shancang Li. «Internet of Things in Industries: A Survey». IEEE Transactions on Industrial Informatics 10, n.o 4 (noviembre de 2014): 2233-43. https://doi.org/10.1109/TII.2014.2300753
85. Yi, S., Li, C., & Li, Q. (2015). A Survey of Fog Computing: Concepts, Applications and Issues. 37–42. https://doi.org/10.1145/2757384.2757397
86. Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., … Jue, J. P. (2019). All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey. Journal of Systems Architecture, https://doi.org/10.1016/j.sysarc.2019.02.009
87. Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7–18. https://doi.org/10.1007/s13174-010-0007-6
88. Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21. https://doi.org/10.1109/TNNLS.2018.2876865