Universidad de Oviedo

Máster en Ingeniería de Automatización e Informática Industrial

MAIIND

Mejora de la eficiencia computacional de técnicas machine learning para la detección temprana de defectos superficiales en piezas sometidas a procesos de fundición y estampado de chapa

Autor:
Jaime Philip Mon Bianco
[redacted]
[redacted]*@uniovi.es*

Tutor:
Ignacio Díaz Blanco
[redacted]*@uniovi.es*

Co-Tutor:
Diego García Pérez
[redacted]*@uniovi.es*

8 de febrero de 2022

Índice general

Índice de figuras:

Índice de tablas:

AGRADECIMIENTOS:

Me gustaría agradecer en primer lugar a Ignacio Díaz Blanco, mi tutor académico, por haberme brindado la oportunidad de desarrollar las prácticas en conjunto con mi TFM. A Jorge Marina Juárez responsable y mi tutor de la empresa *Desarrollo de Soluciones Integrales Plus S.L.* (DSIplus) donde desarrollé el proyecto y las prácticas, por haber confiado en mí y darme disponibilidad plena de tiempo y recursos. A Diego García Pérez por haber tenido la paciencia de inculcarme los conceptos y las ideas detrás del rigor científico que ha de tener un trabajo de cualquier tipo, como por dirigirme hacia la elaboración y perfeccionamiento de este proyecto. A Daniel García Peña, por haber tenido la paciencia de enseñarme e introducirme más aún en el mundo *Machine Learning*, así como en proyectos reales además de ayudarme siempre que lo he necesitado, gracias de corazón. A mis padres, mis dos hermanos, a mi hermana y mis amigos, por apoyarme en las decisiones que he tomado tanto a título personal como profesional y estar ahí cuando por momentos, la cabeza no me daba para más, de nuevo, gracias de corazón.

Por último, quería agradecer a todos/as los/as profesores/as que forman parte del MAIIND y a mis compañeros/as de clase, para mí han sido muy enriquecedores estos dos años de máster y animaría a cualquier persona a cursarlo por los conocimientos y valores que se quieren enseñar y transmitir, de nuevo muchas gracias.

1. Introducción

En las últimas dos décadas, el entorno industrial ha experimentado una nueva revolución aupada por las nuevas tecnologías, en especial aquellas conocidas como habilitadores digitales, tales como el *internet de las cosas* (*Internet of Things*, IoT), los servicios de *computación en la nube* y, más recientemente, el *análisis inteligente de datos*. Esta revolución ha desembocado en un nuevo paradigma conocido como *Industria 4.0*, el cual tiene como pilar fundamental el aprovechamiento de los datos de los procesos industriales, no solo para el uso tradicional de control y automatización de procesos, sino también para extraer nuevo conocimiento de estos, con el objetivo de optimizar el proceso, garantizar la trazabilidad en tiempo real de los productos o detectar anomalías sin intervención humana.

El *Machine Learning* (ML), como una de las principales ramas de la *Inteligencia Artificial* (IA), ha despertado un enorme interés en los últimos años, debido a un aumento de la capacidad de computación, una mayor disponibilidad de grandes conjuntos de datos y la consolidación de técnicas de ML tan potentes como las *redes neuronales profundas* (RNN), capaces de resolver tareas tan complejas como la detección de objetos o el procesamiento natural del lenguaje, con un desempeño similar o incluso mejor que un humano. La precisión demostrada por los modelos ML, y en especial las RNN, radica en la extracción de patrones y características complejas de los datos, muy difíciles de obtener mediante mecanismos de computación tradicional, las cuales pueden ser aprovechadas para resolver problemas de clasificación y regresión, y de esta forma extraer conocimiento útil de los datos.

A pesar de que los modelos basados en ML fueron adoptados rápidamente por empresas tecnológicas punteras mostrando resultados extraordinarios en ámbitos de aplicación como la bioinformática o el procesamiento de texto e imagen, dentro del ámbito industrial aún es una tecnología incipiente que la comunidad científica está tratando de aplicar a problemas de corte industrial como la *supervisión de procesos*, el *control inteligente*, la *detección anomalías*, el *mantenimiento predictivo* o la mejora de la e*ficiencia energética*.

Motivados por el enorme potencial de estas técnicas y por la ventaja estratégica que la aplicación de modelos ML a problemas industriales pueda aportar en un futuro próximo, la empresa *Desarrollo Soluciones Integrales plus* (DSIplus), durante el año 2020-2021 comenzó a utilizar esta tecnología para la detección de defectos superficiales, con el objetivo principal de mejorar sustancialmente las técnicas de visión artificial tradicionales que originalmente usaban para la detección de anomalías superficiales. En el trabajo realizado a lo largo del año 2020-2021 [1], se demuestra el gran potencial de estas técnicas y la mejora en la precisión a la hora de detectar defectos. Aunque la tecnología se esté aplicando y ya sea funcional, existe todavía un gran margen de mejora en términos de precisión y eficiencia computacional.

La mejora de la eficiencia computacional, en particular, es un aspecto bastante importante tanto para DSIplus, como para cualquier empresa de carácter industrial, donde exista un ciclo productivo y se usen este tipo de técnicas. Se entiende como eficiencia computacional, el ajuste óptimo de las propiedades atribuidas a un proceso de carácter secuencial para desempeñarlo de la manera más eficiente posible en términos de tiempo de computo y recursos necesarios. Para DSIplus, las propiedades que tienen alta prioridad

son: la latencia, el precio, la precisión del modelo ML y el tamaño del hardware en el que se va a integrar dicha mejora, debido a que este debe ser lo más compacto posible.

A lo largo del siguiente documento se estudiarán y aplicarán algunas de las técnicas más conocidas, para mejorar la eficiencia computacional y disminuir la latencia de los modelos de detección de defectos previamente desarrollados.

La estructura del proyecto será la siguiente:

- En el capítulo 2 de este documento, se expondrán las diferentes técnicas aplicadas, así como las metodologías utilizadas.

- En el capítulo 3 se especificarán los pasos para preparar los ensayos a realizar.

- En el capítulo 4 se expondrán las métricas utilizadas para evaluar las distintas técnicas utilizadas.

- En el capítulo 5 se explicarán los ensayos a realizar.

- En el capítulo 6 se exponen los resultados obtenidos para las distintas técnicas utilizadas.

- En la sección 7, se desarrollará una discusión de los resultados obtenidos y el planteamiento de nuevas líneas de desarrollo para futuros proyectos dentro de la mejora de la eficiencia computacional de modelos ML.

1.1. Motivación y antecedentes

En la actualidad, los controles de calidad en la industria son cada vez más estrictos e importantes, donde se demanda un control de la producción en su totalidad, en aspectos de carácter dimensional y en la detección de defectos tanto superficiales, como internos en los productos. Ante esta situación y con el objetivo de mejorar al máximo los procesos productivos, además de disminuir los costes de fabricación, es necesario el desarrollo de nuevos sistemas que tengan la capacidad de poder llevar a cabo el control de calidad con un alto grado de precisión y sin poner en riesgo los tiempos de producción.

En la última década, los controles de calidad se realizaban de forma estadística, mediante la toma de muestras aleatorias de la producción y llevando a cabo, sobre ellas, las pruebas y ensayos necesarios para certificar como buena o mala dicha muestra. Este enfoque sigue existiendo principalmente en las metodologías de inspección donde el contacto con la muestra a la hora de tomar las medidas es un requisito, además de ser necesario realizar estas de forma relativamente lenta, para evitar golpes que puedan dañar la muestra. Si a todos estos factores le añadimos las vibraciones que de forma muy frecuente se encuentran en los procesos industriales, resulta aún más difícil realizar un control fiable y sin ralentizar el ciclo de producción.

Con la aparición de los métodos de medida sin contacto, tales como *ultrasonidos* [2], *visión artificial* [3], *interferometrías* [4], las limitaciones de velocidad a la hora de realizar la medición han desaparecido y, por lo tanto, ya es posible realizar una inspección de la producción al completo. De todas formas, estas tecnologías y métodos de medición sin contacto están en fase de investigación y desarrollo en algunos campos de aplicación, ya que deben cumplir todos los requisitos que la industria impone para su correcta implementación en procesos industriales.

En esta situación, DSIPlus ha focalizado su trabajo en el desarrollo de sistemas de medición sin contacto basados en visión artificial. Aunque estos sistemas sean capaces de detectar defectos superficiales en piezas con geometrías complejas con gran precisión, es complicado diferenciar los defectos reales de los conocidos como *Falsos Positivos* (FP). La forma de los propios defectos hace que sean complejos de detectar con técnicas de visión por computador clásicas ya que, al utilizar sólo las distancias calculadas por triangulación o medidas derivadas calculadas a partir de estas, es muy difícil establecer manualmente una serie de reglas fijas que determinen la totalidad de los defectos. Hace un año y medio, apostaron por el ML como la tecnología para solventar este problema complementándola con métodos de visión por computador clásicos desarrollados para la segmentación de imágenes. Tras comprobar que es una tecnología que está teniendo buenos resultados, aún quedan aspectos que desarrollar para poder convertir los primeros prototipos en sistemas funcionales en entornos reales. En concreto, los aspectos más relevantes para el despliegue de estas técnicas en entornos reales son: *la latencia* [5] y *la precisión* [6] del modelo. Lograr reducir la latencia y que la precisión se mantenga dentro de unos márgenes aceptables, analizando las posibles soluciones existentes para realizarlo y determinando tanto sus pros como sus contras, es el principal objetivo de este trabajo.

1.2. Estado del arte

La *Inteligencia Artificial* (IA) ya es un término muy conocido e integrado en el vocabulario ordinario de la sociedad, debido a la gran relevancia y el efecto que esta tecnología está teniendo en una gran parte de las actividades rutinarias de las personas. La ya denominada cuarta *Revolución Industrial* (RI) [7], viene acompañada por esta tecnología que hace ya más de 70 años se empezó a esbozar tras la segunda guerra mundial con el *"Test de Turing*" [8]. Debido a la falta de capacidad de computación, de disponibilidad de datos y la necesidad de resolver problemas cada vez más complejos que las técnicas convencionales no pueden resolver, hasta hace tan solo un par de décadas no ha sido posible extraer su potencial en ámbitos de aplicación reales, tales como la industria [9], salud [10], economía [11], alimentación [12], logística [13], entre otros [14]. En la industria existe un abanico de ámbitos donde ha demostrado ser aplicable, como puede ser el *mantenimiento predictivo* [15], técnicas de *visión artificial* [16], *control de calidad* [17], *optimización de recursos* [18], *detección de anomalías* [19]. Las técnicas de visión artificial han demostrado ser aptas para el control de calidad en procesos de producción. En este ámbito las RNN densas han funcionado para la clasificación de texturas [20], clasificación de imágenes multiespectrales [21], clasificación de imágenes satelitales [22] y otras aplicaciones. Pero gracias a la aparición y aplicación de *redes neuronales convolucionales* (CNN) [23], orientadas al procesamiento de imágenes, se está pudiendo atajar este problema de forma más rápida y con mayor precisión. Más en concreto, la *segmentación de imágenes* [24] es uno de los paradigmas generales en el campo de la visión artificial, consistiendo en dividir la imagen en regiones. Ya existen arquitecturas que atajan este problema como la *UNET* [25], *FastFCN* [26], *Gated-SCNN* [27] y otras que son capaces de identificar partes de una imagen incluso en tiempo real como la *YOLO* [28]. Partiendo del contexto en el que se encuentra el ML y de las posibilidades que presenta en el ámbito industrial, la empresa DSIplus ha desarrollado una aplicación de segmentación de imágenes para la detección de defectos superficiales en piezas de fundición, basadas en RNN densas.

Uno de los parámetros más importantes a tener en cuenta dentro de la implementación de los modelos ML en cadenas de producción industrial, es que estos no introduzcan retrasos en el proceso. Estos modelos suponen una gran carga computacional, y al ser desplegados en equipos convencionales, los cuales en muchas ocasiones no son los apropiados para estos modelos, es posible que estos no logren cubrir dicha carga. En la búsqueda de esta mejora en el ámbito del ML, aparecen diferentes técnicas para lograrlo: 1) la reducción en la resolución de los parámetros de los modelos *Quantization* [29]; 2) la supresión de partes de un modelo que reduzcan su tamaño pero que no reduzcan su rendimiento *Prunning* [30]; y 3) mediante técnicas de *Knowledge Destilation* [31] que sintetizan modelos inicialmente complejos (gran cantidad de parámetros), donde los parámetros a eliminar son elegidos mediante técnicas ML; 4) el uso de hardware más potente y específico.

Dentro de este último enfoque, se puede mejorar el hardware en el entorno real mediante el uso de una *tarjeta gráfica* (GPU) [32] ya que estas son capaces de realizar muchas operaciones en paralelo de forma muy rápida. Estas tarjetas gráficas logran abarcar las operaciones que necesitan los modelos ML, pero a su vez, tienen un coste económico y energético alto. La empresa NVIDIA [33] han desarrollado una gama de sistemas embebidos llamados *Jetson Nano* [34], que son más compactos, tienen un precio

y un consumo energético menor que una GPU tradicional, pero son capaces de desempeñar cálculos con la misma latencia con un cierto límite.

La empresa Google [35] tiene una plataforma para acelerar el entrenamiento, evaluación y despliegue de modelos IA, tales como redes neuronales profundas, llamada Coral [36]. Dentro de esta gama de productos, Google ha equipado a todos sus equipos con *Unidades de Procesamiento Tensorial* (TPUs) [37], las cuales se caracterizan por ser capaces de soportar un mayor volumen de cálculo. Google diferencia dos tipos de TPU: *Edge TPU* [38] y *Cloud TPU* [39], donde los dispositivos Edge están pensados para ejecutar los modelos de redes neuronales en fase de inferencia y los Cloud son dispositivos en la nube, los cuales Google pone a disposición de los usuarios para el entrenamiento de modelos complejos dentro de su plataforma *Google Colab* [40]. En el año 2016 presentaron la primera generación de estos dispositivos que eran capaces de ejecutar modelos ML cubriendo la demanda que esto exigen, pero con menor precisión con la que lo haría un *procesador* (CPU) o GPU tradicional. En el año 2018 presentan su tercera generación y declaran que esta es 8 veces más potente que la generación anterior. Existen ejemplos de distintas aplicaciones reales utilizando estos dispositivos, sobre todo para modelos de visión [41].

Otras empresas como Intel [42] desarrolladora de CPUs, también ha presentado una gama de productos dedicados a la IA, Movidius Stick [43] que consiste en una *Unidad de Procesamiento de Visión* [VPU] [44] desarrollada para entornos IA, automatización industrial, seguridad y comercio minorista.

1.3. Objetivos

Basándose en el conocimiento adquirido en proyectos anteriores y en la problemática expuesta, se proponen los siguientes objetivos:

1. Explorar el impacto en el tiempo de inferencia y la precisión que conlleva la reducción de la resolución de los pesos que componen las redes neuronales utilizadas como clasificadores. Estudiar las diferentes estrategias que ayuden a definir a qué pesos se aplica dicha reducción y la magnitud de la misma.

2. Explorar el uso de elementos hardware que aceleren el cómputo de las redes neuronales propuestas y medir la mejora aportada en términos de tiempo de inferencia y precisión. En concreto, se propone el uso de sistemas embebidos específicamente diseñados para la aceleración de modelos basados en redes neuronales.

3. Reducción de la complejidad de los modelos mediante la reducción de la cantidad de pesos utilizados para la clasificación. En este apartado, se explorará tanto la eliminación experimental de pesos, mediante procesos de prueba y error, como mecanismos que reduzcan la cantidad de pesos en base a información extraída de la propia red.

4. La evaluación de las técnicas estudiadas, comparando los modelos de partida con los modelos modificados en términos de tiempo de inferencia y perdida de precisión. Para ello se realizarán experimentos cuantitativos donde se medirán la perdida de precisión por los métodos mediante métricas conocidas (recall, precisión, F1-score, etc.) y la mejora de los tiempos de inferencia en el entorno a desplegar los modelos, así como evaluaciones cualitativas de los enfoques estudiados, analizando visualmente en qué defectos o partes de la pieza se materializan las pérdidas de precisión.

2. Métodos y técnicas

La integración de sistemas que utilizan ML, tanto en la industria, como en elementos de nuestro día a día como el teléfono móvil o el coche, hace que surja la necesidad de integrar estos modelos en hardware cada vez más pequeño, rápido y con la mayor eficiencia energética posible. Estos modelos han de llevar a cabo miles, cientos de miles o millones de operaciones, lo que hace, en la mayoría de los casos, que sean computacionalmente lentos.

2.1. Explicación de modelos ML

Para tener una noción básica de en qué consisten estas operaciones y por qué pueden suponer una gran carga computacional, se tomará como ejemplo una RNN densa sencilla y se irán describiendo los pasos seguir para realizar los cálculos: se parte de una RNN de *L* capas con *j* neuronas en cada capa, *i* neuronas en la capa de entrada y con una neurona en la salida. En la Figura 1, se muestra el grafo correspondiente a dicha arquitectura.

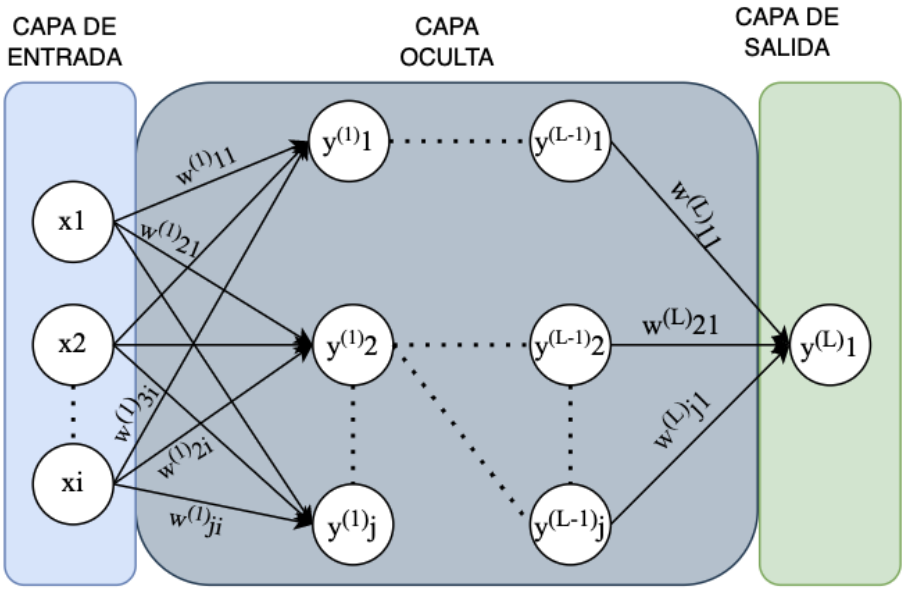


Figura 1. Grafo de una red neuronal

La operación que se realiza es una multiplicación de matrices, de manera que los pesos (las $w^{(L)}_{ji}$ de la imagen anterior) se multiplican por todos los valores de entrada de cada capa, es decir, si se quiere sacar la salida, por ejemplo, de la primera capa de la capa oculta (las $y^{(L)}_{j}$, llamadas activaciones) la operación en forma de matrices (1) y en forma de ecuaciones (2) se muestran a continuación:

$$\begin{bmatrix} \boldsymbol{y_1^{(1)}} \\ \boldsymbol{y_2^{(1)}} \\ \vdots \\ \boldsymbol{y_j^{(1)}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w^1{}_{11}} & \boldsymbol{w^1{}_{12}} & \dots & \boldsymbol{w^1{}_{1i}} \\ \boldsymbol{w^1{}_{21}} & \boldsymbol{w^1{}_{22}} & \dots & \boldsymbol{w^1{}_{2i}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w^1{}_{j1}} & \boldsymbol{w^1{}_{j2}} & \dots & \boldsymbol{w^1{}_{ji}} \end{bmatrix} * \begin{bmatrix} \boldsymbol{x1} \\ \boldsymbol{x2} \\ \vdots \\ \boldsymbol{x_i} \end{bmatrix} + \boldsymbol{\beta} \qquad \textit{(1)}$$

$$\begin{aligned} \boldsymbol{y_1^{(1)}} &= \boldsymbol{w^1{}_{11} \cdot x1 + w^1{}_{12} \cdot x2 \dots + w^1{}_{1i} \cdot x_i + \beta_1} \\ \boldsymbol{y_2^{(1)}} &= \boldsymbol{w^1{}_{21} \cdot x1 + w^1{}_{22} \cdot x2 \dots + w^1{}_{2i} \cdot x_i + \beta_2} \\ &\vdots \\ \boldsymbol{y_j^{(1)}} &= \boldsymbol{w^1{}_{j1} \cdot x1 + w^1{}_{j2} \cdot x2 \dots + w^1{}_{ji} \cdot x_i + \beta_i} \end{aligned} \qquad \textit{(2)}$$

Donde las $\mathbf{y_j^{(L)}}$ serían las activaciones resultantes a la salida de la capa L, las ${w^L}_{ji}$ los pesos, las x_i las entradas a la capa y $\boldsymbol{\beta}$ el bias Atendiendo a esta ecuación (2), las operaciones para el cálculo de las activaciones de una capa son $i \cdot j$ más j sumas. Si suponemos $j = 3$ y $i = 4$, se tiene que el número total de operaciones son 16. Aunque esta cantidad de operaciones no supone para un ordenador de hoy en día un cálculo complejo, los modelos que se suelen utilizar en ML normalmente no constan de unos cuantos parámetros, sino de millones. En consecuencia, la capacidad de computación de esos ordenadores ha de ser importante para suplir dicha carga de trabajo sin comprometer el tiempo de predicción teniendo en cuenta que algunos procesos, especialmente los industriales, tienen especificaciones temporales muy estrictas. Está claro, que siempre se puede invertir en una GPU o CPU que sea capaz de cubrir la demanda de estos modelos, pero esto puede no interesar en grandes procesos industriales donde se tienen varias cadenas de producción o incluso no sea posible debido a que los equipos dispuestos en planta no tienen la compatibilidad con una GPU o CPU nueva. En definitiva, no es sencillo cambiar el hardware de todo un proceso, de modo que las técnicas que se han desarrollado para la mejora computacional de estos modelos no cambian el hardware directamente, sino que lo complementan o son compatibles con el hardware estándar.

La aparición de las técnicas y dispositivos dedicados a mejorar la eficiencia computacional de estos modelos ha permitido con la misma capacidad de cálculo o menor, poder manejar dichos modelos.

2.2. Modelos de detección de defectos superficiales

Tomando como punto de partida el trabajo desarrollado en la empresa DSIplus [1], se está trabajando con piezas de fundición de geometría compleja como se observa en la Figura 2. A la hora de detectar los defectos superficiales se llevaron a cabo dos enfoques de cara a dicha detección: 1) clasificando todos sus píxeles uno a uno; 2) mediante un análisis de toda la imagen buscando manchas. Para este proyecto se usará el primer enfoque, clasificando todos los píxeles como normal o defecto.

Figura 2. Pieza de fundición de geometría compleja

A raíz de esta decisión, se desarrollaron dos arquitecturas ML para la detección de defectos superficiales: RNN densas y CNN basadas en la UNET [25]. A continuación, se irán describiendo cuales son los pasos seguidos desde la medición de los sensores hasta la predicción de los modelos ML. La pieza que se muestra en la Figura 2, avanza por una cinta en la que están dispuestos 8 sensores de triangulación laser que miden la distancia. En la Figura 3, se muestra cómo es la disposición real de los sensores.

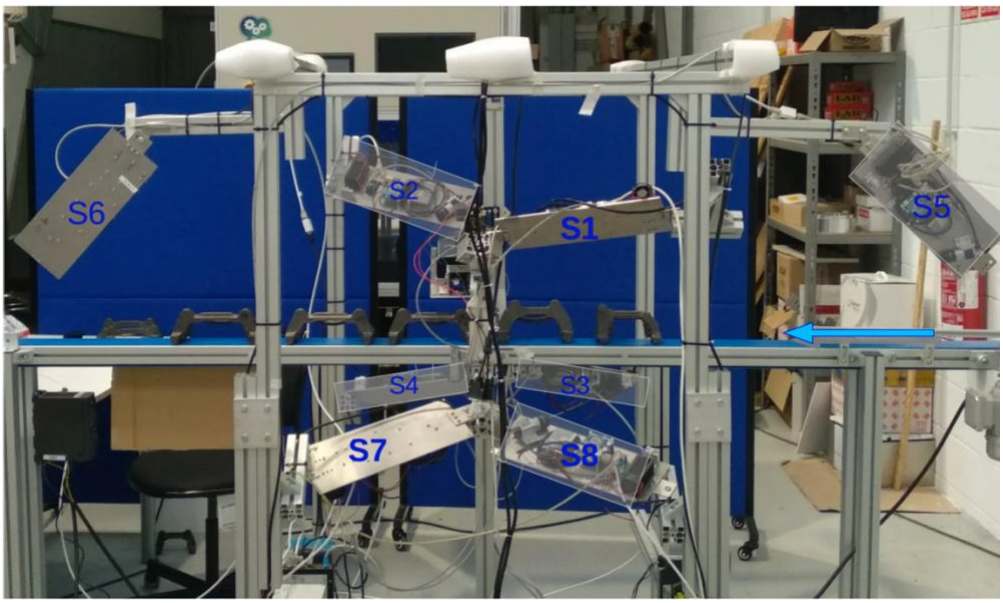


Figura 3. Sensores de medición en posición de trabajo (Fuente: [1])

Para simplificar el problema, se trabajará con las distancias de uno de los sensores nada más. Considerando como punto de partida una imagen de distancias del sensor escogido, en este caso el S1 de la Figura 3, tenemos que $\boldsymbol{X} \in \mathbb{R}^{lxn}$. De esa imagen de distancias, se escoge la zona de interés para ser procesada por el modelo y se deciden extraer de dicha zona m características como se muestra en la ecuación (3), mediante la operación de extracción de características $\boldsymbol{F}(\boldsymbol{X}^{(i)})$.

$$\boldsymbol{F}(\boldsymbol{X}^{(i)}) = \{\boldsymbol{F_1}^{(i)}, \boldsymbol{F_2}^{(i)}, \dots, \boldsymbol{F_m}^{(i)}\} \quad (3)$$

Se calculan un total de 15 características a partir de la imagen de distancias original. Como ya se mencionó anteriormente, se trabajó con dos arquitecturas distintas donde para cada una de ellas se han de introducir los datos en el modelo de manera distinta. Para las RNN densas, la entrada es un vector y para las CNN se trata de una

imagen. Para generar el vector de entrada del modelo RNN, de la característica F, se extrae una ventana de tamaño ws, donde $P(x_i, y_i)$ es el píxel del centro de la ventana y, esta ventana se convierte en un vector de tamaño $(ws \cdot ws, 1)$, tal y como se muestra en la Figura 4. Esto se replica para las m características calculadas cogiendo el mismo píxel P como centro de la ventana. Los m vectores extraídos de cada una de las imágenes F_i se concatenan todos los vectores en uno solo que conforman el vector de entrada de la RNN densa. En la Figura 5 se muestra cómo se realiza la extracción del vector para las m características y se unifica para ser introducida en el modelo ML. Para recorrer el tamaño de las imágenes de características se desplaza el píxel $P(x_i, y_i)$ de izquierda a derecha y desde arriba hacia abajo.

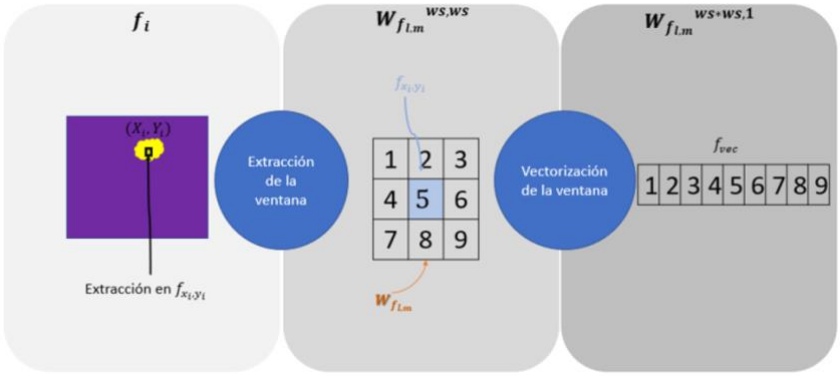


Figura 4. Vectorización en un punto $P(x_i, y_i)$ (Fuente: [1])

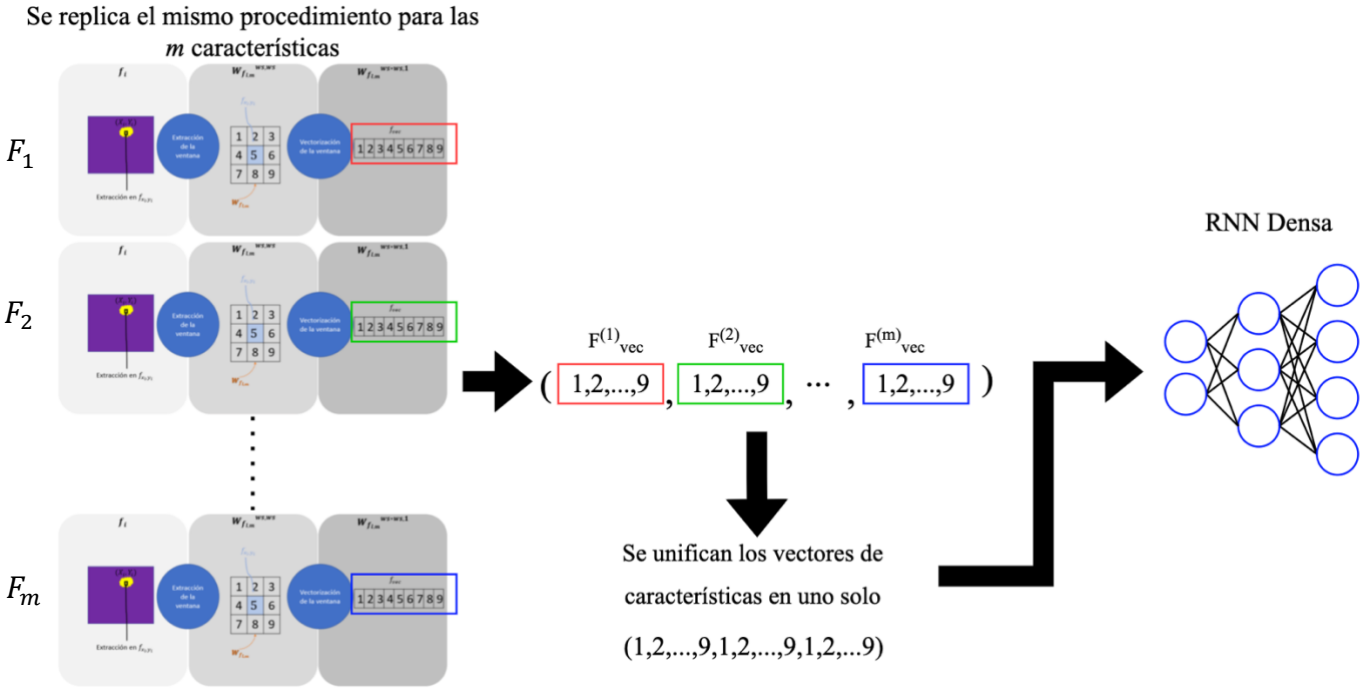


Figura 5. Generación del vector de características de entrada de una RNN densa

Para las RNN entonces, el tamaño de entrada al modelo es el correspondiente a $\boldsymbol{X} \in \mathbb{R}^{(ws \cdot ws \cdot m, 1)}$. Por otro lado, en los modelos basados en CNN el procedimiento es similar, la diferencia fundamental es que, en vez de ser un vector apilado de las características, es un tensor de imágenes de características. Cada una de las imágenes F_i se divide en una serie de ventanas del mismo tamaño que las dimensiones de entrada del modelo, tal y como se muestra en la Figura 6. Una vez definido el tamaño de la ventana

de entrada como ws, se sitúa el píxel origen de la ventana $P(x_i, y_i)$, que en este caso es el de la parte superior izquierda y se desplaza ws píxeles hacia la derecha y hacia abajo. Del mismo modo que con las RNN, se itera sobre todas las características con ese tamaño de ventana y se concatenan en un único tensor formado por imágenes de características que corresponden a la entrada del modelo.

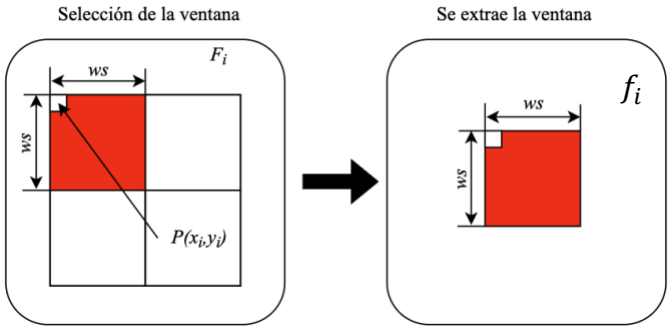


Figura 6. Extracción de una ventana la característica F_i

La Figura 7 muestra como se generaría este vector de imágenes de características que correspondería con la entrada al modelo CNN de la UNET.

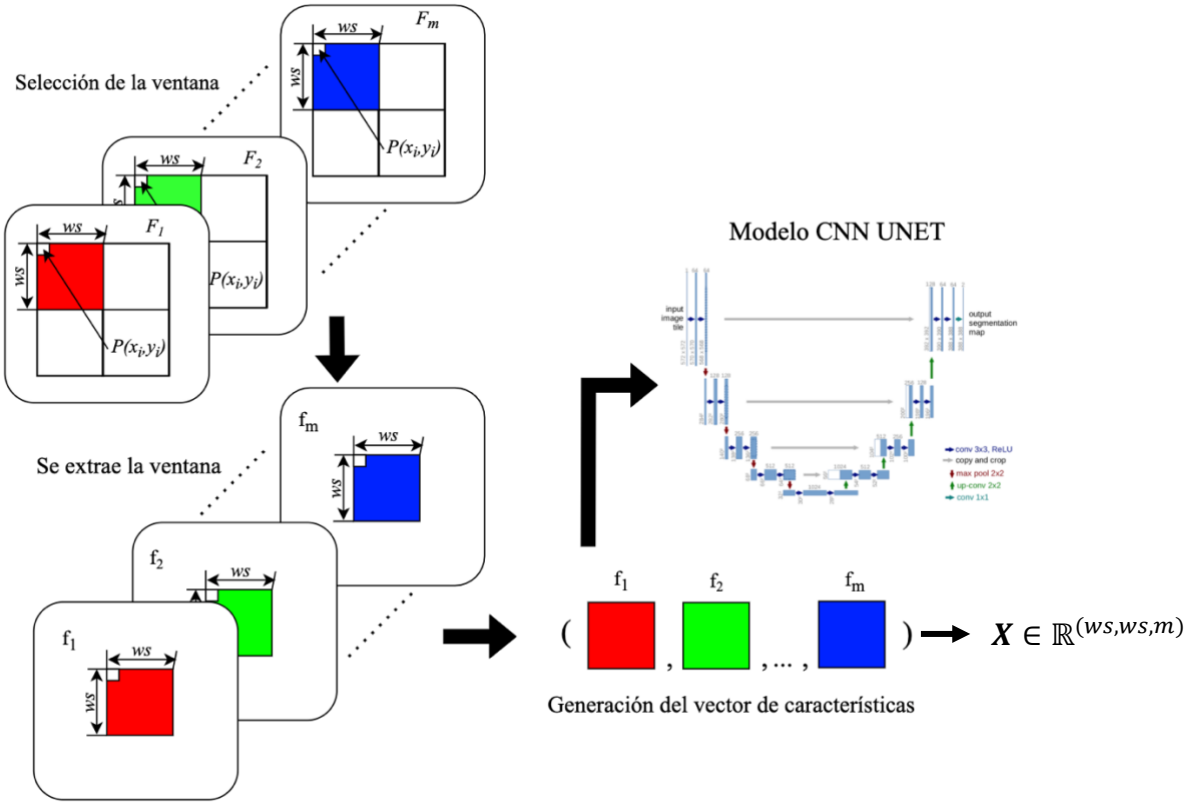


Figura 7. Generación del tensor de características de entrada del modelo CNN (Fuente foto arquitectura UNET: [25])

Para las CNN, el tamaño de entrada al modelo es un tensor de dimensiones (ws, ws, m).

Los datos con los que se trabajan en ambas arquitecturas son los mismos, lo que cambia es la forma de cargarlos en el modelo a la hora de ejecutar la inferencia. En la Figura 8, se pretende representar esto.

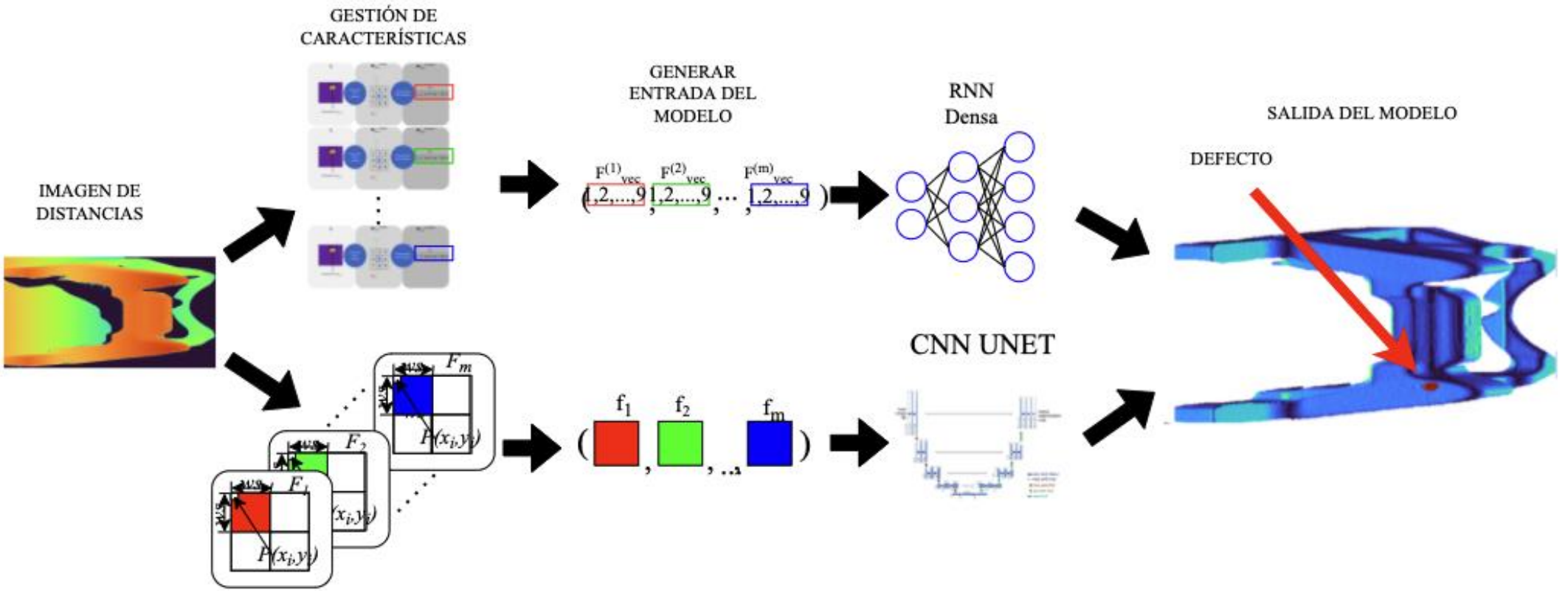


Figura 8. Flujograma modelo ML

2.3. Técnicas para la mejora de la eficiencia computacional

A lo largo de este apartado se propondrán las distintas técnicas utilizadas para lograr una mejora computacional de los modelos ML en términos de tiempo de inferencia y precisión, describiendo su funcionamiento y cómo es su aplicación sobre los propios modelos. Las técnicas a explorar son: la reducción de la resolución de los parámetros de los modelos, también conocida por la comunidad ML como *Quantization*, tareas de post-procesamiento y el uso de dispositivos del tipo Edge TPU.

2.3.1. Quantization

Comenzando con la *quantization* [45] que el término traducido al español es cuantificación, consiste en la reducción de la resolución de los parámetros con los que el modelo opera (p.ej. en vez de trabajar en FP64 trabajar en FP32), de manera que se logra una reducción de los tiempos de cálculo y, en consecuencia, un aumento de la velocidad del modelo en inferencia. Por otro lado, hay que tener en cuenta que esto puede suponer una reducción de la precisión de los modelos, por lo tanto, se pretende que esta reducción de la resolución se haga de manera que afecte lo menos posible.

Para hacerse una idea del cambio en la precisión de los valores de un tipo de dato a otro, se va a tomar como ejemplo la multiplicación de matrices (4), partiendo de una resolución FP32, la cual tiene una resolución aproximada de 7 decimales según el estándar para aritmética del IEEE 754 [46]:

$$\begin{pmatrix} \mathbf{-0.18120981} & \mathbf{-0.29043840} \\ \mathbf{0.49722983} & \mathbf{0.22141714} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{0.77412377} \\ \mathbf{0.492993965} \end{pmatrix} = \begin{pmatrix} \mathbf{-0.28346319} \\ \mathbf{0.49407474} \end{pmatrix} \textit{(4)}$$

El rango de valores en las matrices anteriores está entre (f_{min}, f_{max}), en este caso $(-1,1)$. Si se reduce su resolución a *enteros de 8 bits* (INT8) siendo (q_{min}, q_{max}) los límites de dicha resolución, cuyos valores son $(-128,127)$, se tiene que hacer una transformación al nuevo rango de valores. En la Figura 9 se muestra la idea de esta reducción de la resolución.

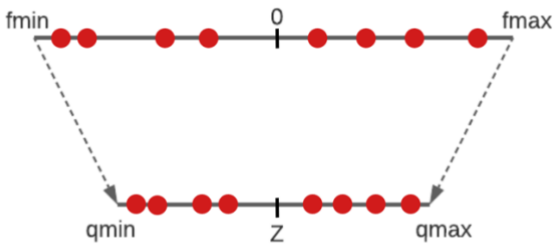


Figura 9. Representación gráfica de la reducción de la resolución (Fuente: [47])

Para ello, se utilizará (5) que es la ecuación general para el cambio de una resolución alta de los valores a una más baja [47].

$$\boldsymbol{r = S \cdot (q - Z)} \textit{ (5)}$$

Donde los valores con una resolución más baja (en este caso INT8), representado por q en la ecuación, se pueden calcular a partir del valor real original r en resolución alta (en este caso FP32), junto con el ajuste de los parámetros S la pendiente y Z el punto de corte con cero. Para el cálculo de la pendiente S, se sigue el mismo criterio que con una recta (6). Por otro lado, para el cálculo del paso por cero Z, se realiza también con el mismo criterio (7).

$$\boldsymbol{S = \frac{f_{max} - f_{min}}{q_{max} - q_{min}}} \textit{ (6)}$$

$$\boldsymbol{Z = q_{min} - \frac{f_{min}}{S}} \textit{ (7)}$$

Volviendo al ejemplo de la multiplicación de matrices (1), se convierten los valores a una resolución mas baja (8).

$$\begin{pmatrix} \mathbf{-24} & \mathbf{-38} \\ \mathbf{63} & \mathbf{28} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{99} \\ \mathbf{63} \end{pmatrix} = \begin{pmatrix} \mathbf{-4770} \\ \mathbf{8001} \end{pmatrix} \textit{ (8)}$$

La matriz resultante, se obtiene de multiplicar dos números con una resolución INT8, lo que equivale a una resolución de un *entero de 16 bits* (INT16). Si se deshace la reducción de la resolución y se vuelve a la resolución original de FP32, se puede observar la pérdida de precisión resultante (9) con respecto al resultado original (1):

$$\begin{pmatrix} \mathbf{4770} \\ \mathbf{8001} \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{-0.2911377} \\ \mathbf{0.48834229} \end{pmatrix} \textit{ (9)}$$

Una vez se comprende que supone una reducción en la resolución, se encuentra que, dentro de esta técnica, existen distintas formas de lograr esta reducción. A continuación, se describen las distintas variantes que se han utilizado:

1. *Quantization Pre-Training*: esta técnica, como su propio nombre indica, consistiría en predefinir el tipo de dato con el que se quiere trabajar antes de entrenarlo, de manera que, desde un inicio, todas las operaciones se realizan en el tipo de dato predefinido y los parámetros del modelo se ajustan con dicha resolución a lo largo de todas las etapas de generación del modelo. Para esta técnica se realizará una conversión de FP32 a FP16. Se estima que el tiempo durante la inferencia se vea reducido a la mitad al trabajar a la mitad de la resolución original. Cada técnica tiene su proceso de implementación y esta es la más directa de las planteadas, debido a que tan solo se tiene que definir el tipo de dato en el que se quiere trabajar a la hora de entrenar el modelo, en el caso de este proyecto FP16.

2. *Quantization Post-Training*: esta técnica es similar a la anterior, con la diferencia fundamental que, tras entrenar el modelo y generarlo, se realiza la conversión al nuevo tipo de dato. Además, se realiza un entrenamiento con el modelo convertido al nuevo tipo de dato de cara al ajuste de los parámetros del modelo a la nueva resolución, esto se conoce como *fine-tuning* [48]. Para esta técnica se hará una conversión de FP32 a FP16. Del mismo modo que la técnica anterior, se estima que el tiempo durante la inferencia se vea reducido a la mitad. Por otro lado, para esta técnica, la conversión, a diferencia de la técnica anterior, se realiza tras el entrenamiento de manera que los parámetros del modelo se ajustan en alta resolución y luego se realiza la conversión. Otra diferencia fundamental con respecto a la técnica *Quantization Pre-Training*, es que se realiza un *fine-tuning* de los parámetros tras realizar la conversión. Esto consiste

en reentrenar el modelo convertido con una parte de los datos, de este modo se pretende lograr el ajuste óptimo de los parámetros en baja resolución de manera que tengan el comportamiento más parecido a trabajar en la resolución original.

3. *Quantization Aware-Training*: esta técnica es algo diferente a las dos mencionadas anteriormente. La conversión del tipo de dato del modelo se realiza después de entrenarlo, pero durante el entrenamiento se limita el rango de los valores con los que se opera a la resolución deseada para simular el entrenamiento en dicha resolución. Además, se recoge información acerca de la distribución de los pesos, registrando los valores mínimo y máximo de estos en cada una de las capas, de forma que, a la hora de hacer la conversión al tipo de dato deseado, la precisión del modelo no se vea tan afectada. Para esta técnica se hará una conversión de FP32 a INT8. A diferencia de las anteriores, la reducción de la resolución para esta técnica es mucho mayor, en consecuencia, se espera poder conseguir en la inferencia que tarde 5 veces menos que de manera original. Por último, para esta técnica se llevan a cabo distintos pasos. La conversión a baja resolución se realiza tras el entrenamiento, pero durante el mismo, se hace una emulación de lo equivalente a trabajar en baja resolución, en este caso INT8. En la Figura 10 se muestra cómo es la implementación interna de esta técnica durante el entrenamiento.

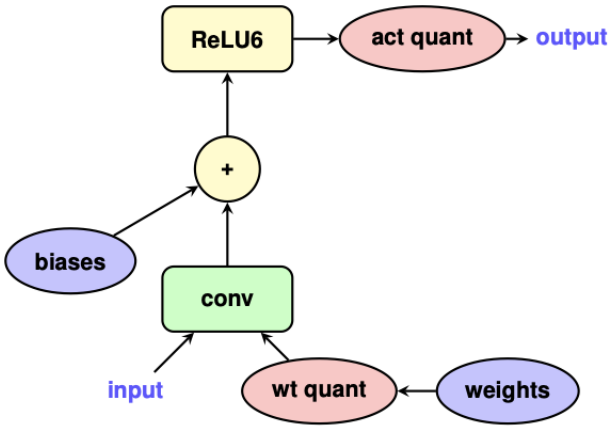


Figura 10. Implementación de la técnica Quantization Aware-Training (Fuente: [49])

Todas las operaciones realizadas durante el entrenamiento se hacen con una resolución de FP32, los nodos *"wt quant"* y *"act quant"* se insertan para efectuar la emulación mencionada. Esta se realiza acotando el rango de valores en FP32 a los de una resolución de INT8, es decir, los valores se limitan a trabajar solo en el rango de INT8 $(-128,127)$ pero siguen siendo FP32. Realizar esta emulación durante el entrenamiento supone que los pesos y las activaciones se ajusten en el rango de resolución baja, de manera que cuando se cambie a la resolución final, este cambio tenga el menor impacto posible sobre la precisión del modelo. Esta técnica se aplicará junto con la Edge TPU debido a que actualmente no hay disponibilidad de otro hardware donde se realizan operaciones matriciales de forma óptima.

Para la inferencia, se tiene una representación similar, donde directamente se trabaja en baja resolución como se puede observar en la Figura 11.

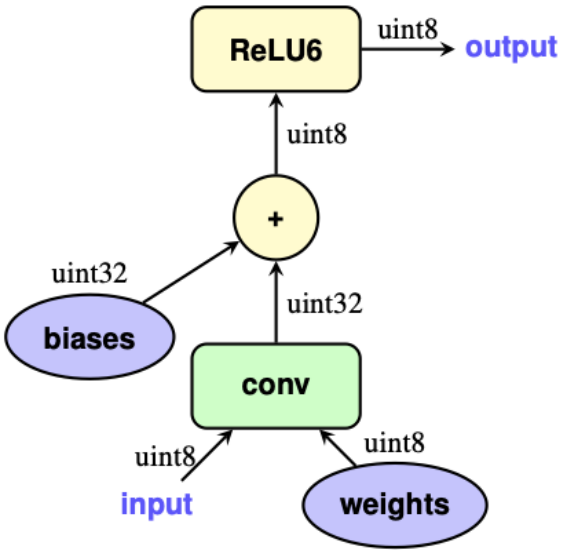


Figura 11. Modelo en baja resolución durante la inferencia (Fuente: [49])

Como se puede observar en la figura anterior, tras realizar la convolución entre la entrada y los pesos, ambos siendo UINT8, aumenta la resolución de la activación a la salida a UINT32 pero, la suma entre la activación y el bias vuelve a reducirse a la resolución original UINT8, esto se debe a cómo se realizan estas operaciones internamente. Si se vuelve a la fórmula (5), se supone que se quieren multiplicar dos valores r_1 y r_2 que resulta en un valor r_3, si se aplica dicha fórmula se obtiene (10).

$$\boldsymbol{r_3 = r_1 \cdot r_2 \rightarrow S_3 \cdot (q_3 - Z_3) = S_1 \cdot (q_1 - Z_1) \cdot S_2 \cdot (q_2 - Z_2)} \quad \textit{(10)}$$

El valor que se quiere obtener finalmente es q_3 que sería el resultado de la operación tras haber realizado la cuantización. Si se despeja dicho valor se obtiene (11).

$$\boldsymbol{q_3 = Z_3 + M \cdot (q_1 - Z_1) \cdot (q_2 - Z_2)} \quad \textit{(11)}$$

$$\boldsymbol{M = \frac{S_1 \cdot S_2}{S_3}} \quad \textit{(12)}$$

Como se puede observar, el factor $\boldsymbol{M}$ junta las S de cada operación de cuantización y, en consecuencia, este provoca el ajuste real a la resolución deseada, en este caso, UINT8.

En la Figura 12 se muestra en forma de diagrama las etapas que se siguen al aplicar esta técnica, con las diferencias entre cada una de ellas y aplicándose la conversión correspondiente para este proyecto.

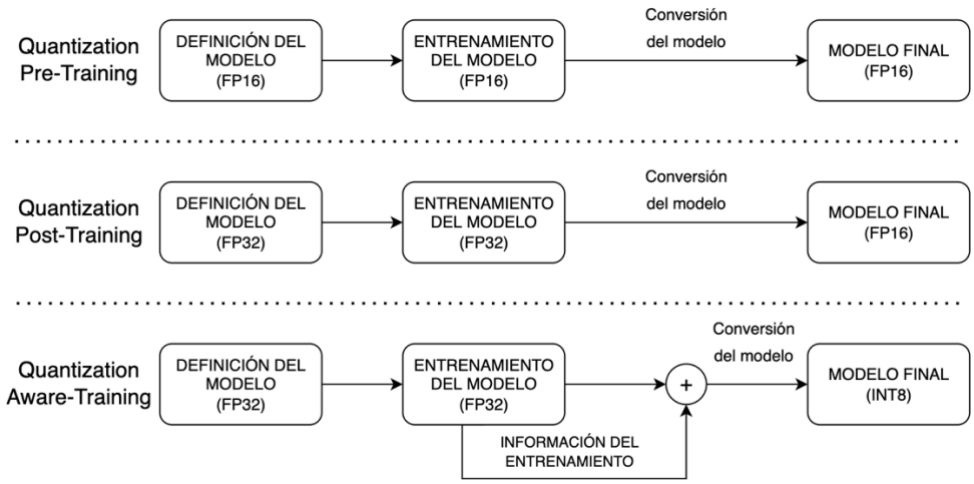


Figura 12. Diagrama comparativo entre tipos de Quantization

2.3.2. Reducción de los píxeles a procesar

Esta técnica parte de la base de que el modelo de predicción en sí no se modifica, lo que se modifica es el número de píxeles a procesar por el modelo, es decir, reducir la resolución de las imágenes de entrada. A diferencia de otros modelos ML basados en CNN donde la entrada al modelo es una imagen, por ejemplo, la *MobileNetV1* [50] o la *ResNet-50* [51], la entrada al modelo en este caso, al tratarse de una RNN densa, es un vector de características como ya se explico en el apartado 2.2, no imágenes en crudo. La idea fundamental detrás de esta técnica es que, al procesar menos puntos, implica directamente una menor carga computacional y, en consecuencia, un aumento de la velocidad del modelo a la hora de predecir la imagen completa. Inicialmente, se plantearon dos alternativas: 1) Quitar píxeles a procesar de la muestra de entrada al modelo generando una máscara de forma aleatoria; 2) Quitar píxeles a procesar de la muestra de entrada al modelo generando la máscara basándose en un patrón.

Como ya se explicó en el apartado 2.2, se tiene la máscara que contiene los píxeles a procesar por el modelo, y es sobre esa máscara donde se aplica el patrón, obteniendo una nueva máscara con los puntos a procesar con menor cantidad que los originalmente seleccionados. A partir de la región resultante, como se explicó en el apartado 2.2, se extraen las características y se prepara la entrada del modelo. Se ejecuta la inferencia y, sobre la salida del modelo, se realizan tareas de post-procesamiento, que expanden la resolución de la imagen de salida a la original.

Las tareas de post-procesamiento consisten en un suavizado de la imagen, realizándolo en dos pasos: 1) rellenar los puntos que, en la máscara inicialmente establecida, tras haber aplicado el patrón habían sido descartados, por el valor máximo de la salida del modelo; 2) se aplica una *erosión* [52] sobre la imagen resultante y posteriormente se aplica un *filtro de mediana* [53] para suavizarla. En la Figura 13 se muestra el flujo de etapas que tiene esta técnica.

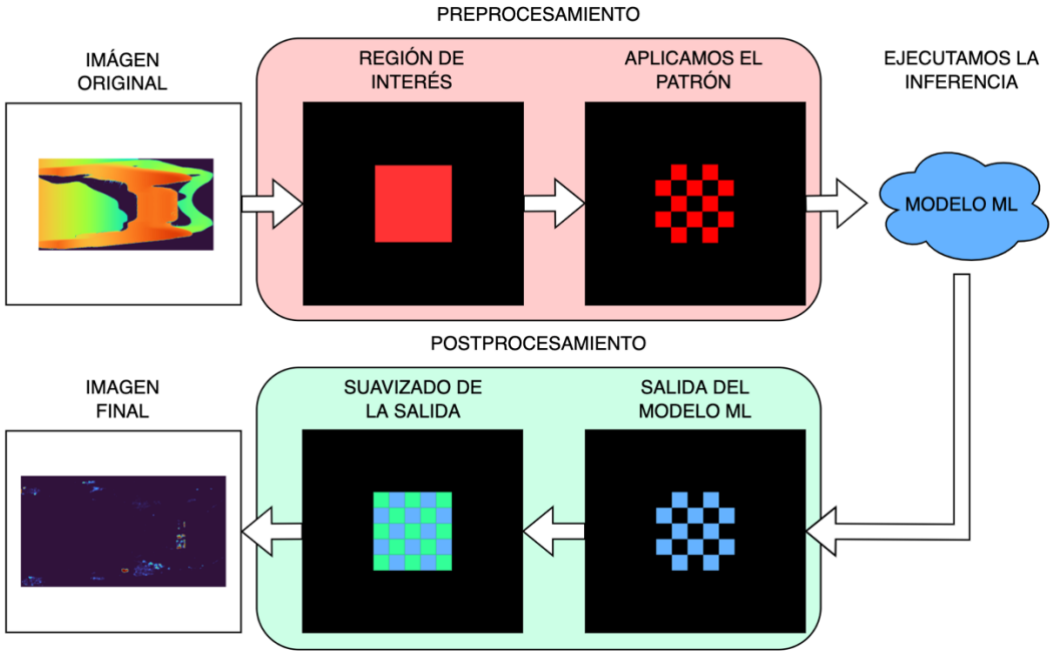


Figura 13. Flujo de las etapas de la técnica reducción de los píxeles a procesar

Con esta técnica se pretende reducir hasta un 80% el número de puntos a procesar por el modelo y que ello, se estima que supondrá una reducción temporal de al menos el 50%.

2.3.3. Edge TPU

Las unidades de procesamiento tensorial, o TPUs según sus siglas en inglés, han demostrado ser dispositivos compactos y potentes con relación al tamaño que tienen, energéticamente muy eficientes y con un coste muy competitivo en el mercado. Como ya se mencionó en el apartado 1.2, empresas como Google o Intel han desarrollado dispositivos propios específicamente diseñados para la ejecución de modelos ML: *Google Coral* o *Intel Movidius*. Para este proyecto se decidió usar la Edge TPU de Google, en concreto, el *USB Accelerator* [54] debido a la disponibilidad, fácil conectividad (USB) y el precio.

Como ya se mencionó en el apartado 2.3.1, reducir la resolución de los parámetros con los que opera el modelo supone un aumento en la velocidad a la hora de realizar la inferencia. La Edge TPU en concreto, trabaja con *enteros de 8 bits* (INT8), y es aquí donde se aplica la técnica mencionada en el apartado 2.3.1 *Quantization Aware-Training*, lo que supone una reducción drástica de la resolución. Otra característica bastante importante de este dispositivo es que al ser externo y por cómo está diseñado, las tareas que pueda estar ejecutando el ordenador al que está conectado, no le afectan a la hora de realizar las operaciones, en consecuencia, la Edge TPU se ocupa de una tarea muy concreta en el despliegue de un modelo ML.

Las CPU y GPU al igual que las TPU, son unidades de procesamiento, con la diferencia fundamental, que la TPU es un *Application-Specific Integrated Circuit* (ASIC) [55], es decir, está diseñada para hacer un tipo de operación en concreto. Para la TPU esta operación es de multiplicación y suma que son las frecuentemente usadas en las RNN, tal y como se expuso en el apartado 2.1. Las CPUs realizan las operaciones leyendo cada entrada y el peso correspondiente del modelo desde memoria, las multiplica usando la

unidad aritmética lógica (ALU) [56], almacena el resultado en memoria y finalmente realiza el sumatorio. Las CPUs actuales tienen un espacio de caché grande, utilizan *predictores de saltos* [57] y tienen relojes de alta frecuencia, con lo cual, todo contribuye a una baja latencia a la hora de ejecutar los modelos con la CPU. En la Figura 14 se muestra como es la operación realizada por una CPU:

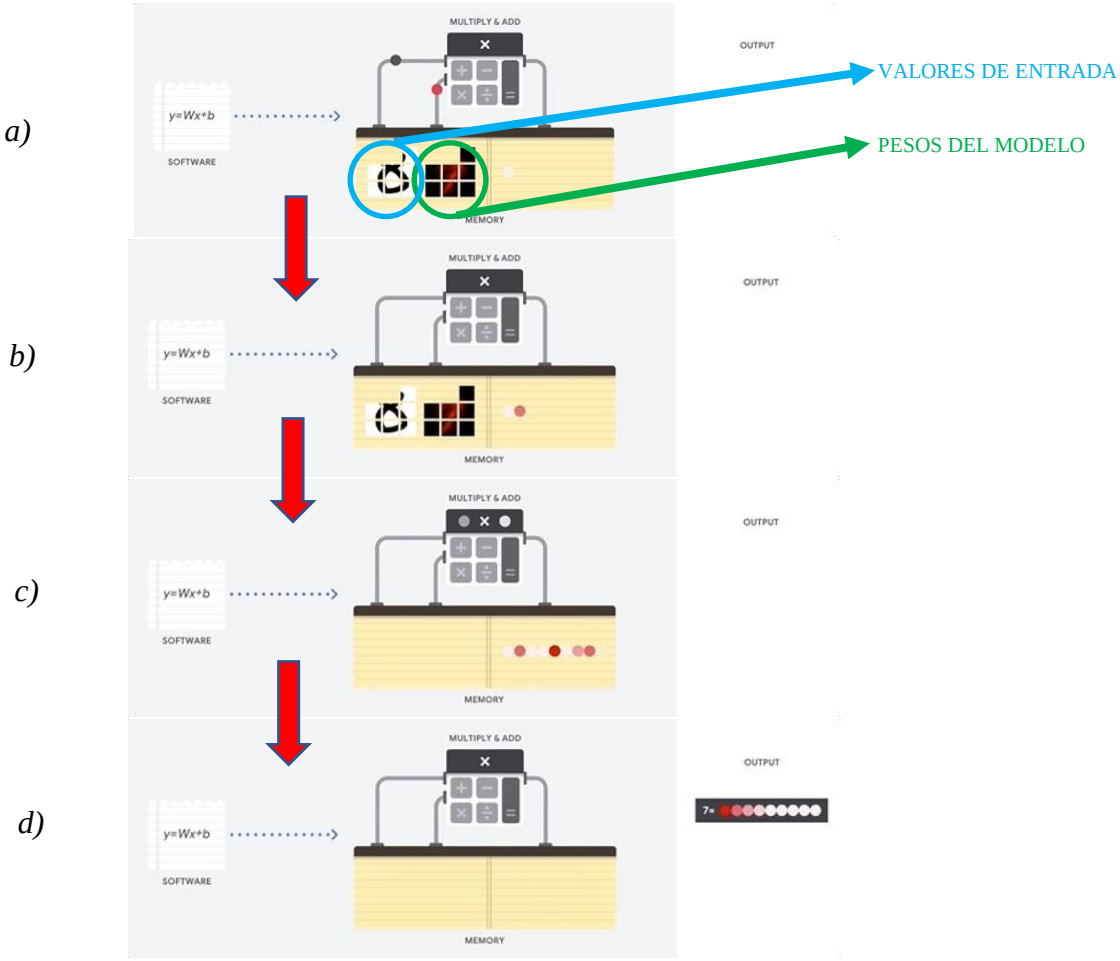


Figura 14. Descripción visual de cómo realiza las operaciones una CPU (Figura basada en la fuente [58])

Las GPU están basadas en el mismo concepto que las CPUs, pero estas tienen miles de ALUs realizando los cálculos, además de que estos se pueden paralelizar. Esto se conoce como *Single instruction, multiple data* SIMD [59] y el ejemplo asociado a este tipo de operaciones son la multiplicación y suma en las RNN. Las GPUs mejoran el rendimiento paralelizando los cálculos. Su principal desventaja es el precio (si se tienen varias líneas de producción quizás no interese comprar varias GPU) y en algunos casos computacionales.

La TPU, por otro lado, opera de manera distinta, las ALUs están conectadas entre sí directamente formando una matriz interconectada de ALUs, sin tener la necesidad de acceder a la memoria para obtener el resultado de la operación anterior.

Por otro lado, los pesos se cargan previamente en los ALUs de la TPU como se representa en Figura 15, lo que supone no acceder a la memoria para cargar los pesos en cada operación.

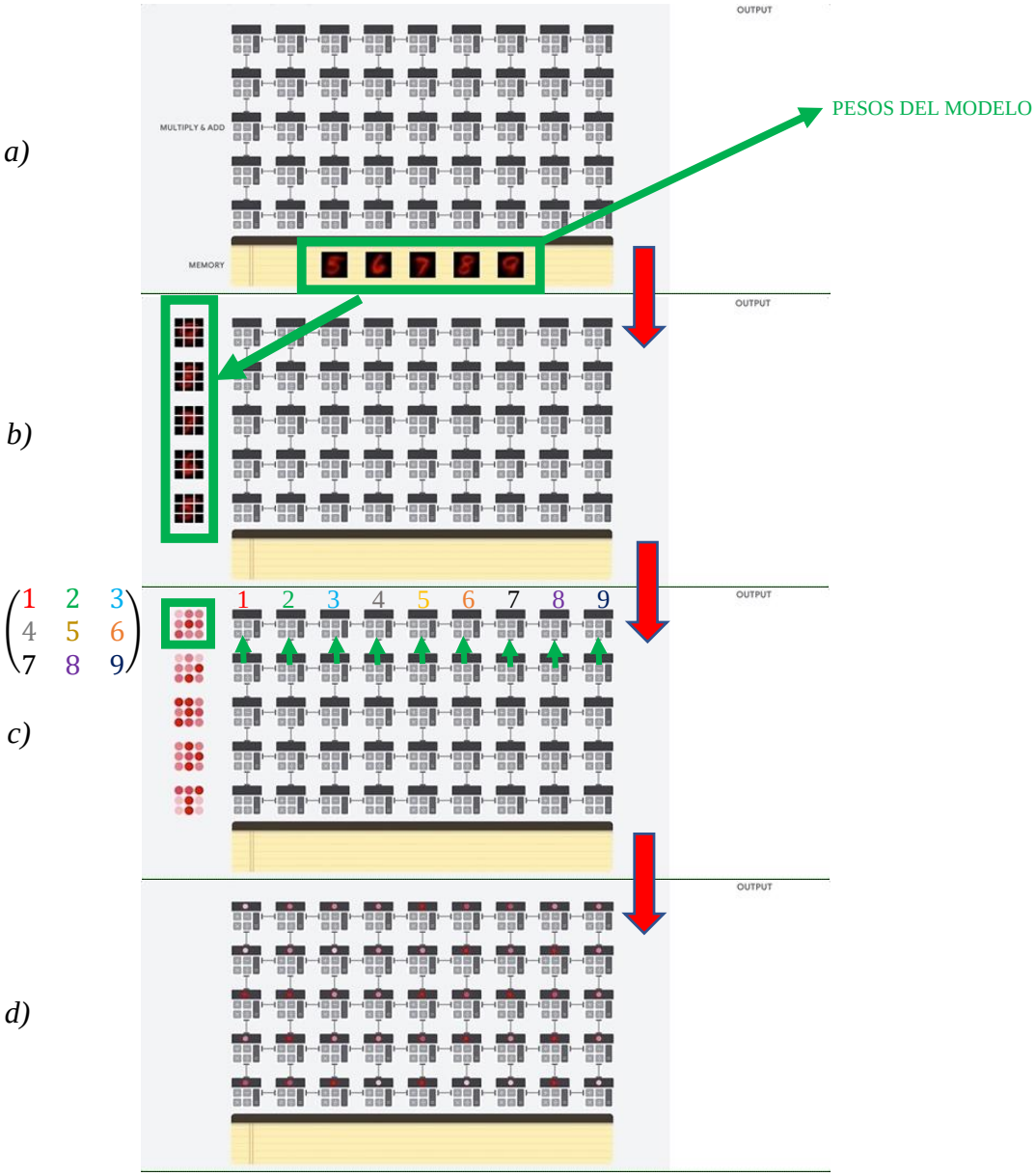


Figura 15. Carga de pesos en las ALUs de la TPU (Figura basada en la fuente [58])

Esto supone una disminución en la latencia de las operaciones. Este es un aspecto clave de este dispositivo y se conoce como Systolic Array (SA) [60]. Esta metodología, permite encadenar operaciones de forma paralela y secuencial. Las entradas se multiplican por el peso del ALU que le corresponde, y el resultado de esta multiplicación se suma al ALU al que está conectado, de manera que una operación se concatena con la siguiente.

En la Figura 16, se puede observar cómo las entradas se insertan en vertical de forma secuencial y el resultado de la operación se transfiere al ALU de la derecha, es decir, los cálculos se hacen de manera paralela y bidireccional.

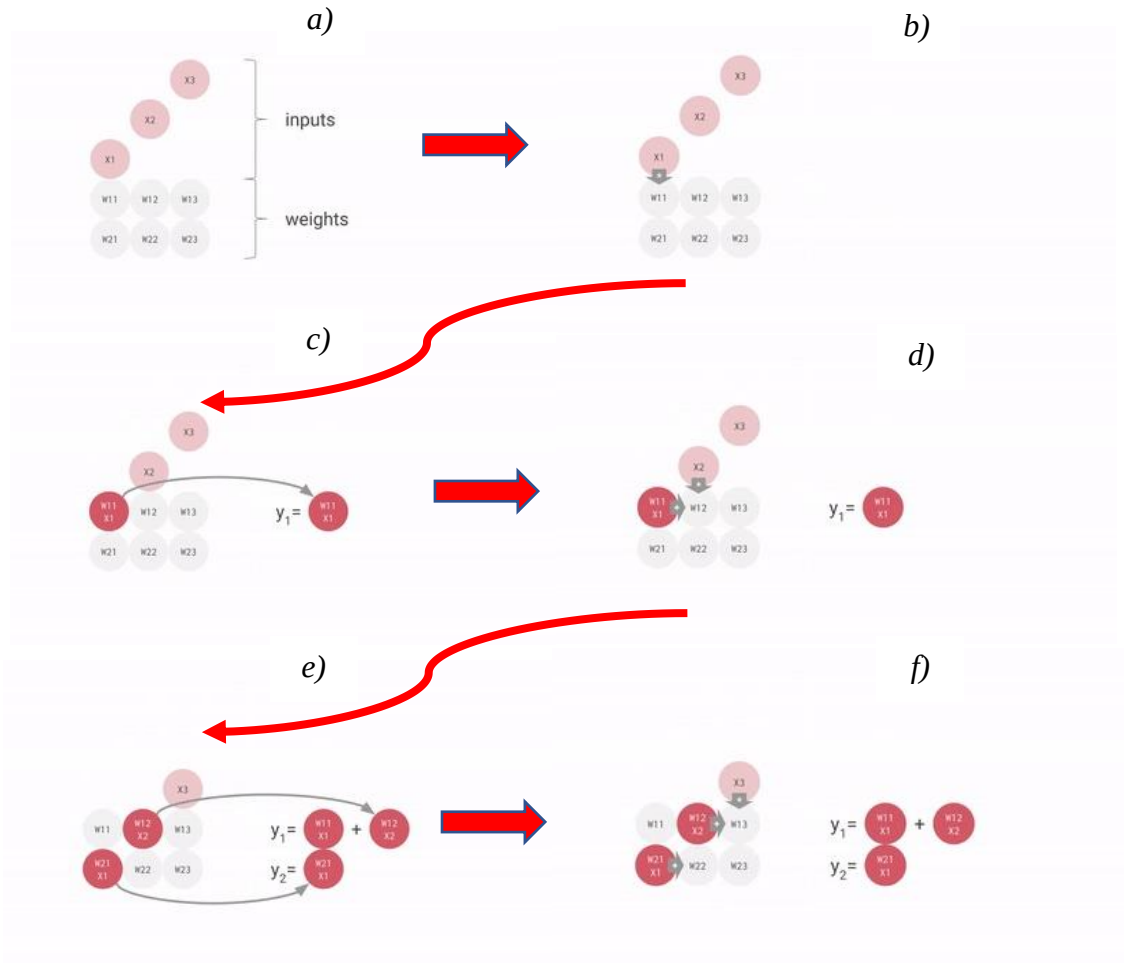
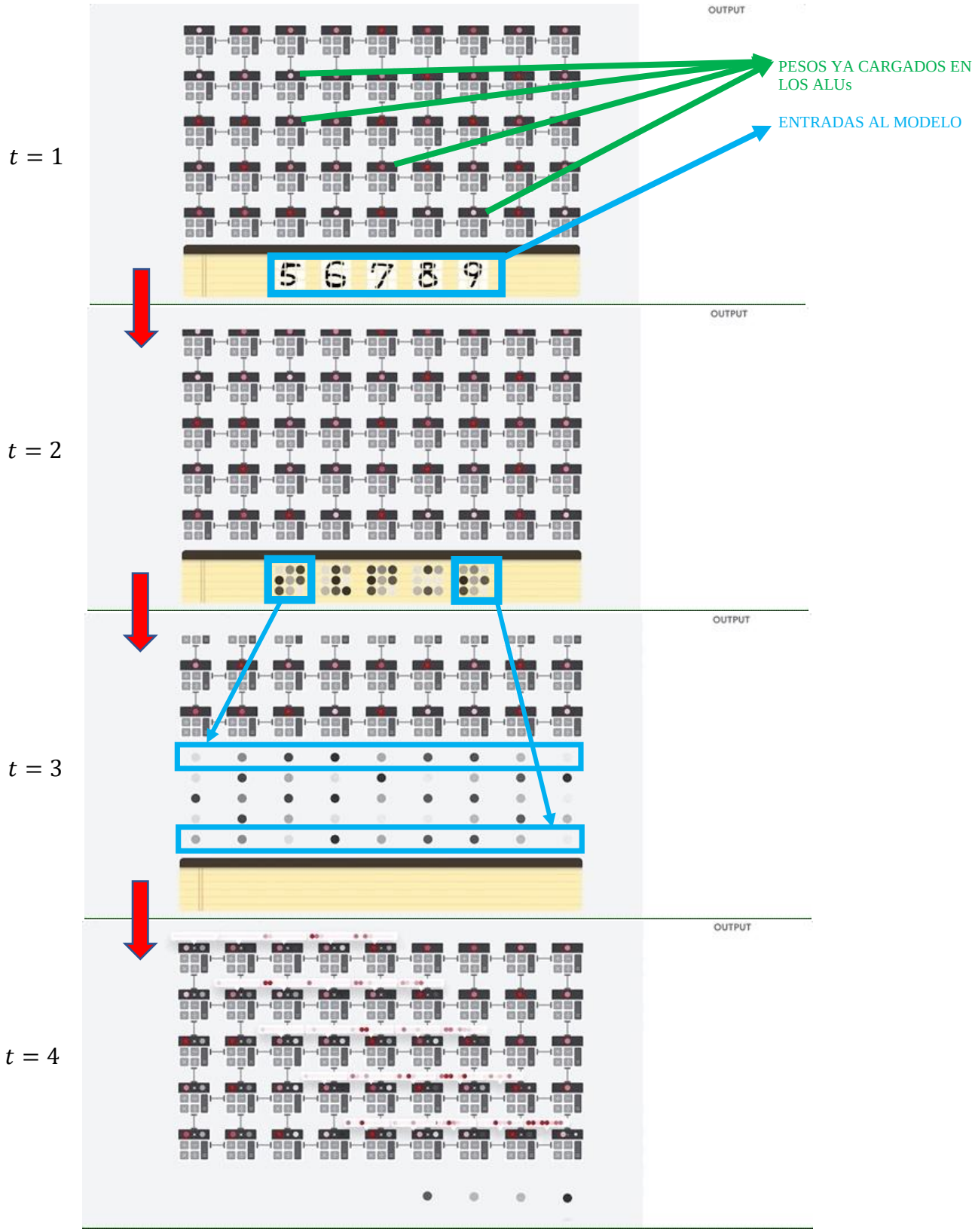


Figura 16. Funcionamiento del Systolic array
(Figura basada en la fuente [58])

En la Figura 17 se trata de representar estas operaciones aplicándose en la ejecución de un modelo ML.



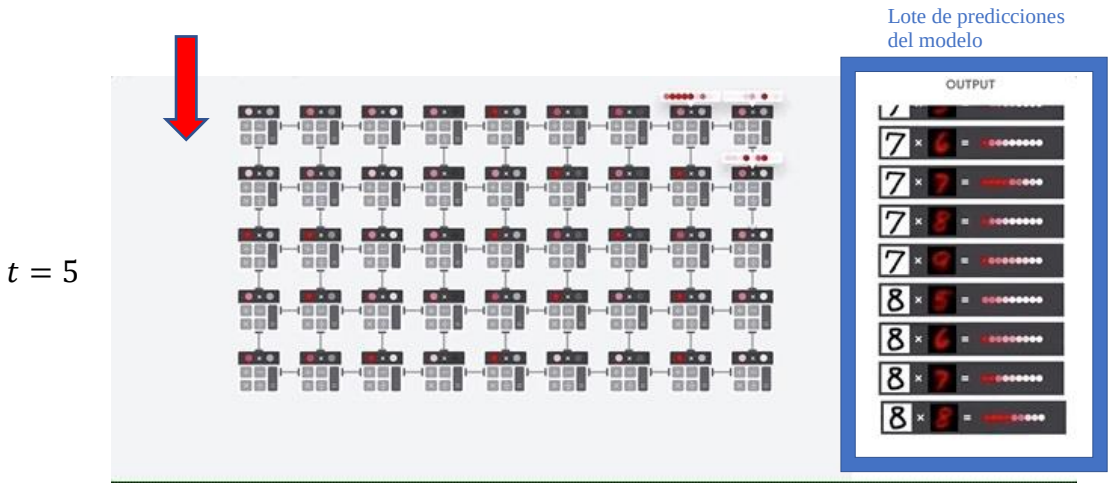


Figura 17. Systolic array aplicado a un modelo ML
(Figura basada en la fuente [58])

Otra característica importante de esta metodología es que estas operaciones se realizan sin necesidad de acceder a la memoria, de manera que esto aumenta con creces la capacidad y la velocidad de cálculo de las TPU.

3. Preparación de ensayos

3.1. Hardware utilizado

Para lograr completar las distintas etapas que forman del desarrollo de los modelos ML, se necesita un hardware para lograr llevarlo a cabo. Para el desarrollo de este proyecto, se han usado 3 equipos distintos especificados en las Tablas 1, 2 y 3 mostradas a continuación:

NOMBRE DE EQUIPO	MacBook Pro (Retina 13 pulgadas, 2015)
PROCESADOR	2.7 GHz Intel Core i5 de doble núcleo
TARJETA GRÁFICA	Intel Iris Graphics 6100 1536 MB
MEMORIA RAM	8 GB 1867 MHz DDR3
SISTEMA OPERATIVO	macOS Big Sur

Tabla 1: Especificaciones del MacBook Pro

NOMBRE DE EQUIPO	Servidor de entrenamiento
PROCESADOR	3.4 GHz Intel Core i7, 8 núcleos
TARJETA GRÁFICA	Nvidia GeForce GTX 1060 6GB
MEMORIA RAM	64 GB
SISTEMA OPERATIVO	Debian 10.3

Tabla 2. Especificaciones del servidor de entrenamiento

NOMBRE DE EQUIPO	Google Coral USB Accelerator
PROCESADOR	Google Edge TPU coprocessor
TARJETA GRÁFICA	-
MEMORIA RAM	8MB
SISTEMA OPERATIVO	-

Tabla 3. Especificaciones del USB Accelerator

La variedad en el hardware se debe a la utilidad que tiene cada uno en las distintas etapas de elaboración de un modelo ML: entrenamiento, evaluación y despliegue ya mencionadas en el apartado 2.1. Los modelos ML se pueden entrenar y desplegar utilizando una CPU, pero la etapa que conlleva una mayor carga de trabajo es el entrenamiento, es por ello, que el servidor al llevar una GPU le lleva menos tiempo realizar esta tarea. La Edge TPU, se ha usado exclusivamente para el despliegue de los modelos y la CPU se ha utilizado para el despliegue y algunas pruebas realizadas.

3.2. Software y framework

El lenguaje de programación más utilizado por la comunidad ML y para el desarrollo de este proyecto es Python. Este lenguaje, ha demostrado a lo largo de la última década ser un lenguaje adecuado para el desarrollo e implementación de modelos ML debido a la facilidad para la programación, el tratamiento de datos e implementación.

Para realizar todas las pruebas y ensayos en ML se suele utilizar un framework que, por lo general, suele facilitar el trabajo a la hora de desarrollar e implementar el

código. Existen distintos frameworks que están dedicados al ML, a continuación, se citan los más conocidos:

1. TensorFlow [61]
2. PyTorch [62]
3. scikit-learn [63]
4. Spark ML [64]

Para este proyecto se utilizó el framework de *TensorFlow* por las razones listadas a continuación:

1. TensorFlow es el framework utilizado en la empresa DSIPlus con la que se ha desarrollado el proyecto.
2. Ofrece herramientas de cara a la implementación de distintas técnicas existentes para la mejora computacional de los modelos ML.
3. La implementación y aplicación de las utilidades que ofrece suele ser sencilla debido a la cantidad de información y ejemplos End-to-End que ofrece en la página web y documentación asociada.

TensorFlow, tiene una librería llamada *TensorFlow Lite* [65] que fue desarrollada para desplegar modelos ML en sistemas móviles y embebidos. Esta es la misma librería que se usa para convertir algunos de los modelos y que sean desplegables en dispositivos como la Edge TPU. Además, TensorFlow, tiene desarrolladas e implementadas algunas de las técnicas mencionadas relacionadas con la reducción de la resolución. En la Figura 18, se muestra un diagrama generalizado de las etapas de desarrollo de los modelos ML para este proyecto, diferenciando los entornos de ejecución y el Hardware utilizado para cada una de ellas.

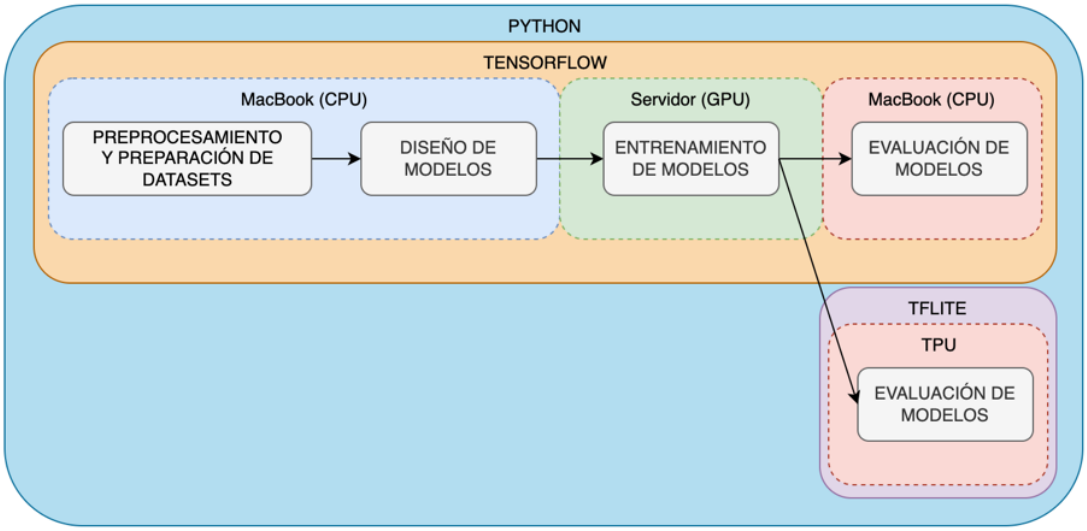


Figura 18. Diagrama de las etapas de desarrollo de un modelo ML para este proyecto

3.3. Recomendaciones previas

Con el fin de que los ensayos se puedan comparar bajo las mismas condiciones, se estableció que se debía usar siempre el mismo equipo y que este estuviera corriendo los mismos programas, o al menos, las tareas que se estuvieran ejecutando en el momento de realizar los ensayos fueran las mismas o muy similares. Monitorizar dichas tareas es importante en los ensayos relacionados con modelos ML, debido a la gran carga computacional que suponen las operaciones que se realizan, y cualquier alteración en los recursos utilizados mientras estos se están ejecutando, pueden suponer un resultado no válido en términos de latencia. A continuación, se listan una serie de recomendaciones a seguir en la elaboración de un ensayo:

- Finalizar las *tareas innecesarias* del ordenador.

- Si se realizan las pruebas en un ordenador portátil (como es en este caso), hay que asegurarse de que esté *enchufado a la alimentación para obtener el mayor rendimiento*, ya que el ordenador por defecto intentará reducir los recursos utilizados para ahorrar batería.

- Separar los ensayos ya sean con distintos modelos o distintas configuraciones, en archivos separados, es decir, *tener el código en scripts separados*, de manera que estos se tengan que ejecutar por separado y no interfieran unos con otros.

- *Ejecutar los programas un número fijo de veces* y corroborar que los números sean consistentes en los resultados, sobre todo en términos de latencia.

- Para realizar los ensayos con la *Edge TPU de Coral* y obtener los mejores resultados, se debe enchufar a un ordenador con la entrada del tipo USB 3.0 para obtener la mayor velocidad de transmisión de datos posible.

Para este proyecto, además de seguir estas recomendaciones, para evitar problemas de carga en la caché del equipo se decidió que cuando se realizara un experimento, se ejecutarían las operaciones concretas de inferencia 100 veces y luego se calcularía la media del tiempo que han tardado las ejecuciones, ya que se detectó que la primera ejecución del modelo es siempre la más lenta al tener que cargar el propio modelo en la primera iteración.

4. Métricas

En este apartado se expondrán todas las métricas utilizadas para los ensayos realizados.

4.1. Métricas de precisión

Para medir la precisión de las distintas técnicas aplicadas, se utilizaron 4 métricas explicadas a continuación. A lo largo de este proyecto, se decidió utilizar la medida de precisión conocida por la comunidad ML como *F-Score* [66] o *F1-Score*, que mide la precisión que tiene un modelo y está definida, como la *media armónica* [67] de la *precisión* y la *sensibilidad* [68].Se calculó la diferencia porcentual entre la precisión del modelo original y de una de las técnicas aplicadas, para ello se utilizó la fórmula (7).

$$\%\boldsymbol{Diferencia\ Precisión\ Absoluta} = \mathbf{100} * \left(\mathbf{1} - \frac{\boldsymbol{Precisión\ modelo\ con\ técnica}}{\boldsymbol{Precisión\ modelo\ sin\ técnica}}\right) \textbf{(7)}$$

Por otro lado, se decidió utilizar el residuo generado, siendo este calculado como la diferencia entre la imagen de salida del modelo y la imagen etiquetada, esto se calcula siguiendo la fórmula (8), donde $\widehat{\mathbf{Y}}_{\mathbf{i,j}}$ es la predicción del modelo y $\mathbf{Y}_{\mathbf{i,j}}$ es la imagen etiquetada.

$$\boldsymbol{Residuo\ a\ nivel\ de\ pixel} = \sum_{i,j}\left|\widehat{\boldsymbol{Y}}_{i,j} - \boldsymbol{Y}_{i,j}\right| \textbf{(8)}$$

Para analizar como es la mejora frente al planteamiento original, se calculó la diferencia porcentual neta entre los residuos como se especifica en la fórmula (9).

$$\%\boldsymbol{Diferencia\ Porcentual\ Residuo} = \mathbf{100} * \left(\frac{\boldsymbol{Residuo\ Modelo\ Nuevo - Residuo\ Modelo\ Original}}{\boldsymbol{Residuo\ Modelo\ Original}}\right) \textbf{(9)}$$

Por último, para la evaluación de los modelos basados en CNN, se utilizó la métrica *Coeficiente de Sorensen-Dice* [69], que mide la similitud entre dos imágenes siguiendo la fórmula (10), donde $\widehat{\boldsymbol{Y}}$ es la predicción del modelo e $\boldsymbol{Y}$ es la imagen etiquetada.

$$\boldsymbol{Dice\ Coefficient} = \mathbf{2} \cdot \frac{|\hat{Y} \cap \boldsymbol{Y}|}{|\hat{Y}| + |\boldsymbol{Y}|} \textbf{(10)}$$

4.2. Métricas temporales

De cara a la medición de la ganancia temporal entre el modelo original y una de las técnicas aplicadas o entre las técnicas entre sí, se utilizó la fórmula (11).

$$\%\boldsymbol{Diferencia\ Porcentual\ Tiempo} = \mathbf{100} * \left(\frac{\boldsymbol{Tiempo\ Modelo\ Nuevo - Tiempo\ Modelo\ Original}}{\boldsymbol{Tiempo\ Modelo\ Original}}\right) \textbf{(11)}$$

5. Ensayos a realizar

Teniendo en cuenta las técnicas del apartado 2.3.1, que se van a utilizar en este proyecto, a continuación, se listan los experimentos que se van a realizar junto con el hardware utilizado.

5.1. Quantization

En este primer bloque de experimentos, se compararán los tiempos de inferencia y la precisión de modelos con una arquitectura de la red basada en RNN aplicando la técnica *Quantization*. Se escogieron varios modelos con distinto número de parámetros de cara a analizar cómo afecta el tamaño del propio modelo en los tiempos de la inferencia. Para esta primera técnica se realizará el cambio de la resolución de los modelos de FP32 (estándar que usa TensorFlow para sus modelos) a FP16. Debido a que la arquitectura hardware de las CPU de Intel, que son el tipo de CPU que están disponibles para este proyecto, no están optimizados para utilizar el tipo de dato FP16, para realizar esta serie de experimentos se utilizará el servidor de entrenamiento, que está equipado con una GPU que soporta el tipo de dato mencionado. Para esta técnica se utilizará un tamaño de lote fijo de 2048, que corresponde al número de muestras que se procesan durante la inferencia.

Los experimentos están listados a continuación:

1. Generar un modelo base en el tipo de dato FP32 y ejecutar la inferencia almacenando el tiempo que ha tardado en efectuarla.

2. Utilizar la técnica *Quantization Post-Training*, entrenando el modelo y posteriormente convertirlo de FP32 a FP16 y ejecutar la inferencia almacenando el tiempo que ha tardado en efectuarla.

3. Utilizar la técnica *Quantization Pre-Training*, entrenando el modelo directamente en el tipo de dato FP16 y ejecutar la inferencia almacenando el tiempo que ha tardado en efectuarla.

5.2. Reducción de los píxeles a procesar

En este segundo bloque de experimentos, se compararán los tiempos de inferencia y la precisión de modelos con una arquitectura basada en RNN aplicando como técnica tareas de post-procesamiento. De cara a la realización de estos experimentos, no es necesario hacer uso de una GPU, ya que es una tarea que no afecta a la propia ejecución del modelo y que esta se puede realizar bajo cualquier CPU. Como ya se mencionó en el apartado 2.3.2, la idea detrás de esta técnica se basa en quitar puntos a procesar por el modelo para reducir el número de operaciones a realizar y, en consecuencia, reducir el tiempo de inferencia. Los experimentos para esta técnica se plantearon de manera que la variable a modificar sería el número de puntos a quitar con respecto al total de la imagen, es decir, modificar el porcentaje de puntos procesados. Esto se aplica a las dos alternativas planteadas. En este caso, el tamaño del lote es variable debido a que depende del tamaño de la imagen. A continuación, se listan los experimentos a realizar:

1. Se ejecuta la inferencia procesando la imagen completa y se almacena el tiempo que se ha tardado en procesarla para tenerlo como referencia sobre la que compararse.

2. Utilizar la primera alternativa para quitar un porcentaje de los puntos de la imagen original generando una máscara de forma aleatoria, modificando el valor del porcentaje de puntos eliminados entre 10-90% de los puntos.

3. Utilizar la segunda alternativa para quitar un porcentaje de los puntos de la imagen original generando una máscara basándose en distintos patrones, de forma que, al aplicar ese patrón, este equivale a quitar un porcentaje de puntos eliminados entre 10-90% de los puntos.

5.3. Edge TPU

En este tercero y último bloque de experimentos, se compararán los tiempos de inferencia y la precisión de modelos con una arquitectura basada en RNN y en CNN . De cara a la realización de estos experimentos se utilizará la *Coral USB Accelerator* y el MacBook Pro. Como ya se mencionó en el apartado 2.3.3, utilizar un Edge TPU para ejecutar la inferencia implica no solo cambiar la forma en la que el propio dispositivo realiza las operaciones, también supone una combinación con la primera técnica mencionada, quantization, ya que los modelos que se ejecutan en dicho dispositivo deben tener una resolución INT8.

Por otro lado, la TPU trabaja con una librería específica de TensorFlow, como ya se mencionó en el apartado 2.4.2 llamada TFLite. Este caso es algo particular, ya que después de generar el modelo, entrenarlo y convertirlo al formato de TFLite, este se debe de compilar con la librería de Coral [70], para luego poder cargarse en la TPU y ejecutar la inferencia. Para entender esto con mayor detalle, la Figura 19 representa los pasos a seguir para lograr ejecutar la inferencia en este dispositivo.

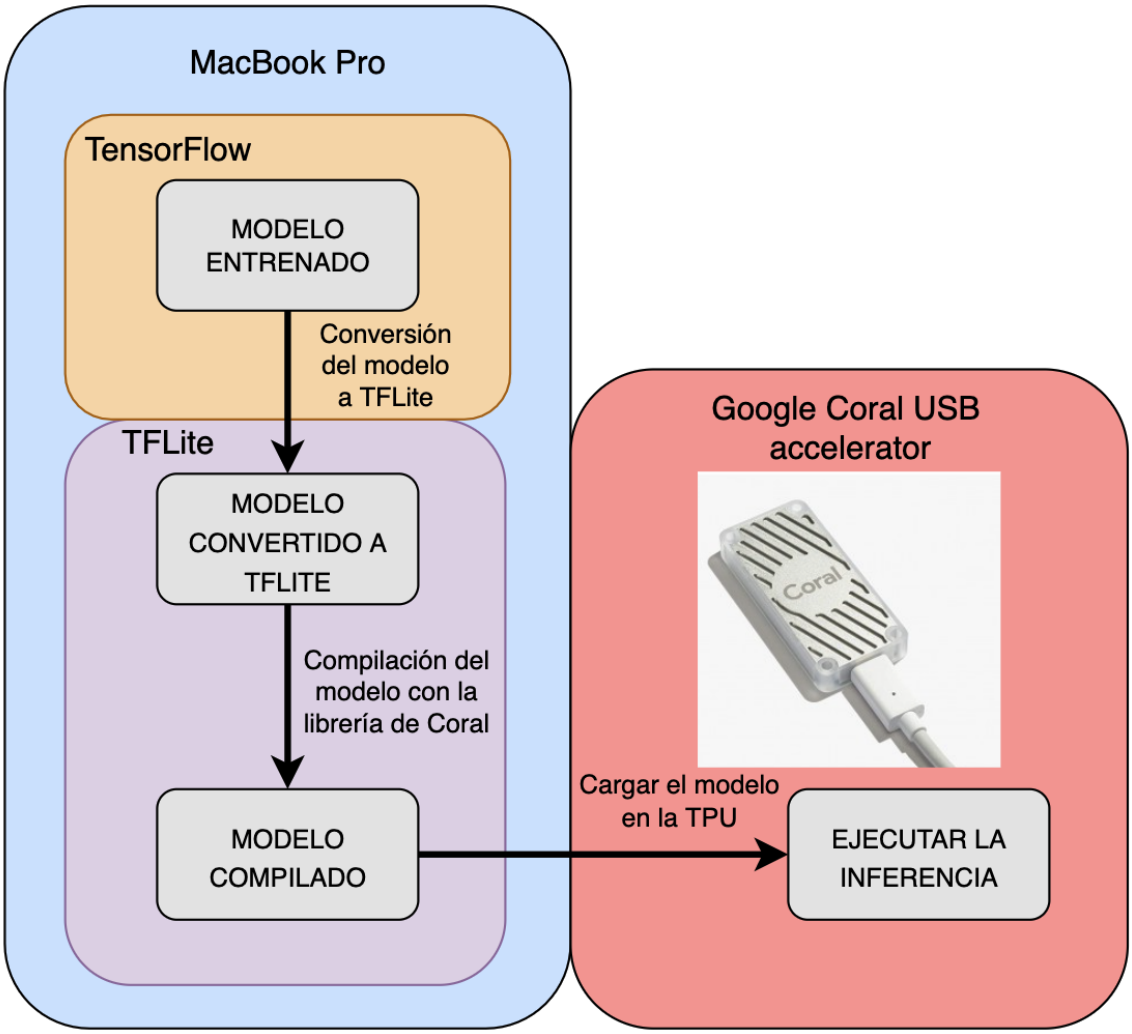


Figura 19. Ejecución de la inferencia en la Edge TPU de google Coral

Los experimentos se han realizado sobre distintos tamaños para ambas arquitecturas, tanto RNN como CNN, de manera que se pueda observar la influencia que tiene el tamaño del modelo sobre el tiempo de la inferencia. Para esta técnica se utilizará un lote fijo de 10 muestras. A continuación, se listan los experimentos a realizar:

1. Se ejecutan los modelos con la arquitectura basada en RNN y CNN utilizando la CPU del MacBook. Se almacenan los tiempos que este ha tardado en ejecutar la inferencia para tener como punto de comparación.

2. Se ejecutan los modelos planteados para las arquitecturas basadas en RNN y CNN en la Edge TPU y se almacenan los resultados de tiempo y precisión obtenidos de la inferencia.

6. Resultados

6.1. Quantization

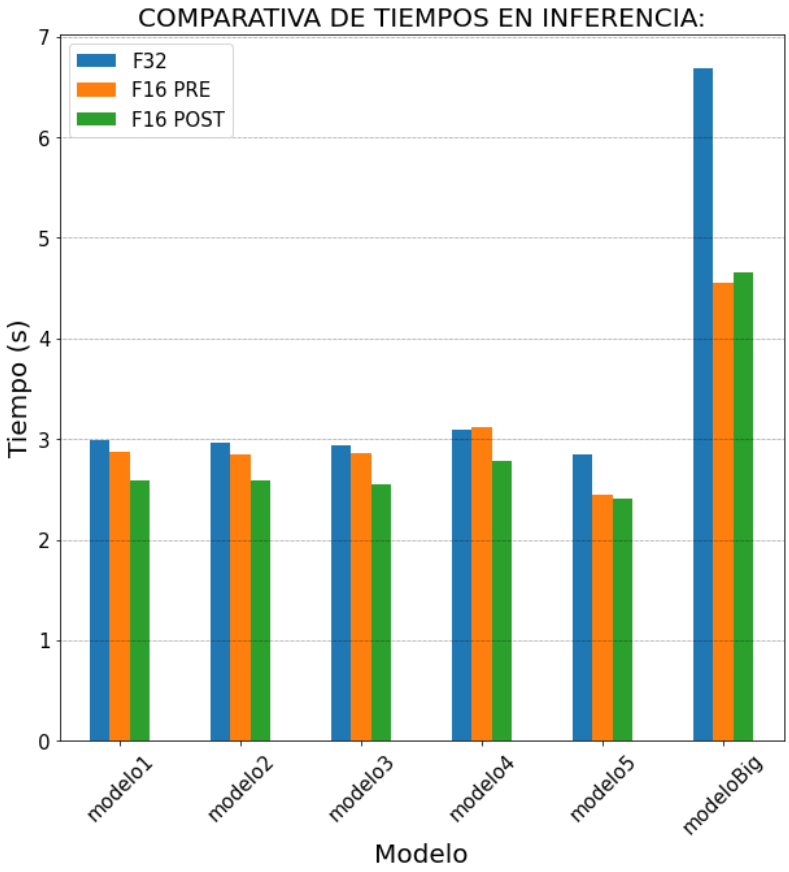
Como ya se definió en el apartado 5.1, para este bloque de experimentos, se utilizarán modelos basados en arquitecturas de RNN. Se utilizaron 6 modelos distintos, donde sus características están especificadas en la Tabla 4, indicando el número de parámetros que posee cada modelo, su nombre y su arquitectura por capas.

Nombre	Número de parámetros	Arquitectura (Nº neuronas por capa)
Modelo1	323.585	[256,128]
Modelo2	646.913	[512,128]
Modelo3	1.687.553	[1024,512]
Modelo4	1.818.625	[1024,512,256]
Modelo5	1.851.393	[1024,512,256,128]
ModeloBig	21.434.369	[4096,4096,4096,4096]

Tabla 4. Información sobre modelos basados en RNN para ensayos de Quantization

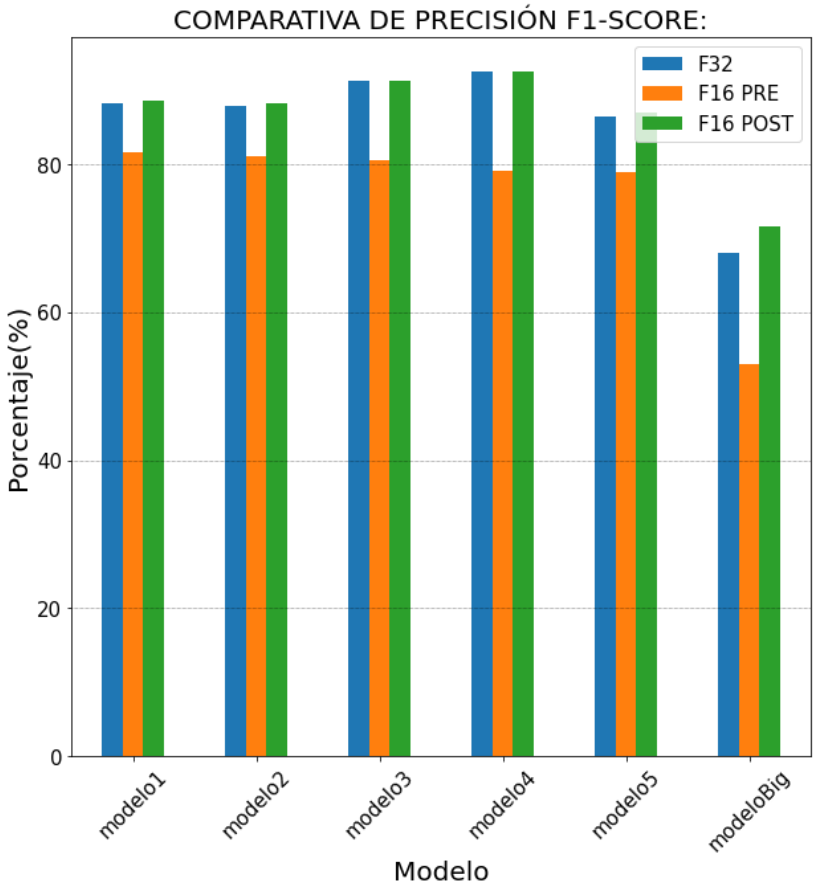
Como se puede apreciar en la tabla anterior, se escogió una cierta variedad en la arquitectura de los modelos, modificando la profundidad (número de capas que tiene el modelo) y densidad (número de neuronas por capa), de manera que se pueda apreciar cómo el cambio en la arquitectura y el número de parámetros final afecta al rendimiento de los modelos utilizando un tipo de dato u otro. En la Figura 20a, se representan los resultados para las distintas casuísticas planteadas en términos de tiempo que tardan los modelos en realizar la inferencia, utilizando la métrica mencionada en el apartado 4. Por otro lado, en la Figura 20b, se muestran los resultados para los distintos modelos planteados en términos de precisión del modelo.

a)



Hardware: GPU

b)



Hardware: GPU

Figura 20a. Resultados Quantization en términos de precisión F1-score; Figura 20b. Resultados Quantization en términos de tiempo

Comenzando con los resultados obtenidos en términos de tiempo, se puede observar que la diferencia entre los valores no es muy notable para los modelos 1, 2, 3, 4 y 5, de echo, se gana entre un 10% y un 15% para estas arquitecturas, siendo esta ganancia calculada con la formula (7) del apartado 4.

En cambio, en el *modeloBig*, se aprecia una diferencia notable y una ganancia entorno al 31%. Esto se debe principalmente a que el cuello de botella pasa a ser el propio hardware a medida que se modifica el tamaño del modelo ya que, la velocidad a la que se realizan las operaciones está limitada por la cantidad de ellas que es capaz de hacer de forma paralela la GPU. Además, la velocidad de estas es independiente del tipo de dato con el que se esté trabajando. La técnica de Quantization, empieza a dar sus frutos en cuanto el modelo es lo suficientemente complejo para que la resolución utilizada en los cálculos pase a ser el elemento determinante a la hora de calcular el tiempo que se tarda en realizar la inferencia. A medida que el modelo se hace más complejo, la GPU llega un momento que queda sin espacio de memoria y, en consecuencia, tiene que descargar los pesos de las primeras operaciones y volver a cargar los de las siguientes, esto supone un retraso temporal que, gracias a la reducción de la resolución de los valores cargados se ve reducido al tardar menos.

Por otro lado, para el tiempo de inferencia, la técnica de Quantization aplicada es un poco indiferente. Para hacerse una idea, la diferencia de tiempo absoluta entre las dos técnicas aplicadas oscila entre el 2 y 10% para todos los modelos sobre los que se ha aplicado, siendo esta diferencia absoluta calculada con la fórmula (11) del apartado 4.

En términos de precisión, se mantiene la diferencia de valores siempre en la misma proporción, es decir, la diferencia entre aplicar una técnica de Quantization o no, es siempre la misma y, además, esta no cambia con el tamaño del modelo. Para poner valores, tomando el *modelo1* y el *modeloBig*, que son los modelos menos y más complejo respectivamente y utilizando la fórmula (7) del apartado 4 se obtiene una diferencia absoluta entre precisiones para el *modelo1* del 0.08% y para el *modeloBig* del 5%.

Otro factor a destacar de los resultados es la pérdida de precisión de los modelos tras aplicar la técnica de Quantization Pre-training. Esto se debe a que, a la hora de entrenar el modelo en menor resolución, los valores pierden precisión a la hora de ajustar la función de coste durante el entrenamiento y, en consecuencia, no se llega a entrenar con la misma precisión que se logra con los modelos de resolución original. Otro factor que puede justificar esta bajada en la precisión, son los propios datos de entrenamiento, ya que estos se deben de convertir a FP16 y en consecuencia el ajuste de los pesos, no se haga con tanta precisión.

6.2. Reducción de píxeles a procesar

Para este segundo bloque de experimentos, se usará un modelo basado en RNN y, para este bloque en concreto, no se consideró necesario tener una variabilidad en el tamaño del modelo ya que la variable modificada para esta técnica es el número de puntos procesados por el modelo. Como ya se especificó en el apartado 4.2 del documento, se plantearon dos alternativas: eliminar puntos aplicando un patrón predefinido o generado de forma aleatoria.

Para la elaboración de estos experimentos, se consideró necesario utilizar piezas ya etiquetadas, es decir, que de antemano se posee una imagen con los puntos que corresponden al defecto. La métrica de precisión en este caso es el residuo generado como la diferencia entre la imagen de salida del modelo y la imagen de etiquetas siguiendo la ecuación (8) del apartado 4.

Por otro lado, para ver cómo es la mejora frente al planteamiento original, es decir, sin aplicar ningún patrón, se calculó la diferencia porcentual neta entre los residuos como se especifica en la fórmula (9) del apartado 4.

La otra métrica utilizada es la temporal que, combinada con la diferencia de residuos, se utilizarán para una evaluación de las distintas casuísticas planteadas. La métrica temporal no es más que el tiempo que tarda el modelo en realizar la inferencia. Al igual que con el residuo se calculó la diferencia porcentual neta entre tiempos de inferencia como se especifica en la fórmula (10) del apartado 4.

Comenzando con la primera alternativa, se generará una máscara binaria a partir de la máscara que indica la región de interés a procesar por el modelo, como se explicó en el apartado 2.3.2, donde se quitan un porcentaje de los puntos a procesar de forma aleatoria. Se decidió modificar el porcentaje de puntos a no procesar por el modelo, yendo desde un 2% a un 50%.

En la Figura 21a se muestra una gráfica donde se representa cómo varía la *Diferencia Porcentual del Residuo* frente al porcentaje de puntos no procesados y en la Figura 21b, como varía la *Diferencia Porcentual Temporal* frente al porcentaje de puntos no procesados. Cada punto dentro de la nube representa una pieza distinta.

Por otro lado, en la Figura 22a se representa la media de cada nube de puntos para cada porcentaje de puntos no procesados del la *Diferencia Porcentual del Residuo* y *Diferencia Porcentual Temporal*, en la Figura 22b se muestra la desviación típica de cada nube de puntos para cada porcentaje de puntos no procesados.

a) **Hardware: CPU** *b)*

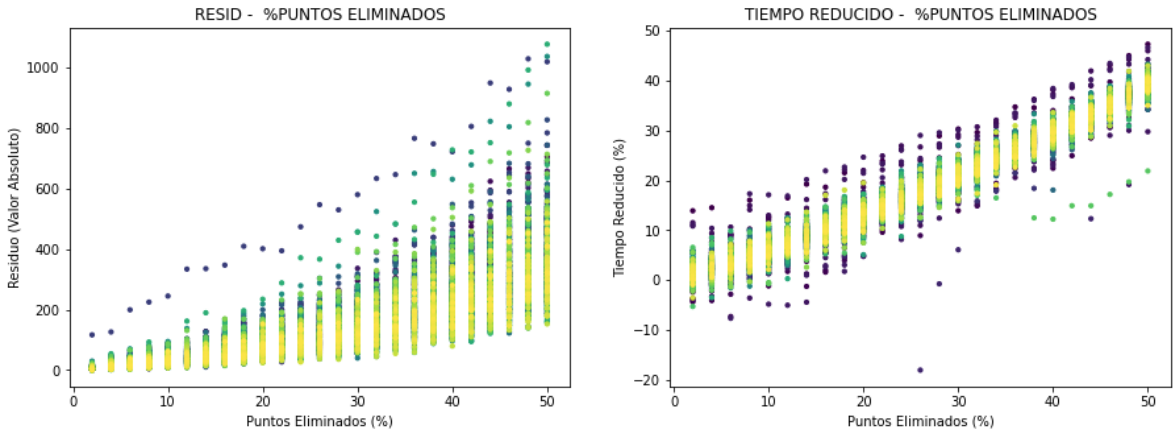
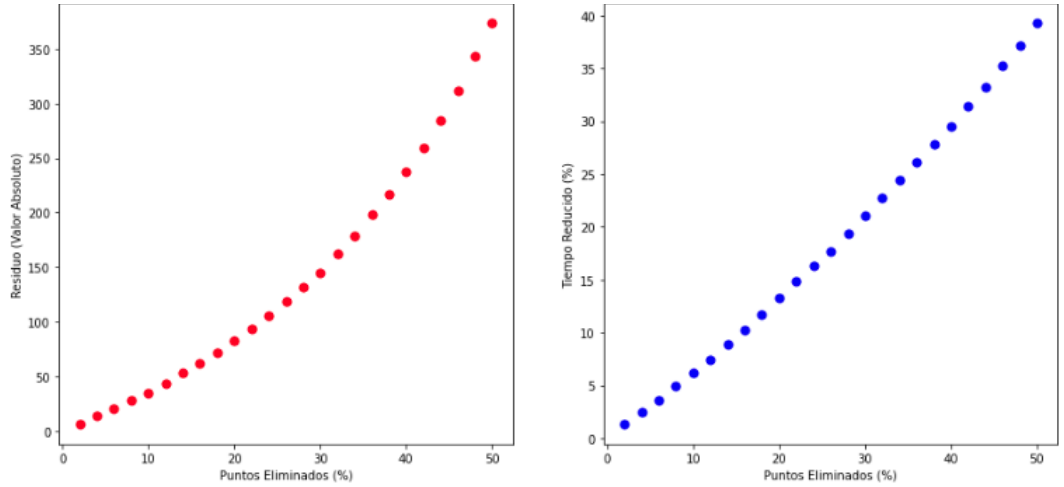


Figura 21a. Gráfica del Residuo-%Puntos eliminados. Figura 21b. Gráfica del Tiempo Reducido-%Puntos eliminados

Media: Resid-%Puntos eliminados *Media: Tiempo reducido-%Puntos eliminados*

a)



STD: Resid-%Puntos eliminados *STD: Tiempo reducido-%Puntos eliminados*

b)

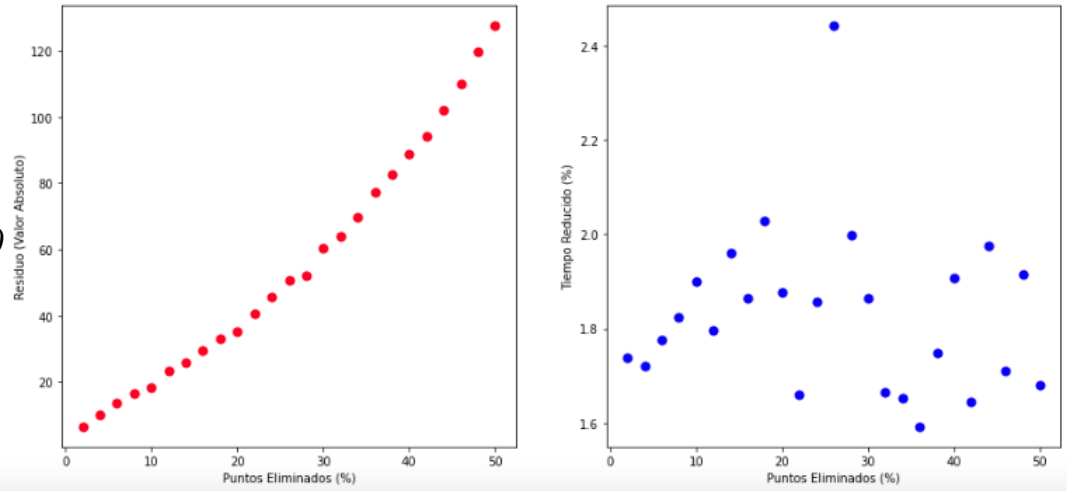


Figura 22a. Gráficas de la media del Residuo-%Puntos eliminados y Tiempo Reducido-%Puntos eliminados. Figura22b. Gráficas de la desviación típica del Residuo-%Puntos eliminados y Tiempo Reducido-%Puntos

En la primera figura, se representan los resultados obtenidos para todas las piezas, donde cada punto dentro de cada agrupación vertical representa el resultado de una pieza en concreto en unas condiciones concretas, es decir, se realiza el ensayo sobre todas las piezas en todas las condiciones planteadas, donde la variable modificada es el porcentaje de puntos no procesados. Para la gráfica referente al residuo, se puede apreciar una tendencia de tipo exponencial, en cambio, para la gráfica referente al tiempo, se puede ver un aspecto lineal. Ambas tendencias se ven de forma mucho más clara en la segunda figura, donde cada punto, representa la media de los resultados de todas las piezas quitando ese porcentaje de puntos en concreto.

Con estos resultados, se demuestra que el modelo, al procesar menos puntos, le lleva menos tiempo realizar la inferencia y que, además, esta tiene una relación lineal con el porcentaje de ellos que se quitan. En definitiva, el modelo al procesa menos puntos, suponen menos operaciones y en consecuencia menos tiempo durante la inferencia.

Por otro lado, en términos de precisión, se observa una relación exponencial entre el número de puntos eliminados y el residuo generado. Se esperaba que el residuo aumentase a medida que se aumenta el número de puntos no procesados. Una posible razón de la relación exponencial es que, al generar el patrón de forma aleatoria, puedan aparecer áreas donde se acumulan puntos a no procesar, y estos coincidan con un defecto y, en consecuencia, se reduzca la precisión y esta vaya empeorando a mayor número de puntos no procesados.

En la industria, siempre se intenta buscar la repetitividad, es decir, si el modelo realiza sobre una misma pieza la inferencia 100 veces, se espera que el resultado sea el mismo siempre y la aleatoriedad puede causar que esto no ocurra y en consecuencia no se trate de una técnica adecuada. Para la segunda alternativa, se plantearon 3 patrones distintos. La forma de generar la máscara se realizó de la siguiente manera: se comenzó escogiendo un porcentaje de puntos a no procesar, para este bloque de experimentos, se decidió quitar aproximadamente un 83% y cerca de un 90% de los puntos. A continuación, se escogió la estructura de los patrones, se definieron dos patrones que eliminaran el 83% de los puntos y uno que eliminara el 90%. En la Figura 23 se muestra cómo se parte de la estructura del patrón, cómo esta se transforma en una máscara binaria y se extiende al tamaño de la imagen al completo.

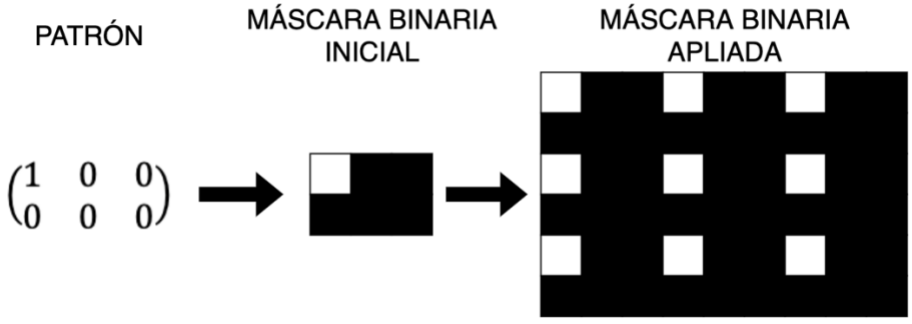


Figura 23. Generación de máscara binaria basada en el patrón 1

El patrón utilizado en la Figura 23, es el correspondiente al primer patrón que supone quitar aproximadamente el 83% de los puntos. En la Figura 24, se muestra el segundo patrón correspondiente a eliminar el 83% de los puntos. Por último, en la Figura 25, se muestra la estructura del patrón correspondiente a eliminar cerca del 90% de los puntos.

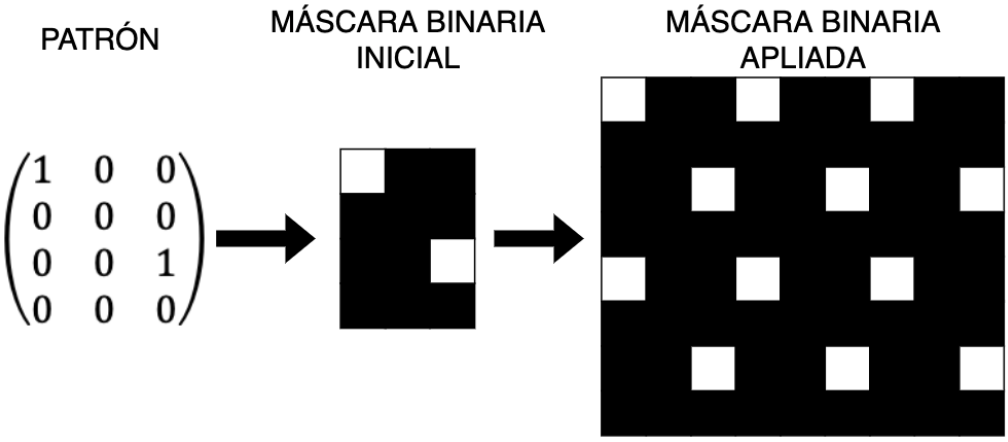


Figura 24. Generación de máscara binaria basada en el patrón 2

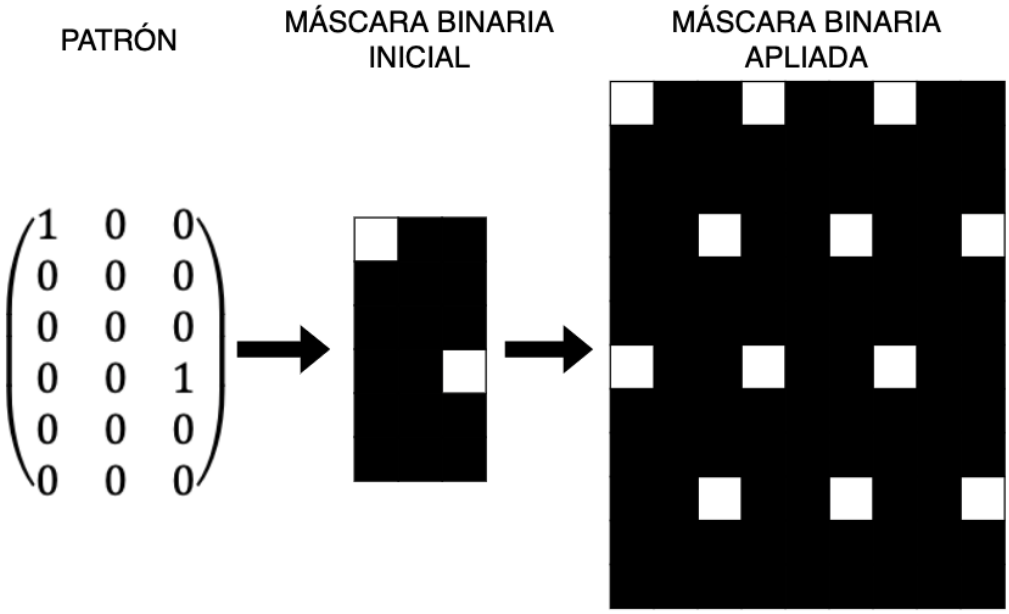


Figura 25. Generación de máscara binaria basada en el patrón 3

Para esta segunda alternativa, los resultados se mostrarán de distinta manera, ya que la variable modificada entre cada ensayo es el porcentaje de puntos no procesados asociados a un patrón con un aspecto concreto y, en definitiva, al fijarse dos porcentajes lo que realmente cambia es el patrón. Del mismo modo, se realizaron las pruebas sobre todas las piezas y las métricas utilizadas son las mismas, el residuo generado y el tiempo que se tarda en realizar la inferencia.

En esta parte de los experimentos, se decide representar gráficamente la diferencia porcentual del residuo generado frente a la diferencia porcentual del tiempo que tarda en ejecutarse la inferencia, ambas métricas mencionadas en el apartado 4. Para visualizar mejor los resultados, se decidió utilizar la librería de Python *Seaborn* [71], que permitió representar puntos acumulados entre un cierto rango de valores como uno solo, siendo el canal del color el que representa el número de puntos que se concentran entre esos valores. En la Figura 26, se representan los resultados correspondientes a el patrón 1.

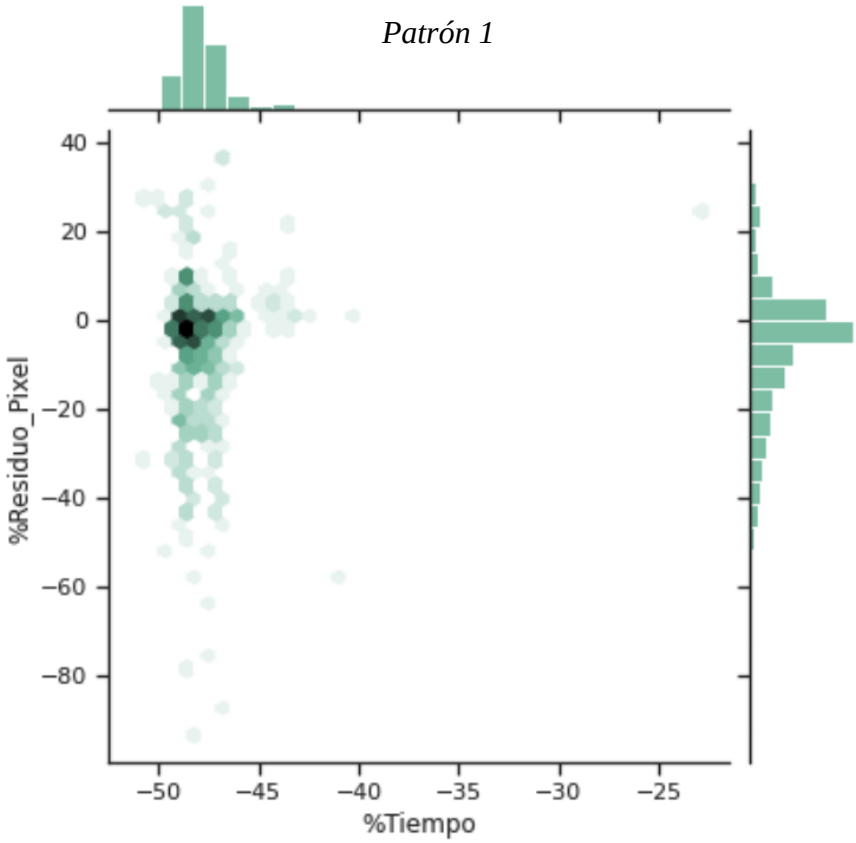


Figura 26. Resultados tras la aplicación del patrón 1

Como se puede observar, a diferencia del primer bloque de resultados, el residuo generado no oscila con respecto a la situación inicial, y en una buena parte de los casos, es incluso menor. Esto probablemente se deba, a que la distribución de los puntos a procesar, tras la aplicación del patrón sea más homogénea a lo largo de la imagen. Por otro lado, en términos temporales, el porcentaje de tiempo se acumula en un rango de valores específico como era de esperar, ya que se está quitando siempre la misma proporción de puntos y, en consecuencia, esto supone realizar siempre aproximadamente el mismo número de operaciones. En la Figura 27, se muestran los resultados correspondientes al patrón 2.

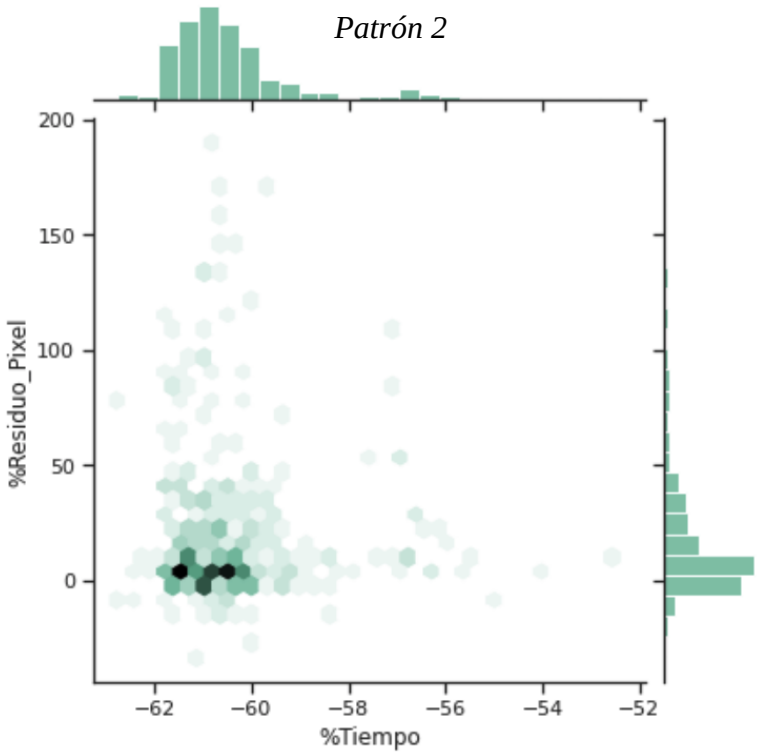


Figura 27. Resultados tras la aplicación del patrón 2

Para este segundo patrón, se puede observar que en términos temporales los resultados son un poco más favorables, rondando un 60% menos a la hora de ejecutar la inferencia, por otro lado, el residuo generado es un poco mayor. Por último, la Figura 28 muestra los resultados correspondientes al patrón 3.

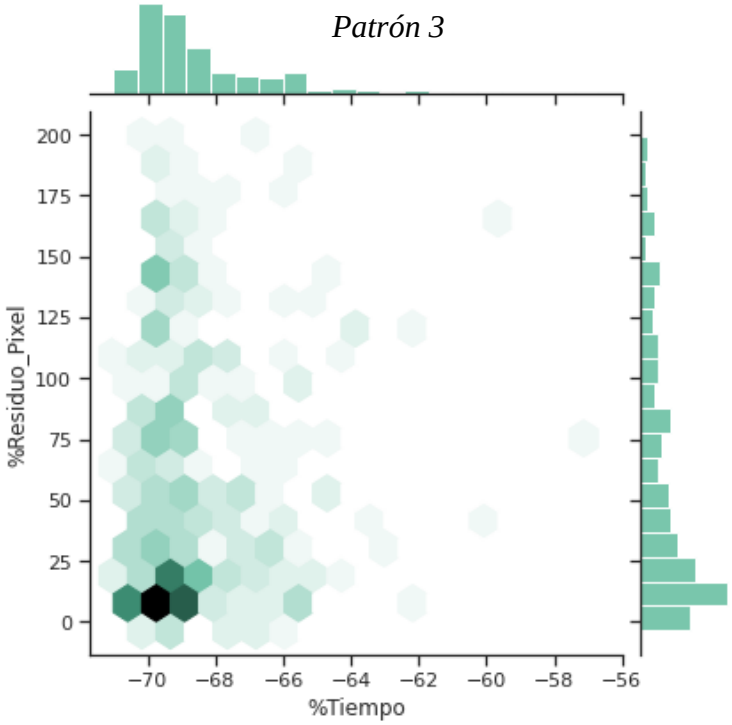
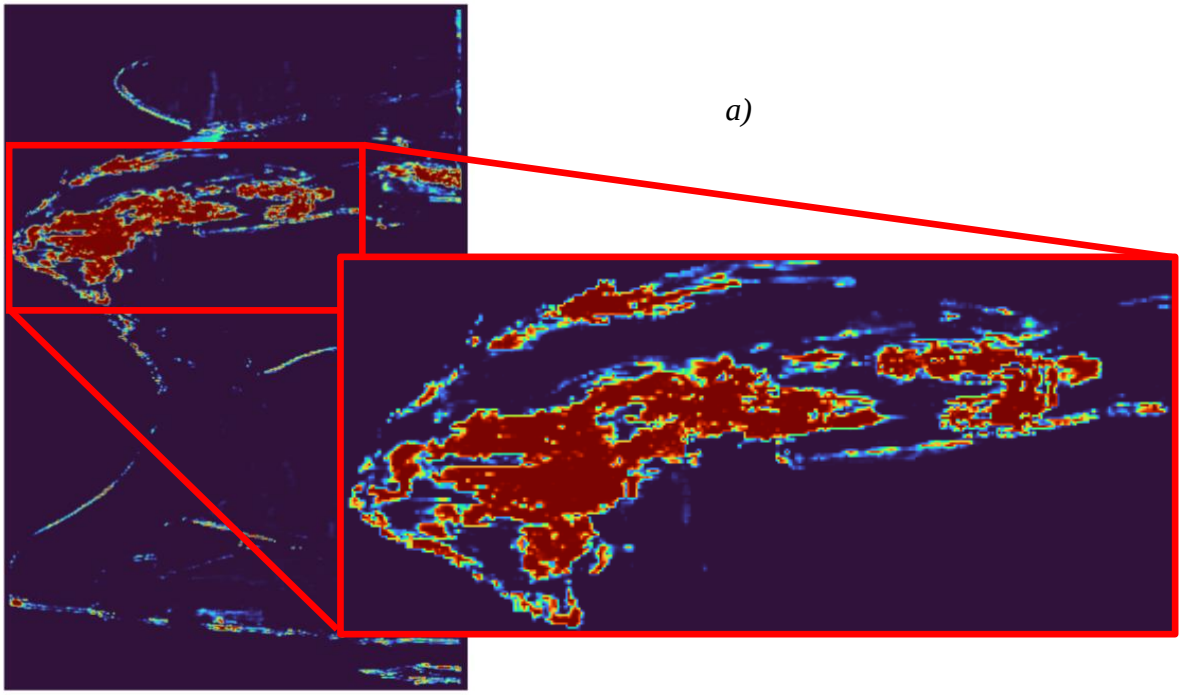


Figura 28. Resultados tras la aplicación del patrón 3

Para este último patrón, se puede ver que los resultados son bastante similares en términos de residuo con respecto a los otros dos patrones. Por otro lado, en términos temporales, en la mayoría de los casos, se reduce cerca de un 70% el tiempo de inferencia.

En esta segunda parte de los ensayos, se puede apreciar una mejora notable en el residuo generado ya que, en los tres patrones aplicados, los valores son más estables. No hay que olvidar que se están procesando tan solo el 17% y el 10% de los puntos con respecto a la imagen original y que, gracias a ello, la reducción en tiempo de inferencia siempre está entre el 50% y el 70% del tiempo que tardaba originalmente.

Para visualizar más en detalle en que consisten las tareas de post-procesamiento mendionadas en el apartado 2.3.2, en la Figura 29 se muestra la diferencia que hay entre la salida del modelo original, la salida con el patrón aplicado sin post-procesamiento y por último la salida tras haber aplicado el patrón con post-procesamiento.



a)

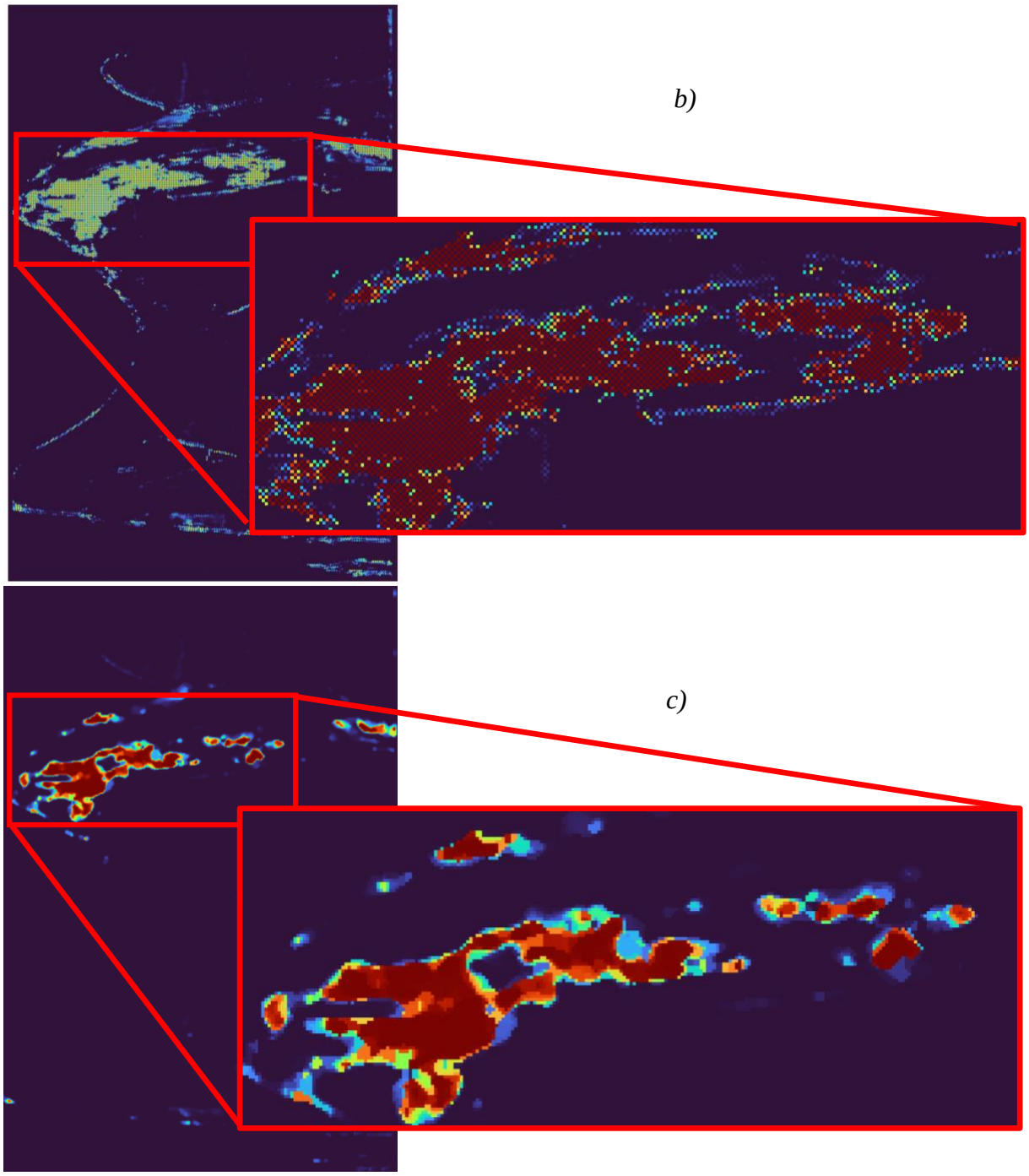


Figura 29.a) Salida del modelo original; b) Salida del modelo tras aplicar el patrón sin tareas de post-procesamiento; c) Salida del modelo tras aplicar el patrón con tareas de post-procesamiento

Si se vuelven a las dos alternativas planteadas, en realidad, la segunda opción se plantea debido a que una distribución aleatoria puede dar pie a la aparición de una zona donde se centren los puntos eliminados y en consecuencia no tener buenos resultados en la predicción de los defectos pequeños que pueden encontrarse en dicha zona.

6.3. Edge TPU

Para este último bloque de experimentos se utilizó la *Edge TPU de Coral* y el *Macbook Pro* para ejecutar la inferencia. Se emplearon las dos arquitecturas mencionadas en el apartado 2.2 modificando su complejidad (número de parámetros) para ver la influencia de esta en el rendimiento. En la Tabla 5, se muestran la estructura de los modelos utilizados.

Modelos RNN densas		
Nombre	Nº de neuronas por capa	Nº de parámetros
modeloDense_1	[256,128]	323,585
modeloDense_2	[512,256,128,128]	1,024,641
modeloDense_3	[2048,2048,512,256,128]	7,734,273
modeloDense_4	[2048,2048,2048,512,512,512]	12,292,097
Modelos CNN UNET		
Nombre	Nº de filtros para encoder y decoder de la UNET	Nº de parámetros
modeloUNET_1	[64]	412,161
modeloUNET_2	[64,128]	1,871,745
modeloUNET_3	[64,128,256]	7,709,313
modeloUNET_4	[64,128,256,512]	31,049,776

Tabla 5. Estructura de los modelos para los ensayos con la Edge TPU de Coral

Como ya se hizo con otras técnicas, se utilizó la métrica *F1-score* para medir la precisión de los modelos RNN y el *Coeficiente de Sorensen-Dice* para las CNN. Por otro lado, se midieron los tiempos que tardaba la inferencia en ejecutarse para cada uno de ellos y así ver el comportamiento de esta técnica en términos temporales. En la Figura 30, se muestran los resultados en términos de precisión para los modelos planteados.

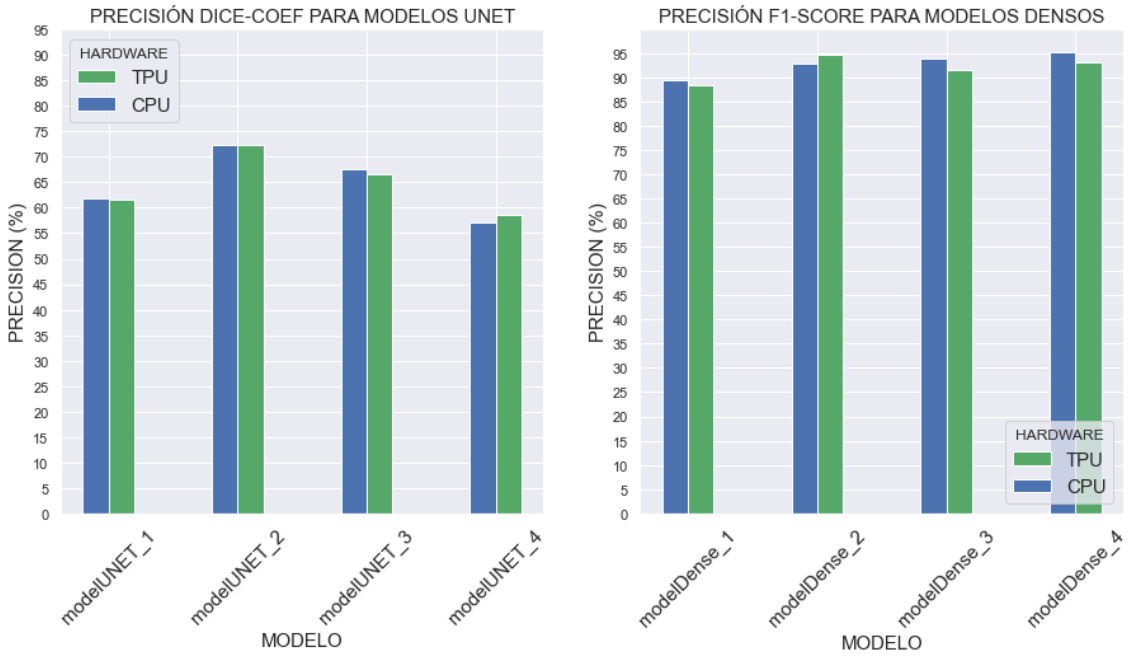


Figura 30. Resultados de precisión para los modelos en la Edge TPU y la CPU

Como se puede apreciar, la Edge TPU aún trabajando en INT8 no pierde apenas precisión durante la inferencia. Esto probablemente se deba en mayor parte a la técnica aplicada a la hora de generar los modelos, *Quantization-aware*, y el posterior *fine tuning* realizado para ajustar los valores máximos y mínimos asociados a los datos de entrada, de manera, que a la hora de realizar la conversión la pérdida de información sea la menor posible.

En la Figura 31 se muestran los resultados de los modelos, pero en términos temporales.

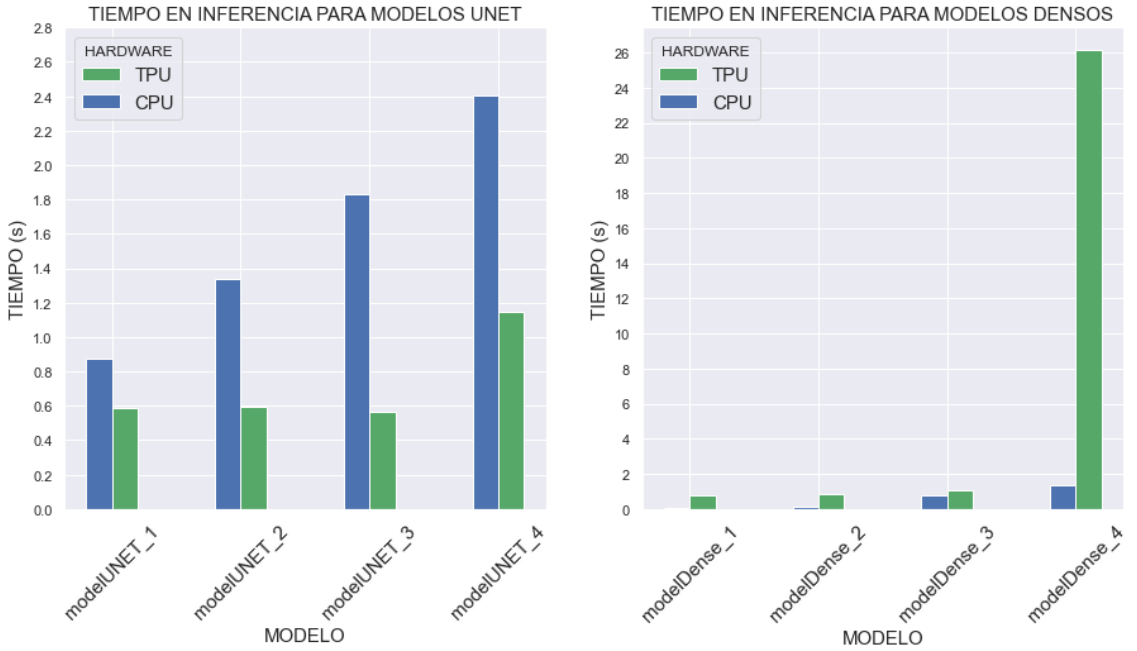


Figura 31. Resultados de tiempo para los modelos en la Edge TPU y la CPU

En términos de tiempo, los resultados son muy favorables en modelos basados en CNN, donde se logra reducir el tiempo de inferencia ejecutando los modelos en la Edge TPU, hasta un 70% con respecto al tiempo que tarda la CPU. Por otro lado, para las RNN densas los resultados son muy desfavorables, llegando a tardar, en el peor de los casos, 20 veces más.

Una posible justificación de por qué el rendimiento para las RNN densas es tan malo en comparación a las CNN, es por cómo se realizan las operaciones en la TPU internamente debido a la propia arquitectura de los modelos. Como ya se mencionó en el apartado 2.3.3, los pesos se cargan en los ALUs de la TPU de cara a que, a la hora de realizar la inferencia, estos no tengan que acceder a la memoria y ahorrar tiempo de cálculo. Pero hay una diferencia en cómo se realizan las operaciones con las CNN respecto a como se hacen con las RNN densas.

En las CNN se realizan operaciones de convolución y los *kernels* con los que se realizan dichas operaciones se comparten, por lo tanto, esto permite optimizar el espacio y las operaciones realizadas en la TPU. En cambio, para las RNN, el número de pesos depende del tamaño de la capa y del número de entradas a la capa, por lo tanto, si el número de pesos supera el espacio de memoria de la TPU, esta serializará las operaciones, es decir, realizará los cálculos por grupos, descargando los pesos y cargando los nuevos hasta finalizar con las operaciones de esa capa. En la Figura 32 se muestra la velocidad a la que se cargan los pesos en la *Edge TPU de Coral*

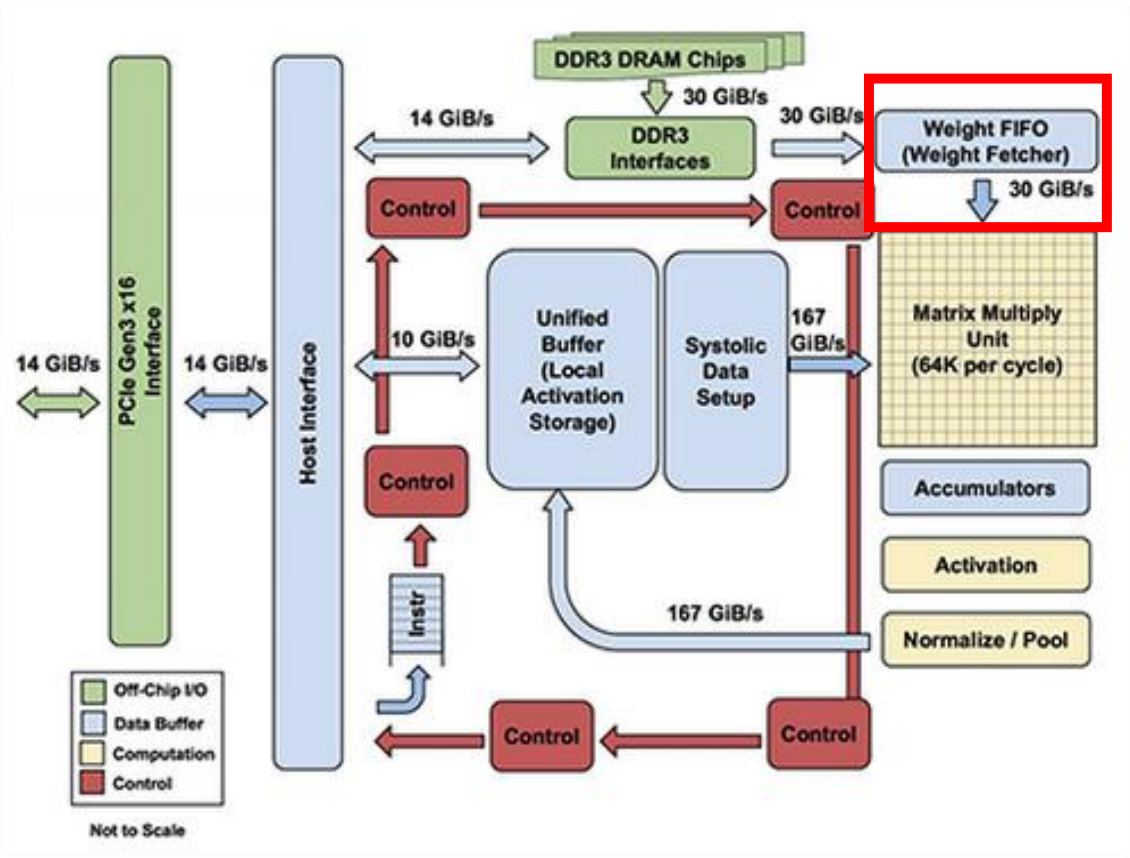


Figura 32. Arquitectura interna de la Edge TPU de Coral señalando la velocidad de carga de los pesos (Fuente: [72])

En la Figura 33 se muestra un diagrama, mostrando la diferencia a la hora de realizar las operaciones en la TPU, para una arquitectura basada en RNN densas y CNN.

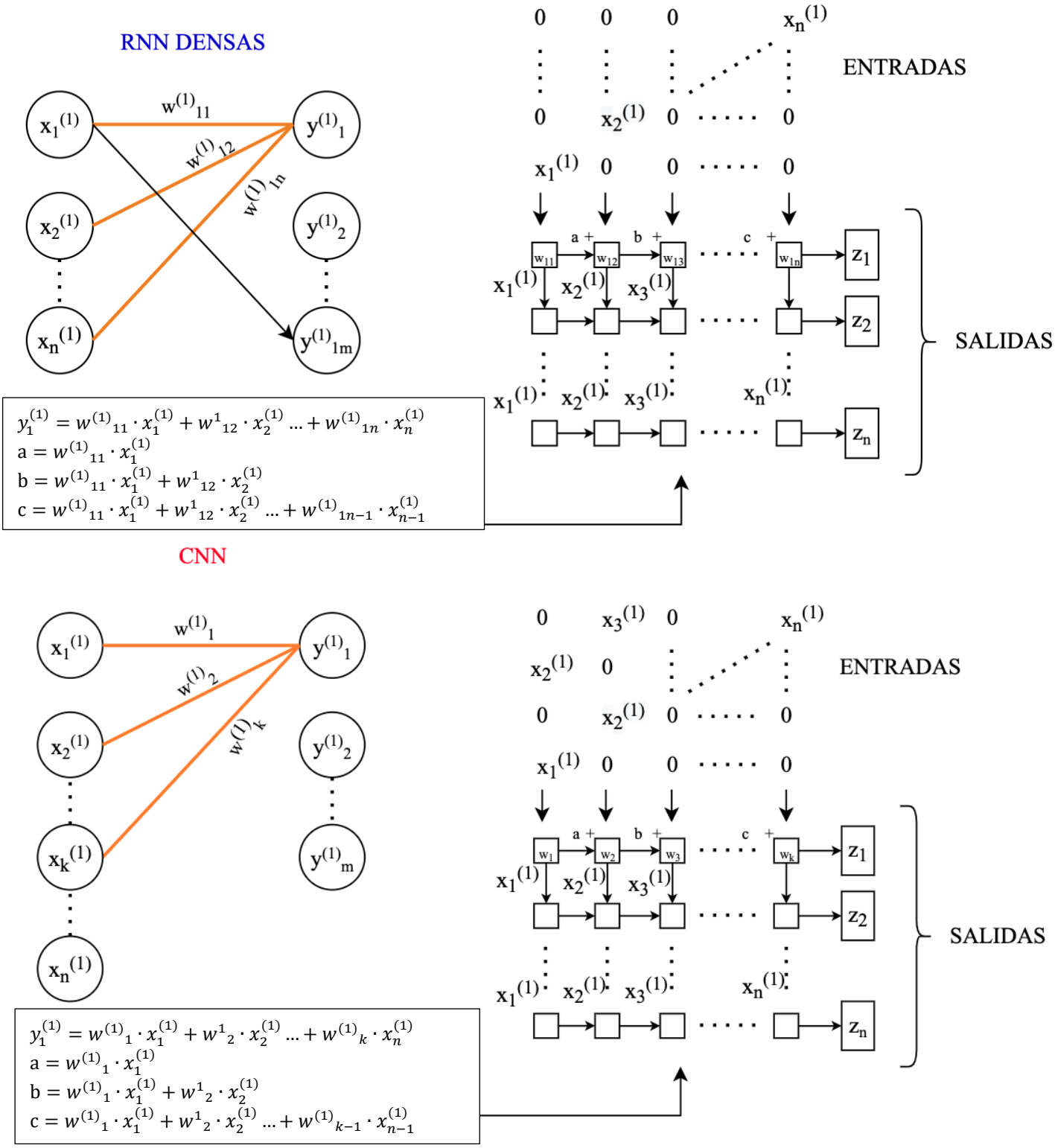


Figura 33: Diagrama que representa como se realizan las operaciones en la TPU para RNN densas y CNN

Para las RNN, sí la TPU es de tamaño $\mathbb{R}^{m,m}$ en caso el caso de que $m > n$, no hay problema, pero, si $m \ll n$, como puede ocurrir con las RNN complejas puede ocurrir lo mencionado anteriormente. Para dar una idea la TPU de Coral es de $\mathbb{R}^{256x256}$, entonces una red con más de 256 pesos no entraría. Además, sí $m \ll n$, $z_1 \neq y_1^{(2)}$ entonces el resultado z, ha de ser guardado en un acumulador, descargar los pesos $w_{11}, w_{12}, \dots, w_{1m}$, cargar los pesos restantes $w_{1,m+1}, w_{1,m+2}, \dots, w_{1,n}$, repetir la operación y, por último, se suma el acumulador para obtener $y_1^{(2)}$.

Para las CNN, como se puede observar en la Figura 33, el número de salidas dependen del número de filtros que se apliquen en cada capa de convolución, en este caso el tamaño del kernel está representado por k, donde $k < n$. Estos por lo general, suelen ser pequeños (p.ej. 3x3, 4x4 , es decir 9 y 16 pesos respectivamente) y, en consecuencia, suponen un número bajo de pesos a diferencia de las RNN donde el número de pesos depende del tamaño de la capa y del número de entradas a la capa. Como ya se mencionó, en las convoluciones se comparte el kernel utilizado y en este caso, lo que se cambia es el orden de los valores de entrada.

7. Conclusiones

7.1. Discusión

Comenzando con los objetivos iniciales, se ha logrado explorar distintas técnicas basadas en la reducción de la resolución de los pesos con el objetivo final de mejorar la eficiencia computacional de modelos ML. Como ya se mostró en el apartado 6.1, trabajando con un modelo en FP16, se ha logrado reducir el tiempo de inferencia hasta en un 31%, que se cree que puede ser mayor, con modelos aún más complejos (con mayor número de parámetros).

La técnica propuesta que tenía como objetivo la reducción del número de píxeles a procesar por el modelo ha resultado ser más fructífera de los esperado ya que, como se demostró, al aplicar patrones generados de forma manual justo con las tareas de post-procesamiento, se ha logrado reducir hasta en un 70% el tiempo de inferencia sin apenas aumentar el error en la imagen de salida.

Por otro lado, se ha logrado demostrar con éxito el potencial de elementos hardware externos como la Edge TPU de Coral, en la detección de defectos superficiales a partir de imágenes de distancias que, en el caso de la CNN se ha logrado reducir un 70% el tiempo de inferencia sin apenas perder precisión y para las RNN se ha visto que los resultados no son nada buenos a la hora de utilizar este dispositivo.

Todos los experimentos asociados a las distintas técnicas propuestas se han realizado bajo una serie de requisitos y, durante la ejecución, se ha asegurado la veracidad de los resultados repitiéndolos varias veces. Junto a los requisitos, se declararon una serie de métricas, algunas ya conocidas en el mundo del ML y otras, generadas para este proyecto en particular, como las mencionadas en el apartado 4.

Una posible limitación o acotación en este proyecto ha sido el hardware utilizado. Los resultados obtenidos dependen en gran medida del hardware utilizado durante este proyecto y esto probablemente comprometa en cierta medida la reproducibilidad de los experimentos aquí expuestos. Otras alternativas hardware de cara a la aplicación de estas técnicas, serían las *Field-programmable gate array* (FPGA) [73] o *Digital signal processor* (DSP) [74], donde ambas tienen arquitecturas similares a las TPUs y serían capaces de desempeñar las tareas requeridas para ejecutar un modelo ML. Las GPU y las CPU son las arquitecturas más utilizadas en ML, pero explorar otras alternativas que sean capaces de realizar la misma tarea o incluso de manera más rápida, además de ser más económicas es lo que se pretende de cara al futuro.

7.2. Futuros proyectos

Como líneas de trabajo futuras a desarrollar de cara a la mejora de la eficiencia computacional de modelos ML, en nuestro ámbito de aplicación, se propone la exploración de técnicas más sofisticadas y novedosas que se centren en la reducción eficiente del número de pesos entrenables, tales como el *Pruning* [75] y el *Knowledge Distillation* [76]. A continuación, se describe brevemente la idea detrás de cada una de ellas.

1. *Pruning*: por lo general los modelos ML, suelen estar formados por cientos, miles o millones de parámetros y, algunos de esos parámetros tienen más importancia que otros en lo que se refiere a cómo afecta su valor en la salida de los modelos. De esta forma, los parámetros que no tienen tanta importancia se eliminarían (neuronas, pesos), logrando reducir la dimensión de los modelos y, en consecuencia, reducir el número de operaciones que debe de realizar. En la Figura 34 se muestra una idea de esta técnica.

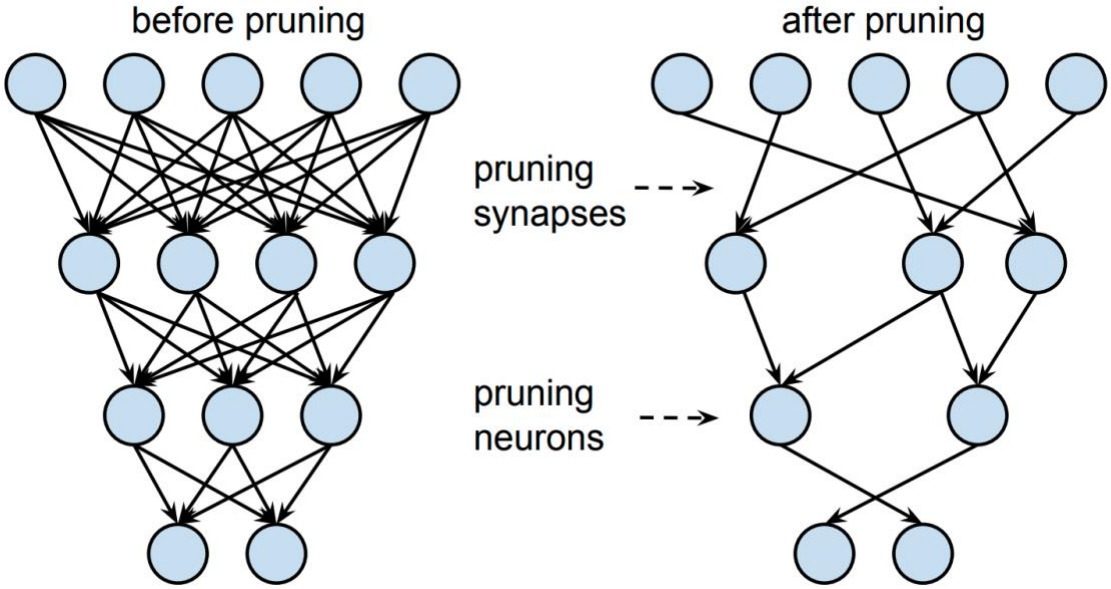


Figura 34. Representación visual del Pruning (Fuente: [77])

2. *Knowledge Distillation*: la idea de esta técnica es bastante similar al *pruning* pero, en este caso, se tienen dos puntos de vista: el llamado *profesor* y el *estudiante*. Al igual que se hace en un colegio, el profesor enseña al estudiante una parte muy concreta de su conocimiento, por ejemplo, una asignatura, de manera que, todo el conocimiento que pueda tener el *profesor* sobre dicha asignatura se condensa en la información necesaria que necesita aprender el estudiante para su desarrollo personal y profesional. Esta idea es la que hay detrás de esta técnica, se tiene un modelo complejo (el *profesor*) y lo que se pretende es transferir el conocimiento necesario para funcionar correctamente a un modelo más sencillo (el *estudiante*).

En la Figura 35 se muestra la idea de esta técnica de forma visual.

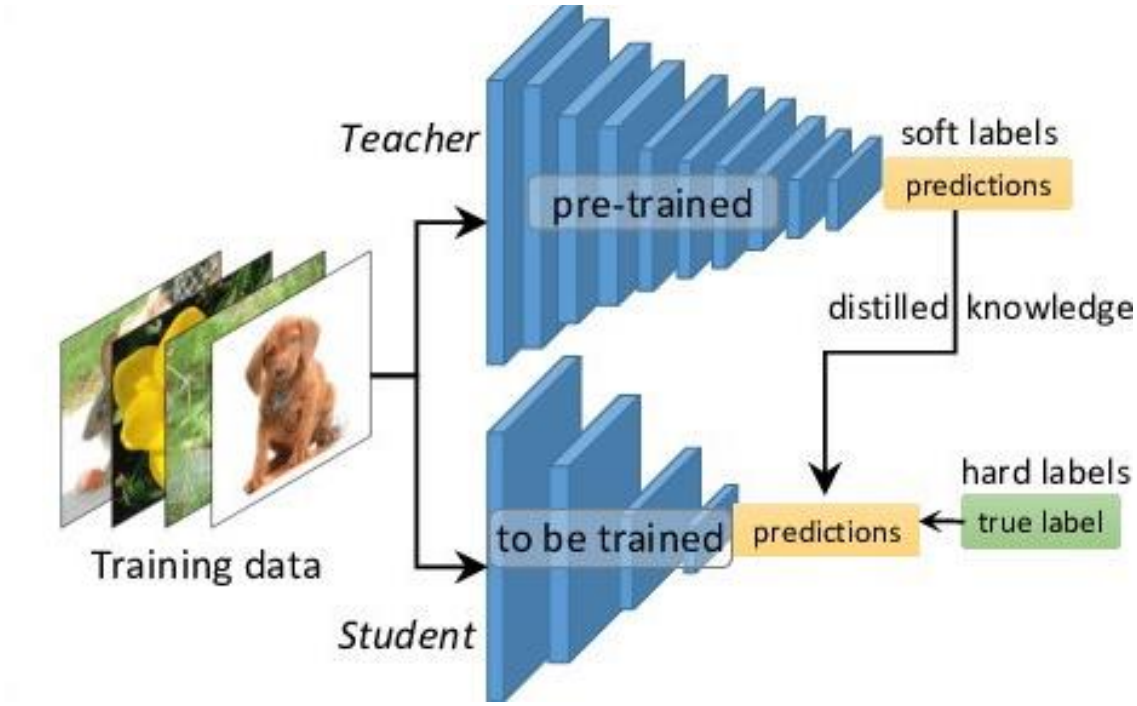


Figura 35. Representación visual del Knowledge Destilation (Fuente: [76])

7.3. Flujograma resumen

Por último, basándose en los resultados obtenidos, se plantearon los siguientes árboles de decisión de cara a la elección de una de las técnicas planteadas en función del tipo de modelo con el que se esté trabajando y la disponibilidad hardware. En la Figura 36, se muestra el árbol de decisión correspondiente a modelos basados en RNN.

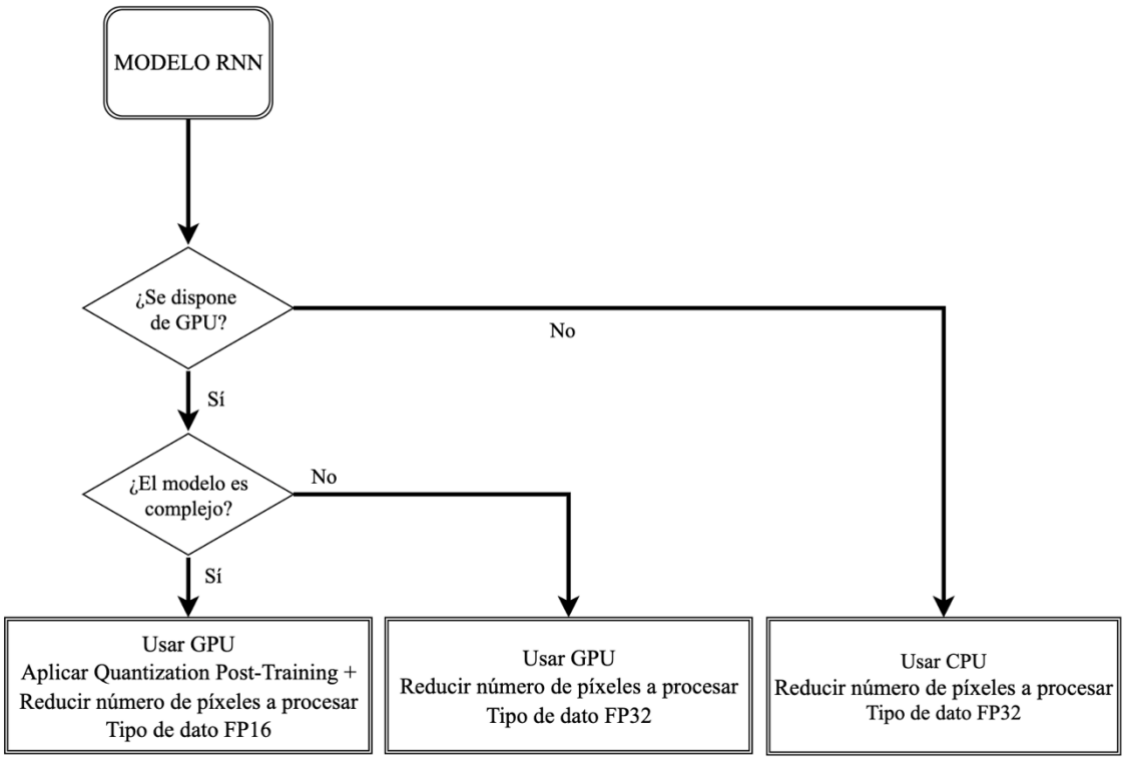


Figura 36. Árbol de decisión para modelos basados en RNN

En la Figura 37, se muestra el árbol de decisión correspondiente a los modelos basados en CNN.

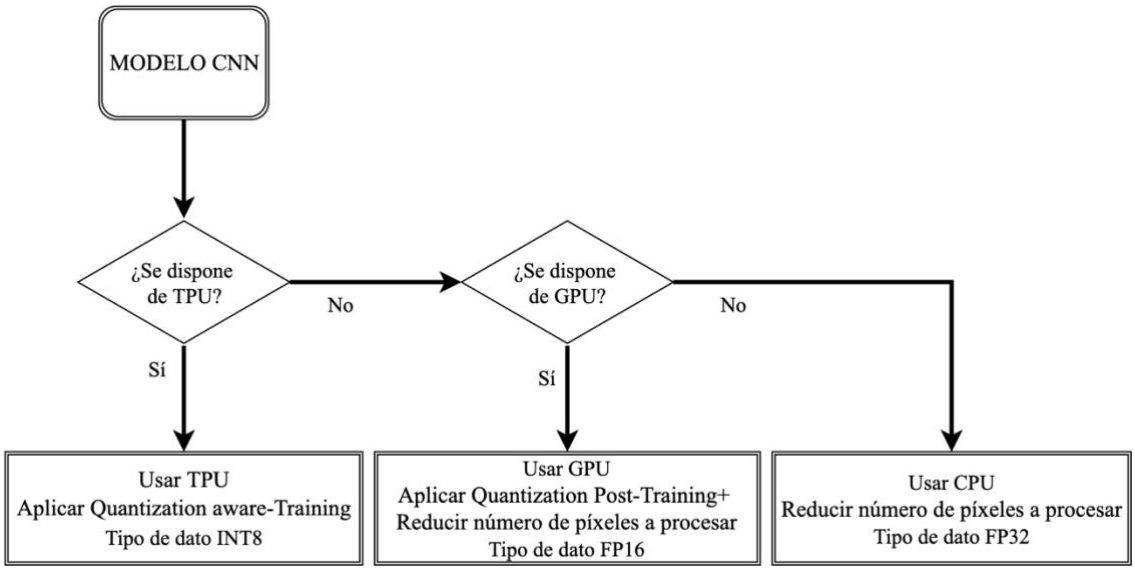


Figura 37. Árbol de decisión para modelos basados en CNN

8. Bibliografía

[1] D. Garcia Peña, "Repositorio Institucional de la Universidad de Oviedo", [En línea]. Available: https://digibuo.uniovi.es/dspace/handle/10651/58097.

[2] V. Cantavella, D. Llorens, A. Mezquita, C. Moltó, M. Bhardwaj, P. Vilanova, J. Ferrando y S. Maldonado-Zagal, "USO DE LA TÉCNICA DE ULTRASONIDOS PARA MEDIR LA DENSIDAD APARENTE DE LAS BALDOSAS EN CRUDO Y OPTIMIZAR EL PROCESO DE PRENSADO", *QUALICER*, 2006.

[3] A. P. G. León y F. Puente, "Vision Artificial", *ADIMA*.

[4] P. d. Groot, J. Biegen, J. Clark, X. C. d. Lega y D. Grigg, "Optical interferometry for measurement of the geometric dimensions of industrial parts", *OPTICA PUBLISHING GROUP*, vol. 41, 2002.

[5] Wikipedia, "Latencia", [En línea]. Available: https://es.wikipedia.org/wiki/Latencia.

[6] Wikipedia, "Precision ML", [En línea]. Available: https://en.wikipedia.org/wiki/Precision_and_recall.

[7] Wikipedia, "Revolución Industrial etapa cuatro", [En línea]. Available: https://es.wikipedia.org/wiki/Revolución_industrial_etapa_cuatro.

[8] Wikipedia, "Test de Turing", [En línea]. Available: https://es.wikipedia.org/wiki/Prueba_de_Turing#Versiones_de_la_prueba_de_Turing.

[9] MeMorándum multimedia, "Visión Artificial para Industria: Inspección de producto en línea, control dimensional y detección de ausencia-presencia de componentes", [En línea]. Available: https://memorandum.net/vision-artificial/industria?gclid=Cj0KCQiA_c-OBhDFARIsAIFg3ez65wU-DV9pr4xCw7lfr039k-AUYL3p1_LsZKCBPnwkn07CC7RezqUaAsaOEALw_wcB.

[10] Instituto tecnologico de Informática, "HELPSALUD: Investigación aplicada al sector de salud mediante técnicas de Machine Learning desplegadas en plataforma de BDA", 2017. [En línea]. Available: https://www.iti.es/proyectosidi/helpsalud-machine-learning-salud/.

[11] S. Hansen, "Aplicación del aprendizaje automático al análisis económico y la formulación de políticas", *funcas*, 2018.

[12] I. d. l. A. Pascual, J. Ramírez y A. Ortiz, "Métodos de inteligencia artificial para la predicción del rendimiento y calidad de gramíneas", *REDVET*, vol. 17, nº 12, 2016.

[13] J. Cisneros, "LA INTELIGENCIA ARTIFICIAL (IA) EN LA LOGÍSTICA 4.0.", *Datadec*, 2020.

[14] J. H. S. Azuela y A. P. Ayala, "ESTADO DEL ARTE EN INTELIGENCIA ARTIFICIAL Y CIENCIA DE DATOS", 2019.

[15] M. Á. Gracia, "MANTENIMIENTO PREDICTIVO MEDIANTE INTELIGENCIA ARTIFICIAL Y ALGORITMOS DE DEEP LEARNING", *ITA INNOVA*, 2020.

[16] A. INNOVATION, "La Visión Artificial", 2020. [En línea]. Available: https://www.atriainnovation.com/vision-artificial-ventajas-aplicaciones/.

[17] C. A. Escobar y R. Morales-Menendez, "Machine learning techniques for quality control in high conformance manufacturing environment", *SAGE journals*, 2018.

[18] iic, “Machine Learning y optimización para la gestión del stock”, *Instituto de ingeniería del conocimiento (iic).*

[19] F. J. Villar Freire, “Detección de anomalías de red mediante técnicas de machine learning”, *Repositorio Universidade Coruña (ruc),* 2019.

[20] G. Pajares, V. Moreno y J. M. d. l. Cruz, “Clasificación de texturas mediante redes neuronales”, *CEDEX,* 2001.

[21] J. A. Triñanes, J. Torres, A. Tobar y C. Hernández, “Clasificación de imágenes multiespectrales mediante redes neuronales”, *Revista de Teledetección,* 1994.

[22] J. E. R. Rodríguez, “Redes neuronales artificiales para la clasificación de imágenes satelitales”, *Avances Investigación en Ingeniería,* 2008.

[23] P. Loncomilla, “Deep learning: Redes convolucionales”, [En línea]. Available: https://ccc.inaoep.mx/~pgomez/deep/presentations/2016Loncomilla.pdf.

[24] J. A. Serra y Jean, “Segmentación de Imágenes en Color utilizando Histogramas Bi–Variables en Espacios Color Polares Luminancia/Saturación/Matiz”, *SCIELO,* 2005.

[25] O. Ronneberger, P. Fischer y T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, 2015.

[26] W. Wan, A. Walsman y D. Fox, “Part Segmentation for Highly Accurate Deformable Tracking in Occlusions via Fully Convolutional Neural Networks”, *IEEE Xplore,* 2019.

[27] T. Takikawa, D. Acuna, V. Jampani y S. Fidler, “Gated-SCNN: Gated Shape CNNs for Semantic Segmentation”, *Cornell University,* 2019.

[28] M. J. Shafiee, B. Chywl, F. Li y A. Wong, “Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video”, 2017.

[29] T. Danka, “How to accelerate and compress neural networks with quantization”, *towards data science,* 2020.

[30] D. Banik, A. Ekbal y P. Bhattacharyya, “Machine Learning Based Optimized Pruning Approach for Decoding in Statistical Machine Translation”, *IEEE Xplore,* 2018.

[31] P. Ganesh, “Knowledge Distillation : Simplified”, *towards data science,* 2019.

[32] Wikipedia, “GPU”, [En línea]. Available: https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gráfico.

[33] Wikipedia, “Unidad de procesamiento gráfico”, [En línea]. Available: https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gráfico.

[34] NVIDIA, cKIT PARA DESARROLLADORES JETSON NANO 2 GB”, [En línea]. Available: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/education-projects/.

[35] Google, “Google”, [En línea]. Available: https://about.google/intl/es-419/.

[36] Google, “Coral”, [En línea]. Available: https://coral.ai.

[37] Wikipedia, “Unidad de procesamiento tensorial (TPU)”, [En línea]. Available: https://es.wikipedia.org/wiki/Unidad_de_procesamiento_tensorial.

[38] Google, “Google Cloud Edge TPU”, [En línea]. Available: https://cloud.google.com/edge-tpu.

[39] Google, “Google Cloud TPU de Cloud”, [En línea]. Available: https://cloud.google.com/tpu.

[40] Google, “Colab Research”, [En línea]. Available: https://colab.research.google.com.

[41] V. Meel, “Google Coral for Computer Vision Applications in 2022”, 2021. [En línea]. Available: https://viso.ai/edge-ai/google-coral/.

[42] Intel, “Intel”, [En línea]. Available: https://www.intel.es/content/www/es/es/homepage.html.

[43] Intel, “Unidades de procesamiento de visión Intel® Movidius™”, [En línea]. Available: https://www.intel.es/content/www/es/es/products/details/processors/movidius-vpu.html.

[44] Wikipedia, “Vision processing unit (VPU)”, [En línea]. Available: https://en.wikipedia.org/wiki/Vision_processing_unit.

[45] T. Danka, “How to accelerate and compress neural networks with quantization”, *Towards Data Science.*

[46] Wikipedia, “IEEE 754”, [En línea]. Available: https://es.wikipedia.org/wiki/IEEE_754.

[47] V. Nandwani, “Inside Quantization Aware Training”, *Towards Data Science,* 2021.

[48] TensorFlow, “Transfer learning and fine-tunning”, [En línea]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning.

[49] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam y D. Kalenichenko, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”, *CVPR* , 2018.

[50] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto y H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, 2017.

[51] K. He, X. Zhang, S. Ren y J. Sun, “Deep Residual Learning for Image Recognition”, 2015.

[52] A. Y. Çiğdem, “A Comprehensive Guide to Image Processing: Part 3”, 2021. [En línea]. Available: https://towardsdatascience.com/image-processing-part-3-dbf103622909.

[53] Universidad de Murcia, “Técnicas de filtrado”, [En línea]. Available: https://www.um.es/geograf/sigmur/teledet/tema06.pdf.

[54] Coral, “USB Accelerator”, [En línea]. Available: https://coral.ai/products/accelerator/.

[55] Wikipedia, “Circuito integrado de aplicación específica”, [En línea]. Available: https://es.wikipedia.org/wiki/Circuito_integrado_de_aplicación_espec%C3%ADfica.

[56] Wikipedia, “Unidad aritmética lógica”, [En línea]. Available: https://es.wikipedia.org/wiki/Unidad_aritmética_lógica.

[57] Wikipedia, “Branch Predictor (Predictor de Saltos)”, [En línea]. Available: https://en.wikipedia.org/wiki/Branch_predictor.

[58] S. Callewaert, “Google’s Edge TPU. What? How? Why?”, *ML6,* 2019.

[59] Wikipedia, “SIMD”, [En línea]. Available: https://en.wikipedia.org/wiki/SIMD.

[60] Wikipedia, “Systolic Array”, [En línea]. Available: https://en.wikipedia.org/wiki/Systolic_array.

[61] TensorFlow, “Por qué TensorFlow”, [En línea]. Available: https://www.tensorflow.org/?hl=es-419.

[62] PyTorch, “From Research to Production”, [En línea]. Available: https://pytorch.org.

[63] Scikit Learn, “scikit-learn Machine Learning in Python”, [En línea]. Available: https://scikit-learn.org/stable/.

[64] APACHE Spark 3.2.0, [En línea]. Available: https://spark.apache.org/docs/latest/ml-guide.html.

[65] TensorFlow, "TensorFLow Lite", [En línea]. Available: https://www.tensorflow.org/lite?hl=es-419.

[66] Wikipedia, "F-Score", [En línea]. Available: https://en.wikipedia.org/wiki/F-score.

[67] Wikipedia, "Media armónica", [En línea]. Available: https://es.wikipedia.org/wiki/Media_armónica.

[68] InteractiveChaos making things simple, "Exhaustividad", [En línea]. Available: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/exhaustividad.

[69] Wikipedia, "Coeficiente de Sorensen-Dice", [En línea]. Available: https://es.wikipedia.org/wiki/Coeficiente_de_Sorensen-Dice.

[70] Coral, "Edge TPU Compiler", [En línea]. Available: https://coral.ai/docs/edgetpu/compiler/#system-requirements.

[71] Seaborn, "JoinPlot", [En línea]. Available: https://seaborn.pydata.org/examples/hexbin_marginals.html.

[72] S. Sterckval, "Google Coral Edge TPU vs NVIDIA Jetson Nano: A quick deep dive into EdgeAI performance", *Medium*, 2019.

[73] Wikipedia, "Field-programmable gate array", [En línea]. Available: https://es.wikipedia.org/wiki/Field-programmable_gate_array.

[74] Wikipedia, "Digital signal processor", [En línea]. Available: https://en.wikipedia.org/wiki/Digital_signal_processor.

[75] ODSC Community, "What is Pruning in Machine Learning?", *ODSC*, 2020.

[76] P. Ganesh, "Knowledge Distillation : Simplified", *Towards data science*, 2019.

[77] R. Bandaru, "Pruning Neural Networks", *Towards data science*, 2020.

[78] J. Martínez, "Redes Neuronales Convolucionales en Profundidad", *DATASMARTS*, 2018.

[79] Wikipedia, "Red Neuronal Artificial", [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_artificial.

[80] Z. Zhang, X. Lu, G. Cao, Y. Yang, L. Jiao y F. Liu, "ViT-YOLO:Transformer-Based YOLO for Object Detection", *CVF*, 2021.

[81] D. Calvo, "Red Neuronal Convolucional CNN", [En línea]. Available: https://www.diegocalvo.es/red-neuronal-convolucional/.