UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

FACENAS: BÚSQUEDA DE ARQUITECTURA NEURONAL PARA DETECCIÓN DE ROSTROS

TÉSIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

HANS KONRAD HAROLD STARKE DÍAZ

PROFESOR GUÍA:

JAVIER RUIZ DEL SOLAR

MIEMBROS DE LA COMISIÓN:

CÉSAR AZURDIA MEZA

JOSÉ FRANCISCO DELPIANO

SANTIAGO DE CHILE

2023

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCION ELÉCTRICA y MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: HANS KONRAD HAROLD STARKE DÍAZ
FECHA: 2023
PROF. GUÍA: JAVIER RUIZ DEL SOLAR

FACENAS: BÚSQUEDA DE ARQUITECTURA NEURONAL PARA DETECCIÓN DE ROSTROS

La Búsqueda de Arquitectura Neuronal (NAS) ha adquirido relevancia en el área de clasificación de imágenes debido a su capacidad para impulsar los avances, permitiendo encontrar redes más precisas y eficientes. Este método, generalmente aplicado en la búsqueda de redes neuronales para clasificación, no obtienen resultados similares entre su tarea de clasificación y detección, pudiendo tener buen desempeño en una de estas tareas y mal desempeño en la otra. En este contexto, esta tesis se enfoca en crear un método de búsqueda de arquitectura NAS directamente enfocada en la detección, particularmente en la detección de rostros. Este nuevo método permite trabajar y adaptarse a diversas restricciones, como operaciones de punto flotante por segundos (FLOPs) o la latencia de la red en diferentes dispositivos, según las necesidades del usuario. La red neuronal resultante se evalúa utilizando la base de datos de rostros llamado WIDER FACE, lo que permite comparar esta nueva red encontrada con diversas redes convolucionales pertenecientes al estado del arte en esta tarea específica. Como resultado de este nuevo método, se logran obtener redes neuronales con mejor precisión y competir con el estado el arte, incluso incrementando la velocidad en hasta 1.63 veces.

A mi yo del futuro, que aprecie el esfuerzo del Hans del presente para que siga mejorando y madurando en la vida.

Agradecimientos

En primer lugar, quiero agradecer a mis abuelos, Patricio y Francisca (conocida como Mami), por inculcarme valores que me formaron y me han guiado en mi vida para convertirme en una mejor persona. Gran parte de lo que soy se lo debo a ellos, y estoy enormemente agradecido por su presencia constante en todas las etapas de mi vida.

Agradezco a mi padre, quien, aunque quizás no tenga un don en el fútbol, si lo tiene para demostrar un amor incondicional por su familia y me ha brindado un ejemplo valioso de cómo debo serlo en el futuro.

A mi tía Ángela, por su humildad y amabilidad que siempre la han caracterizado y que han dejado una huella positiva en mi vida.

A mi madre, por sus consejos y experiencia de saber cómo afrontar diversos problemas en la vida.

Agradezco enormemente al profesor Javier Ruiz del Solar, por brindarme las herramientas y el apoyo necesario para poder aprender, investigar y comprender cosas que nunca imaginé que podría hacer. Su mentoría ha sido muy especial.

A don Tito, quien demostró que los gatos pueden ganarse un lugar en mi corazón y, en tiempos difíciles, aliviar mi estrés con su simple presencia, tan solo a cambio de unas cuantas galletas o churus.

A todas las personas que compartieron conmigo en Makalu, y en particular a mi hermano, don Bastián, cuyas experiencias compartidas en Pichilemu y Makalu me hicieron envejecer más rápido, pero con satisfacción de los buenos momentos y risas. Agradecimiento de igual manera a mi hermana Daniela por ayudarme a cuidar a don Bastián.

Finalmente, quiero expresar mi gratitud a todas las personas que han brindado su apoyo y compañía a lo largo de mi camino. Aprecio enormemente su buena voluntad y su presencia, ya que todos ustedes son una parte fundamental de este viaje. Sus contribuciones no solo han enriquecido mi tesis, sino que también han dejado una huella en mi vida.

Tabla de contenido

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

Las redes convolucionales han producido un cambio fundamental en cómo se extrae información de las imágenes, y con el tiempo, se han mejorado mediante nuevos diseños o arquitecturas de redes neuronales. Sin embargo, durante los últimos años, las nuevas redes neuronales convolucionales han disminuido su impacto positivo en la precisión, al no lograr una mejora significativa frente a bases de datos como ImageNet [Deng et al., 2009] y COCO [Lin et al., 2014], las cuales son utilizadas como punto de comparación o benchmark. Estas redes neuronales también se han visto optimizadas para acercar su funcionamiento al tiempo real, reduciendo su tamaño y tiempo de inferencia, sin embargo, estas mejoras también se fueron volviendo cada vez menos significativas. De esta manera se da lugar al concepto de Búsqueda de Arquitectura Neuronal (NAS), una metodología de diseño automático o semi-automático que ha generado avances significativos en precisión, tiempo de inferencia, y parámetros de la red en el área de clasificación. No obstante, estas investigaciones se han centrado principalmente en el problema de clasificación, por lo que su aplicación en la detección de imágenes no ha sido directa y no genera la misma mejora de precisión, como se ha demostrado en trabajos anteriores [Bochkovskiy et al., 2020].

En esta tesis, se propone emplear los nuevos avances en NAS y en el área de las redes convolucionales para diseñar una red neuronal especializada en detección de rostros obtenido mediante un nuevo método NAS propuesto que se adapta a dispositivos específicos. Utilizando la base de datos WIDER FACE [Yang et al., 2016] como punto de referencia, se comparará la solución propuesta con modelos de detección de rostros existentes. Además, se considerará la adaptabilidad del método frente a las restricciones del dispositivo objetivo, permitiendo ajustar la velocidad de inferencia objetivo u otros parámetros para el dispositivo, creando así un enfoque versátil y ajustable a las necesidades del usuario.

La propuesta de tesis incorpora atributos de información a cada bloque básico perteneciente al espacio de búsqueda, facilitando la discriminación entre estos bloques y permitiendo que el método propuesto identifique la arquitectura de la red y maximice su precisión manteniendo una latencia cercana al valor de referencia definido por el usuario. De esta manera se crea un enfoque innovador que difiere con los métodos existentes centrados en los problemas de clasificación, ofreciendo mejoras tanto en precisión como en latencia. Este nuevo método tiene potencial para obtener resultados superiores en detección de rostros frente a los métodos NAS actuales, incorporando una nueva manera de abordar la creación, búsqueda y diseño de estas nuevas arquitecturas neuronales especializadas para la detección de rostros en dispositivos específicos.

1.1 Hipótesis

La obtención de arquitecturas de redes neuronales a través del enfoque de Búsqueda de Arquitectura Neuronal (NAS), utilizando una base de datos de detección de rostros, resultará en modelos más eficientes y precisos para la detección de rostros que trabajar con una base de datos de clasificación como se realiza generalmente. Además, se pretende demostrar que al exportar atributos de bloques básicos permitirá la adaptación de la red neuronal a un dispositivo particular, independientemente de si la optimización NAS se lleva a cabo en un hardware más potente.

1.2 Objetivos

1.2.1 Objetivo Principal

El objetivo principal de esta tesis es diseñar un método NAS enfocado directamente en la detección de rostros. Para lograr esto, se modificará el espacio de búsqueda, optimizándolo para detección y adaptándolo a diferentes restricciones. Estas restricciones pueden variar entre las operaciones de puntos flotantes por segundo (FLOPs), cantidad de activaciones (Acts) o utilizar la latencia de cada bloque en el dispositivo computacional específico.

1.2.2 Objetivos Específicos

Los objetivos específicos de esta tesis se pueden descomponer de la siguiente manera:

- Desarrollar un enfoque de método NAS orientado específicamente a la detección de rostros, contrario al enfoque convencional que se centra en las bases de datos de clasificación
- Adaptar el espacio de búsqueda en el método NAS, incorporando bloques básicos pertenecientes a redes convolucionales de clasificación que se encuentren en el estado del arte (SoA).
- Incorporar en el método NAS restricciones ajustables basadas en características como FLOPs, Acts y latencia, permitiendo una mayor flexibilidad y control de la red neuronal que se busque obtener.
- Crear un procedimiento para asignar atributos a cada bloque básico durante la ejecución del método NAS, con el fin de satisfacer las restricciones impuestas por el usuario.
- Diseñar una estrategia para extraer atributos de latencia de los bloques básicos, especialmente para dispositivos embebidos o de bajos recursos computacionales donde la ejecución completa del método NAS no sea factible.
- Generar redes neuronales para distintos dispositivos computacionales y evaluar su eficiencia respecto a otros modelos neuronales existentes.
- Validar los resultados obtenidos, considerando tanto la precisión en la base de datos WIDER FACE [Yang et al., 2016] como el rendimiento en el dispositivo computacional objetivo.

1.3 Estructura

Esta tesis se encuentra estructurado de la siguiente manera:

Capítulo 2: Marco Teórico

Este capítulo describe los trabajos e investigaciones realizados en el área de Deep Learning y NAS que proporcionan una base teórica y estructural para la investigación llevada a cabo. A grandes rasgos se compone de distintos tipos de estructuras convolucionales básicas, así como variaciones o complementos de estas estructuras que en su conjunto componen a modelos convolucionales ampliamente conocidos en el área de Visión Computacional. Además, se explican arquitecturas de detección y su funcionamiento utilizando estos modelos convolucionales de clasificación como extractor de características y, por otro lado, se describen varios trabajos NAS existentes que abordan el objetivo de optimizar modelos convolucionales en su precisión y coste computacional, pero en este caso solo acotados a realizarlo sobre base de datos de clasificación, y finalmente se describe la base de datos a utilizar para esta tesis en conjunto a sus características y composición.

Capítulo 3: Metodología

En este capítulo se presenta el trabajo base realizado en conjunto a la forma en que se implementa y desarrolla la idea y los objetivos de esta tesis. Se explica el funcionamiento del método NAS creado y se detallan los tipos de experimentos por realizar en base a lo desarrollado. Además, se describe la selección de los hardware objetivos a utilizar, explicando las razones detrás de su elección y su diferenciación frente a los otros.

Capítulo 4: Experimentos, Resultados y Análisis

Expone todos los experimentos realizados, sus resultados y las iteraciones hechas para cada uno de los hardware seleccionados. Para cada resultado obtenido se realiza un análisis, y en base a este análisis los pasos a seguir para cada iteración. También se realiza una comparación entre los resultados actuales y los obtenidos con este método NAS.

Capítulo 5: Conclusiones y trabajo a futuro

Este capítulo menciona las conclusiones alcanzadas en relación con los objetivos e hipótesis planteados en un comienzo de la investigación. Se explica si los resultados obtenidos cumplen con lo esperado y el por qué se obtienen estos resultados. Finalmente, considerando la estructura de este desarrollo e investigaciones actuales en el área, los próximos pasos a considerar para esta tesis y que de esta manera sustente una base eficiente y funcional para futuras investigaciones.

Capítulo 2

Marco Teórico

El área de Búsqueda de Arquitectura Neuronal (NAS) ha sido un área sumamente potenciada en los últimos años. La creación de nuevas redes convolucionales a partir de estas técnicas ha logrado importantes avances para nuevas arquitecturas de visión computacional.

En gran parte de estos métodos NAS, es necesario contar con un espacio de búsqueda que contenga los distintos bloques básicos. Los diseños de estos bloques básicos son obtenidos de redes convolucionales ampliamente conocidas en el área de investigación como son la ResNet [He et al., 2016], la familia de las MobileNet [Howard et al., 2017, Howard et al., 2019, Sandler et al., 2018], la red convolucional DarkNet [Redmon & Farhadi, 2018], entre otras. Para entender mejor estos bloques básicos utilizados en el espacio de búsqueda, se explicará en el siguiente punto el funcionamiento de cada uno de ellos y las variaciones posibles a realizarse sobre estos.

2.1 Tipos de Redes Convolucionales y Módulos

A continuación, se detallarán los tipos de redes neuronales existentes de mayor importancia o relevancia en conjunto a sus bloques o módulos. Posterior a esto, se presentarán y explicarán distintas variaciones que se pueden realizar sobre estos módulos para mejorar su precisión y hacerlos más eficientes.

2.1.1 ResNet, Deep Residual Learning for Image Recognition

La arquitectura de red neuronal ResNet se diseñó con el propósito de solucionar el problema que surgía al entrenar redes neuronales convolucionales de gran profundidad. Esta problemática ocurre cuando se aplican retropropagaciones de gradientes en redes que contienen una gran cantidad de bloques convolucionales. A consecuencia de esto, el rendimiento de estas redes más profundas empeoraba en vez de mejorar. La raíz de este problema radica en que los gradientes a medida que se propagan a través de la red neuronal disminuyen progresivamente en magnitud, limitando así su capacidad de influir y modificar eficientemente los pesos de los bloques más profundos.

La ResNet soluciona este problema mediante la introducción de conexiones residuales en cada bloque consecutivo, como se representa visualmente en la Figura 1. Estas conexiones residuales permiten que la información fluya de manera directa desde las capas anteriores hasta las posteriores, evitando así la disminución en gran magnitud de los gradientes. Como resultado, se logra una correcta preservación de los gradientes a lo largo de la red, lo que permite un entrenamiento y retropropagación más eficiente, permitiendo el uso de arquitecturas neuronales de mayor profundidad con un rendimiento superior.

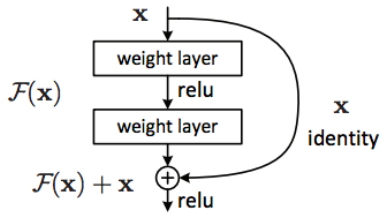

Figura 1: Bloque Residual. Imagen de [He et al., 2016].

Otra mejora que implementa la ResNet, es la modificación de los bloques convolucionales comúnmente utilizados hasta ese momento. Estos bloques convolucionales se componían simplemente de una capa de convolución. Sin embargo, estos son costosos computacionalmente, por lo que diseñan el bloque BottleNeck, una innovadora estructura que tiene como objetivo reducir la carga computacional. Este nuevo diseño se basa en la idea de que gran parte de las características en una imagen residen en espacios de baja dimensión.

El bloque BottleNeck consiste en tres capas de convolución: una capa de convolución de kernel 1x1, una de kernel 3x3 y otra de kernel 1x1. La nomenclatura BottleNeck se origina en su función de disminuir la cantidad de canales en la primera convolución 1x1 (generalmente se reduce el valor del canal de entrada a la cuarta parte). Esto posibilita que la convolución siguiente opere sobre una entrada de menor cantidad de canales. Finalmente, en la última convolución de 1x1, se restaura el canal de entrada principal. Este esquema se ilustra en la Figura 2 para mayor claridad.

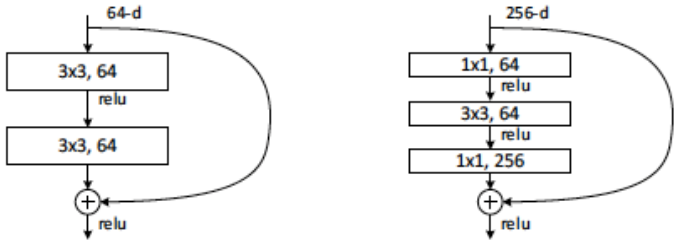

Figura 2: Comparación de BottleNeck frente a un bloque normal. Imagen de [He et al., 2016].

La ResNet fue creada en el año 2015 y sigue siendo ampliamente utilizada en numerosos trabajos de investigación y aplicaciones. A lo largo de los últimos años, ha experimentado mejoras sustanciales mediante la modificación de su método de entrenamiento, como se menciona en [Wightman et al., 2021]. Estas mejoras han permitido que la ResNet se mantenga vigente y superando a redes convolucionales creadas en los últimos años. La popularidad que esta red ha tenido ha tenido se debe principalmente a su buen rendimiento en una variedad de campos de visión computacional. Se ha convertido en un estándar de referencia, facilitando comparaciones

entre distintos trabajos de investigación al proporcionar un punto común de comparación. Incluso sus componentes fundamentales, como los bloques BottleNeck y las conexiones residuales, han sido incorporados y adaptados en una gran cantidad de trabajos posteriores a la creación de la ResNet.

La familia de arquitecturas ResNet presenta diversas variantes que se distinguen por la cantidad de capas que las componen, siendo la ResNet50 la más utilizada y estándar. La estructura de la ResNet50 comienza con una convolución inicial de 3 canales de entrada y 64 canales de salida, utilizando un kernel 7x7 con un stride de 2. Posteriormente, se aplica una operación MaxPool con un kernel de 3x3 y un stride de 2, lo que contribuye a reducir la dimensionalidad de la información. Con esta última operación de MaxPool se introduce una invarianza ante traslaciones y destaca las características más significativas de cada región del filtro. Sin embargo, es fundamental considerar que esta operación puede llevar a la pérdida de detalles menos relevantes, lo que podría impactar la precisión en la detección de objetos específicos.

Tras esta etapa inicial, la estructura de la ResNet se divide en 4 etapas. Cada variante de la familia ResNet presenta un número diferente de capas en cada etapa, lo que contribuye a su configuración única. El desglose detallado de cada una de estas variantes se puede observar en la Tabla 1.

Tabla 1: Diseño y arquitectura interna de las ResNet. Tabla de [He et al., 2016].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Como se puede apreciar, la ResNet50 se compone de 16 bloques de BottleNeck, cada uno como se menciona anteriormente se compone de 3 capas convolucionales, llegando a ser un total de 48 convoluciones, más las dos capas iniciales (convolución inicial de 7x7 y el posterior MaxPool) se llega a las 50 capas, las que corresponde a su nombre ResNet50.

2.1.2 Bag of Tricks for Image Classification with Convolutional Neural Networks

Existe una gran cantidad de investigaciones que utilizan como base la arquitectura neuronal ResNet [He et al., 2016], en particular la ResNet50. Por esta razón, el trabajo de investigación titulado Bag of Tricks for Image Classification with Convolutional Neural busca mejorar la arquitectura a través de pequeñas modificaciones y mejoras específicas sobre la ResNet [He et al., 2016], centrándose para las comparaciones en la ResNet50.

Tal como se señala en el nombre de esta investigación, se exploran múltiples trucos en la ResNet50 con el propósito de mejorar su precisión. Estas mejoras abarcan diversos aspectos, desde la metodología de entrenamiento de la red hasta la configuración de hiperparámetros. También se llevan a cabo modificaciones en los módulos internos de la ResNet [He et al., 2016]. Estos cambios de estructura se detallarán en esta sección, ya que presentan mejoras de relevancia y aplicabilidad en el contexto de este trabajo, FaceNAS.

Las modificaciones estructurales implementadas en la ResNet [He et al., 2016] se desglosan en dos componentes principales, siendo el primero de ellos una modificación en la arquitectura de entrada de la ResNet [He et al., 2016], como se puede observar en la Figura 3.

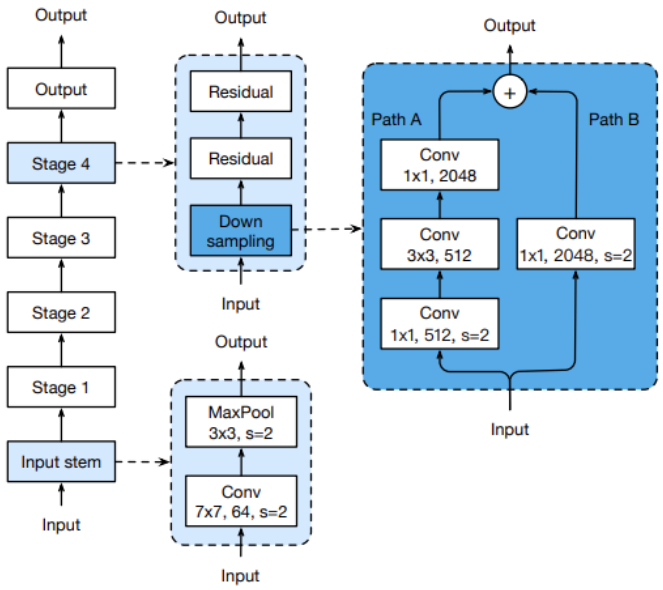

Figura 3: Mejoras internas de la arquitectura de la ResNet. Imagen de [He et al., 2018].

La arquitectura de entrada (Input stem) se construye con una convolución que tiene 3 canales de entrada y 64 canales de salida, utilizando un kernel de 7x7 con un stride 2. Posteriormente, se aplica una operación de MaxPool de 3x3 con stride 2 para nuevamente reducir la dimensionalidad. En conjunto, este bloque de Input stem reduce

la dimensión del ancho y alto en un factor de 4, esto debido a que tanto el MaxPool como la convolución inicial tienen un stride de 2.

El artículo observa que una convolución con un kernel de 7x7 es 5.4 veces más costosa computacionalmente que una convolución de 3x3. Por lo tanto, se procede a modificar esta convolución de 7x7 reemplazándola por tres convoluciones de 3x3, cada una con 32 canales, y la última con una salida de 64 canales. Esta adaptación transforma el Input stem desde su arquitectura inicial a la que se muestra en la Figura 4. Esta modificación recibe el nombre de ResNet-C.

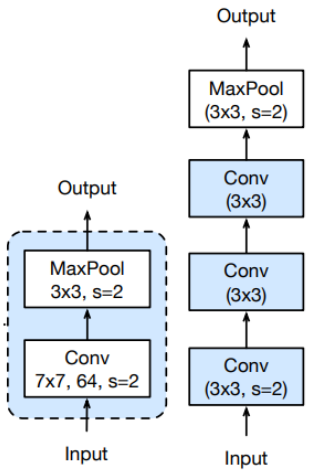

Figura 4: Entrada original de ResNet y su entrada en la variación ResNet-C. Imagen de [He et al., 2018].

El segundo componente principal de modificación es llevado a cabo en el proceso de Submuestreo (DownSampling) de la ResNet [He et al., 2016] y se descompone en dos partes. En primer lugar, este artículo destaca que al establecer un stride de 2 en la primera convolución 1x1 del bloque BottleNeck, se excluye el 75% de la entrada del mapa de características debido al uso de un kernel de 1x1 con un stride de 2. Con el propósito de arreglar este problema, el stride de 2 se traslada a la siguiente convolución que forma parte del BottleNeck. Esta convolución emplea un kernel de 3x3, asegurando que no se pierda ni omita información en el proceso. Esta modificación sobre la ResNet [He et al., 2016] se denomina ResNet-B se ilustra en la Figura 5.

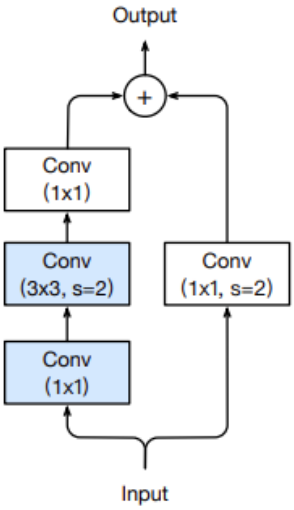

Figura 5: Modificación en el Submuestreo o DownSampling de la ResNet, variación ResNet-B. Imagen de [He et al., 2018].

Otra consideración clave en la fase de Submuestreo es abordar la misma pérdida u omisión de características que surge al emplear una convolución de kernel 1x1 y stride de 2 en la conexión residual del bloque. Para solventar esta situación, se implementa una nueva estructura que implica modificar esta convolución de kernel 1x1 y stride 2, por una operación de Average Pooling con un kernel de 2x2 y un stride de 2, y posterior a esta operación incorporar la convolución de kernel 1x1, la cual ahora se realiza con stride de 1. Esta adaptación en la estructura de la Resnet [He et al., 2016] recibe el nombre de ResNet-D. La representación visual de esta modificación se aprecia en la Figura 6.

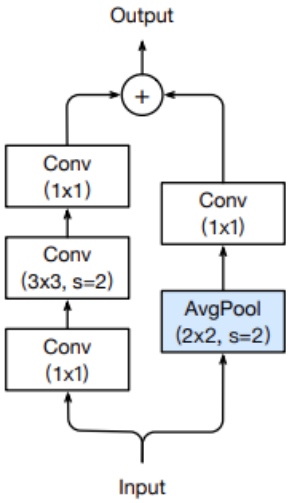

Figura 6: Modificación en el submuestreo de la ResNet, variación ResNet-D. Imagen de [He et al., 2018].

Cada una de estas modificaciones conlleva mejoras en la precisión de la red ResNet [He et al., 2016], aunque implica un ligero aumento en el tiempo de latencia según indica

el artículo. Para estas nuevas variaciones se realiza su entrenamiento en 120 épocas con 8 GPUs Nvidia V100. Para el caso de la ResNet50-D, que es la que genera el mayor costo computacional, es decir, conlleva una mayor cantidad de operaciones de punto flotante por segundo (FLOPs) de las tres variantes, solo genera una reducción del 3% en el tiempo de entrenamiento en comparación con el modelo ResNet [He et al., 2016] básico. Los resultados detallados y el análisis de la precisión y los parámetros se presentan en la Tabla 2.

Tabla 2: Resumen parámetros, precisión de ResNet y sus variaciones. Tabla de [He et al., 2018].

Model	#params	FLOPs	Top-1	Top-5
ResNet-50	25 M	**3.8 G**	76.21	92.97
ResNet-50-B	25 M	4.1 G	76.66	93.28
ResNet-50-C	25 M	4.3 G	76.87	93.48
ResNet-50-D	25 M	4.3 G	**77.16**	**93.52**

2.1.3 TResNet

La TResNet surge a partir de la idea de que, aun cuando han surgido numerosas redes neuronales en los últimos años que han superado a la ResNet50 [He et al., 2016] en términos de precisión con una cantidad menor o similar de operaciones de punto flotante por segundo (FLOPs), la medida de FLOPs no siempre se traduce directamente en la velocidad de inferencia o entrenamiento de la red. De hecho, en términos de equilibrio entre latencia de la red y precisión, la ResNet50 [He et al., 2016], incluso con una mayor cantidad de FLOPs, puede ser significativamente más rápida que varios de sus competidores. En base a esto, el propósito de esta investigación se enfoca en efectuar ajustes y optimizaciones en la ResNet [He et al., 2016] base.

Es relevante mencionar que solo se abordarán las modificaciones que se utilizarán o que formaron parte del diseño y análisis del método aplicado en FaceNAS.

Una de las mejoras implementadas se centra en optimizar el uso de las capas Squeeze-and-Excitation (SE) [Hu et al., 2018]. En términos generales, estas capas son componentes complementarios a los bloques convolucionales, que permiten recalibrar la importancia de los canales de características. Aunque los bloques SE [Hu et al., 2018] contribuyan a la precisión global del modelo de red neuronal, también incrementa su carga computacional. Por lo tanto, en esta investigación se busca optimizar la utilización de estos componentes SE [Hu et al., 2018], empleándolos únicamente en las tres etapas iniciales de la red.

Esta decisión se justifica debido a que la última etapa opera con mapas de baja resolución, por lo que la operación de Global Average Pooling utilizada en el componente SE [Hu et al., 2018] no aporta un beneficio significativo en términos de precisión, y en

cambio introduce un costo computacional mayor que el beneficio obtenido. En la Tabla 3 detalla la arquitectura completa de las TResnet y en qué etapas son utilizados los componentes SE [Hu et al., 2018].

Tabla 3: Diseño y arquitectura interna de la TResNet. Tabla de [Ridnik et al., 2020].

Layer	Block Type	Output	Stride	TResNet					
				M		L		XL	
				Repeats	Channels	Repeats	Channels	Repeats	Channels
Stem	SpaceToDepth Conv1x1	56×56	- 1	1 1	48 64	1 1	48 76	1 1	48 84
Stage1	BasicBlock+SE	56×56	1	3	64	4	76	4	84
Stage2	BasicBlock+SE	28×28	2	4	128	5	152	5	168
Stage3	Bottleneck+SE	14×14	2	11	1024	18	1216	24	1344
Stage4	Bottleneck	7×7	2	3	2048	3	2432	3	2688
Pooling	GlobalAvgPool	1×1	1	1	2048	1	2432	1	2688
#Params.				29.4M		54.7M		77.1M	

Además de optimizar las etapas en las que se emplea el componente SE [Hu et al., 2018] en la red, se ha llevado a cabo una mejora en su ubicación interna dentro de los módulos. Para lograr esto, en los módulos BottleNeck, el componente SE [Hu et al., 2018] se sitúa después de la convolución con kernel 3x3. Esta nueva disposición busca aprovechar la compresión de la BottleNeck. En relación con el componente SE [Hu et al., 2018], se ajusta el valor interno de compresión (denominado *r*) para las SE [Hu et al., 2018] utilizadas en este bloque con un factor de 8. En otras palabras, la compresión interna de las SE [Hu et al., 2018] reduce los canales de entrada en 8 veces antes de restaurarlos a la cantidad original previa a la salida. Por otro lado, en los bloques básicos de convolución (identificados como BasicBlock en la Tabla 3), el componente SE [Hu et al., 2018] se implementa justo antes de la suma residual, con un factor de reducción 4.

La elección de estos diferentes factores internos de reducción se fundamenta en la composición de la TResNet. Los bloques básicos de convolución son empleados en las primeras etapas de la red, donde la cantidad de canales es menor. Como resultado, un factor de compresión muy alto podría sobrecomprimir la información y limitar la eficiencia de este proceso y, en consecuencia, afectar negativamente en la precisión.

Para una comprensión más detallada de estas adaptaciones en relación con la ubicación del componente SE [Hu et al., 2018] y el factor interno de reducción *r*, se puede consultar la Figura 7.

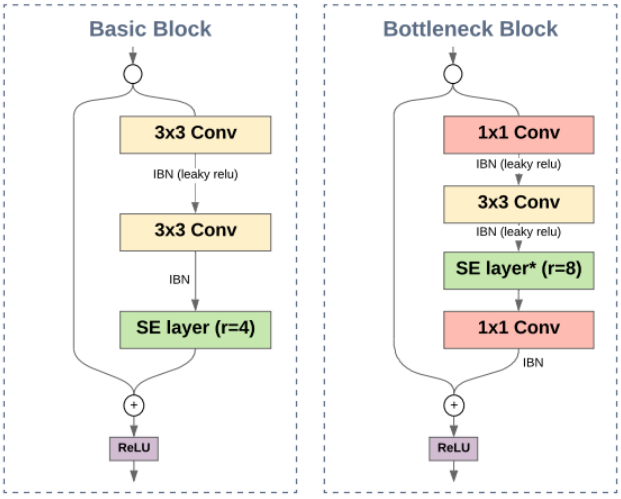

Figura 7: Implementación interna de las SE en Basic Block y BottleNeck. Imagen de [Ridnik et al., 2020].

En este artículo también se destaca que, si bien la ResNet38 [He et al., 2016] se constituye exclusivamente de bloques básicos (convoluciones directas) y la ResNet50 [He et al., 2016] utiliza únicamente BottleNeck, no es necesario que la arquitectura sea uniforme en toda su extensión. Por el contrario, es factible combinar ambos tipos de módulos y aprovechar las ventajas de cada una de ellas.

Es importante considerar que, si bien las BottleNeck ofrecen una precisión superior en comparación con los bloques básicos, en términos de relación entre precisión y costo computacional, también presentan la desventaja de consumir más recursos de GPU en comparación con los bloques básicos. En contraste, los bloques básicos poseen un campo receptivo más amplio, lo que les permite capturar más características y los posiciona de manera más efectiva que las BottleNeck en las primeras etapas de la red.

Basándose en estas consideraciones, el diseño de la TResNet incorpora bloques básicos en las dos etapas iniciales, mientras que las dos subsiguientes emplean BottleNeck. Esta decisión se complementa con el hecho de que los componentes SE [Hu et al., 2018] se utilizarán únicamente en las tres primeras etapas de la red, con el propósito de optimizar su uso.

2.1.4 YOLOV3

El impacto del trabajo realizado en Yolov3 supuso una revolución en el campo de la detección de objetos en tiempo real mediante redes convolucionales. Para este proyecto de FaceNAS, el enfoque recae en explorar el marco teórico de este trabajo, centrándose específicamente en su red de extracción de características denominado DarkNet, que ha sido incluso utilizada en investigaciones más recientes como [Ge et al., 2021]. Esta red neuronal se caracteriza por estar compuesta por bloques conformados por dos convoluciones básicas: la primera utiliza un kernel de 1x1, seguida por una segunda convolución con un kernel de 3x3 que finalmente conecta con una conexión residual.

Estos módulos son parte integral de la arquitectura DarkNet53 que se encuentra detallada en la Figura 8.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 8: Arquitectura interna de la Darknet53. Imagen de [Redmon & Farhadi, 2018].

La DarkNet emerge como un competidor directo de la ResNet [He et al., 2016] en el ámbito de la extracción de características para arquitecturas de detección de objetos. En su diseño original, DarkNet fue creada con la finalidad de ser un extractor de características que logre equilibrar tanto la precisión como la velocidad de inferencia o latencia, permitiendo un desempeño en tiempo real. Mientras tanto, la red ResNet [He et al., 2016] se originó en un principio para problemas de clasificación. A pesar de sus diferencias iniciales de diseño, ambas se desempeñan de manera excepcional en ambas tareas. Así, existen diversas variantes de arquitecturas de detección que ofrecen la opción de emplear tanto ResNet [He et al., 2016] como DarkNet como extractores de características. Este enfoque se materializa en la red de detección Yolov4 [Bochkovskiy et al., 2020], la cual se abordará en la siguiente sección 2.1.5.

2.1.5 YOLOV4

Esta nueva versión de Yolo introduce avances significativos. Uno de los resultados obtenidos es que la red neuronal CSPResNext50 [Wang et al., 2019] (variación de la ResNet50 [He et al., 2016]) es considerada mejor que la CSPDarkNet53 [Wang et al., 2019] (variación de la DarkNet53 [Redmon & Farhadi, 2018]) en el área de clasificación de objetos en ImageNet [Deng et al., 2009]. Sin embargo, estos resultados se invierten en el área de detección de objetos, donde CSPDarknet53 [Wang et al., 2019] supera a la CSPResNext50 [Wang et al., 2019] en la base de datos MS COCO [Lin et al., 2014]. Estos resultados validan la noción planteada en YOLOV3 [Redmon & Farhadi, 2018] (vale la pena destacar que YOLOV4 y YOLOV3 [Redmon & Farhadi, 2018] son obras de

diferentes autores), donde se argumenta que los diseños iniciales de ResNet [He et al., 2016] y Darknet [Redmon & Farhadi, 2018] estaban orientados a objetivos diferentes.

De esta manera llegan a la conclusión de que no siempre una red neuronal óptima para clasificación será igualmente óptima para detección, como se observa en este caso. Se resalta que, en el contexto de la detección de objetos, se deben considerar factores clave:

- **Gran tamaño de entrada a la red:** Esto se traduce en la capacidad de detectar objetos más pequeños presentes en la imagen, a diferencia de la clasificación, donde el objeto completo es el que suele abarcar la imagen. Un ejemplo ilustrativo es la diferencia de resolución entre las redes de clasificación (usualmente 224x224 píxeles) y las de detección (640x640 píxeles).
- **Mayor cantidad de capas neuronales:** Se requiere un mayor número de capas para obtener un campo receptivo más amplio. La mayor resolución de entrada cubre mejor las características presentes en la imagen.
- **Mayor cantidad de parámetros en la red neuronal:** Esto amplía la capacidad para detectar múltiples objetos de diversos tamaños en una misma imagen. Al tener más parámetros, la red tiene mayor flexibilidad y adaptabilidad durante el entrenamiento, lo que evita la pérdida de características cruciales al no estar limitada por el número de parámetros. Esto es esencial para la detección de objetos, donde los tamaños de los objetos pueden variar y la resolución de entrada puede cambiar en distintos escenarios de inferencia. En este contexto, mencionan que, si bien aumentar la cantidad de parámetros es esencial para algunas tareas, también existe el riesgo de sobreajustar si la base de datos es pequeña o si la resolución de entrada no es adecuada para el método. Esto puede resultar en un rendimiento deficiente al no generalizar correctamente para casos distintos a los utilizados en el entrenamiento.

Además, realizan una comparación con otra familia de red neuronal ampliamente conocida en el ámbito de la clasificación: las EfficientNet [Tan & Le, 2019].

Tabla 4: Comparativa entre CSPResNext50, CSPDarknet53 y EffcientNet-B3. Tabla de [Bochkovskiy et al., 2020].

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

Basándonos en la comparativa presentada en la Tabla 4, se pueden extraer conclusiones a partir de los resultados obtenidos. En primer lugar, es evidente que la cantidad de parámetros u operaciones no se traduce directamente en la velocidad de

latencia del modelo. Se destaca que CSPDarkNet53 [Wang et al., 2019], a pesar de tener la mayor cantidad de parámetros y BFLOPs, logra operar con una velocidad de cuadros por segundo (FPS) más alta en comparación con las otras redes. Esta observación pone de manifiesto que una mayor cantidad de operaciones en las unidades de procesamiento gráfico (GPUs) no necesariamente se traduce en un aumento lineal en la velocidad de la red. Esto se debe a la capacidad de realizar muchas de estas operaciones de manera paralela, lo que optimiza el rendimiento en términos de latencia.

Como se mencionó previamente, el uso de FLOPs había sido un estándar para comparar diferentes redes neuronales y evaluar su velocidad. Sin embargo, en los últimos años se ha demostrado en varios estudios que esta métrica no refleja directamente la velocidad real de la red. Por lo tanto, la manera más precisa y directa de comparar la velocidad y la latencia entre distintas redes neuronales es a través de la medición de latencia en un hardware específico.

2.1.6 MobileNetV1

MobileNet fue un antes y un después en el desarrollo de redes neuronales con alta precisión adaptadas a dispositivos móviles y embebidos. Para lograr este objetivo, introdujo modificaciones sobre los módulos tradicionales de convolución, reemplazándolos por convoluciones separables que reducen significativamente la cantidad de parámetros y operaciones.

Este nuevo módulo se llama Depthwise Separable Convolution. Su función es aplicar una única operación de convolución en cada canal de entrada de manera individual. El módulo se compone de dos componentes internos: la Depthwise Convolution y la Pontwise Convolution.

La Depthwise Convolution consiste en realizar una única operación de convolución en cada canal de entrada de manera independiente, permitiendo capturar características espaciales de forma individualizada en cada canal.

La Pointwise Convolution es una convolución de kernel 1x1 y es aplicada al resultado de la Depthwise Convolution. Este bloque se encarga de fusionar las características obtenidas en la etapa anterior, generando una interacción entre los distintos canales y así obteniendo características más representativas y complejas.

Una comparación gráfica entre la arquitectura convencional de convolución y la implementada en MobileNet se representa en la Figura 9.

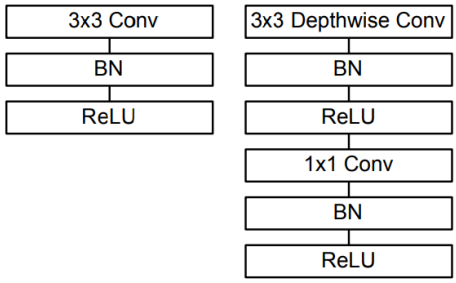

Figura 9: Comparativa entre bloque convolucional normal y una Depthwise Separable Convolution. Imagen de [Howard et al., 2017].

Realizan un experimento en el cual evaluaron la arquitectura propuesta, modificando los bloques utilizados en la red. En este experimento, se alternaron entre el uso exclusivo de convoluciones estándar y el de sólo convoluciones Depthwise Separable Convolution. Los resultados de este experimento se presentan en la Tabla 5.

Tabla 5: Comparativa entre resultados de red basada en convolución normal y en las Depthwise Separable Convolution. Tabla de [Howard et al., 2017].

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Obteniendo una baja sustancial en la cantidad de operaciones y cantidad de parámetros de la red a costa de una leve disminución en la precisión de la red neuronal.

2.1.7 MobileNetV2

Este artículo presenta una evolución con respecto a la MobileNetV1 [Howard et al., 2017], con el mismo objetivo de diseñar una red adecuada para dispositivos móviles y embebidos con recursos limitados, manteniendo al mismo tiempo la precisión de la red neuronal a un nivel alto.

Se introduce un nuevo módulo llamado Inverted Residuals with Linear BottleNecks, que se compone de tres capas de convolución. La primera capa utiliza una convolución de kernel 1x1 para realizar una expansión t (generalmente se expanden los canales de entrada en un factor de 6). Posteriormente, sigue una Depthwise Convolution, y finalmente una Pointwise Convolution (tanto la Depthwise Convolution como la Pointwise Convolution son detalladas en la sección 2.1.6) que revierte la expansión de canales realizada en la primera convolución. Este proceso se ilustra en la Tabla 6.

Tabla 6: Diseño de la Inverted Residuals with Linear BottleNecks. Tabla de [Sandler et al., 2018].

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Otra mejora que implementan, ausente en la MobileNetV1 [Howard et al., 2017], es la incorporación de conexiones residuales entre los módulos. Estas incorporaciones se aplican exclusivamente en los módulos con stride de 1, excluyendo los que cuentan con un stride de 2 debido a que son utilizados en el Downsampling o Submuestreo. La motivación detrás de esta mejora es similar a la presentada en la ResNet [He et al., 2016]: facilitar la propagación de gradientes durante el proceso de entrenamiento, atenuando el problema de degradación de gradientes que puede ocurrir en la retropropagación. Esta mejora presentada tanto para sus versiones de stride 1 y 2 se presentan gráficamente en la Figura 10.

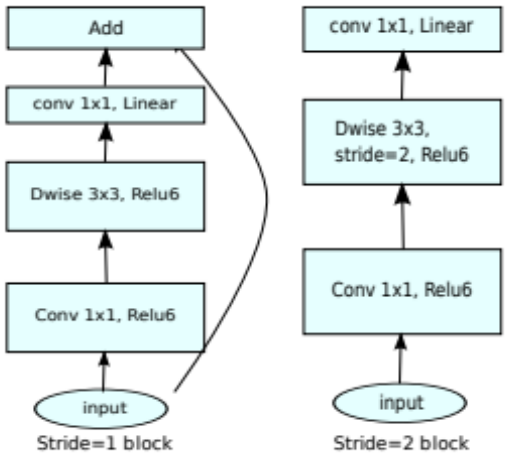

Figura 10: Uso de conexiones residuales en las Inverted Reisduals with Linear BottleNecks. Imagen de [Sandler et al., 2018].

Los resultados obtenidos para esta nueva red MobileNetV2 frente a distintas redes neuronales son los mostrados en la Tabla 7.

Tabla 7: Resultados y comparación de MobileNetV2 frente a otras redes móviles. Tabla de [Sandler et al., 2018].

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Obteniendo una disminución en la cantidad de operaciones (MAdds) y parámetros frente a sus competidores incluyendo la MobileNetV1 [Howard et al., 2017].

2.1.8 MobileNetV3

La MobileNetV3 tiene como objetivo continuar mejorando la red neuronal previa (MobileNetV2 [Sandler et al., 2018]) mediante nuevos diseños y técnicas. Entre las estrategias empleadas se incluyen ajustes en las funciones de activación, variaciones en la cantidad de capas y canales, y la implementación de NAS. Esta última técnica se emplea en conjunto con el diseño manual de humanos expertos para aprovechar ambos enfoques de manera complementaria y potenciar aún más la precisión del modelo.

La función de activación comúnmente utilizada para redes neuronales es la función ReLU (Rectified Linear Unit) [Nair & Hinton, 2010]. No obstante, según se menciona en este artículo, existe un problema asociado a su no linealidad. Cuando el valor de entrada es negativo, la salida se convierte en 0, lo que puede resultar en la pérdida de información. La fórmula de la función de activación ReLU [Nair & Hinton, 2010] se encuentra en la ecuación 1.

$$ReLU(x) = \max(0, x) \tag{1}$$

En lugar de utilizar la función de activación ReLU [Nair & Hinton, 2010], se opta por utilizar la función de activación Swish [Ramachandran et al., 2017]. La función Swish [Ramachandran et al., 2017] es más suave y diferenciable en comparación con ReLU [Nair & Hinton, 2010], lo que resulta en ventajas durante la retropropagación de gradientes durante el proceso de entrenamiento. Además, Swish [Ramachandran et al., 2017] evita el inconveniente de la pérdida de información que se puede experimentar con ReLU [Nair & Hinton, 2010] cuando los valores de entrada son negativos debido a la no linearidad de Swish [Ramachandran et al., 2017]. La fórmula que define a esta función de activación se aprecia en la ecuación 2.

$$Swish(x) = x * sigmoid(x) \quad (2)$$

Sin embargo, esta nueva función de activación conlleva un mayor costo computacional en comparación con ReLU [Nair & Hinton, 2010], cuya operación es más simple. Por ende, emplean una variante de Swish [Ramachandran et al., 2017] denominada HardSwish, la cual busca una aproximación a la función de activación Swish [Ramachandran et al., 2017] original, pero con operaciones menos complejas. Aunque HardSwish se asemeja en gran medida a Swish [Ramachandran et al., 2017], no es exactamente igual, lo que resulta en una ligera reducción del rendimiento en términos de precisión. No obstante, esta variante ayuda a mejorar la eficiencia computacional que implicaría la utilización directa de Swish [Ramachandran et al., 2017]. La función HardSwish se define mediante la operación implementada en la ecuación 3.

$$HardSwish(x) = x * \frac{ReLU6(x+3)}{6} \quad (3)$$

Observando detenidamente el cambio en la operación, se puede notar que, además de las operaciones aritméticas simples, la modificación se enfoca principalmente en reemplazar la función sigmoidal por una ReLU6 [Howard et al., 2017], que es similar a la función ReLU [Nair & Hinton, 2010] pero con un límite superior de 6. La comparación entre estas dos funciones de activación se representa gráficamente en la Figura 11.

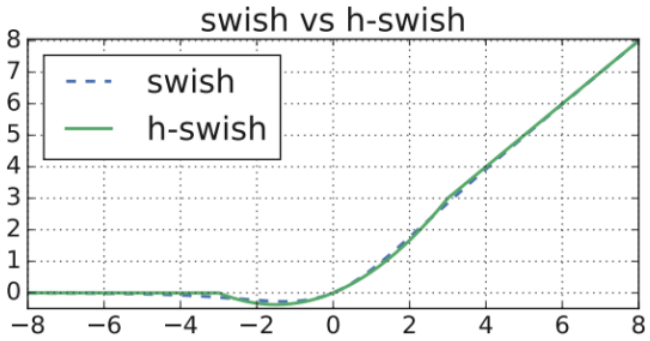

Figura 11: Gráfico y comparativa de funciones de activación Swish y HardSwish. Imagen de [Howard et al., 2019].

Dentro de los experimentos, se observa que los mayores beneficios al utilizar la nueva función de activación HardSwish se obtienen al aplicarla en las últimas etapas de la red neuronal, mientras que se utilizan funciones de activación ReLU [Nair & Hinton, 2010] para las primeras capas de la red. Los experimentos realizados para evaluar esta configuración son los mostrados en la Tabla 8.

Tabla 8: Resultados de variaciones en implementación de función de activación HardSwish. Tabla de [Howard et al., 2019].

	Top-1	P-1
V3-Large 1.0	75.2	51.4
ReLU	74.5 (-.7%)	50.5 (-1%)
h-swish @16	75.4 (+.2%)	53.5 (+4%)
h-swish @112	75.0 (-.3%)	51 (-0.5%)

En estos casos específicos, los experimentos realizados son los siguientes:

- **ReLU:** Se utiliza la función de activación ReLU [Nair & Hinton, 2010] en toda la red neuronal.
- **H-Swish@16:** Se emplea la función de activación HardSwish en la red neuronal completa para módulos con un canal de entrada inicial de 16 o mayor.
- **H-Swish@112:** Se utiliza la función de activación HardSwish únicamente en las últimas etapas de la red. Esto debido a que se implementa HardSwish sólo en los módulos donde su canal de entrada sea igual o mayor a 112.

Es importante destacar que la MobileNetV3-Large 1.0 mostrada en la Tabla 8 corresponde a utilizar funciones de activación HardSwish en los módulos donde los canales de entrada sean igual o mayor a 80, lo que es equivalente a usar su nomenclatura de h-swish@80.

En esta misma línea de mejora, un trabajo relacionado en el área, titulado Designing Network Design Spaces [Radosavovic et al., 2020], también analiza la función de activación HardSwish. Este estudio concluye que HardSwish funciona de manera más efectiva en redes más limitadas y compactas, mientras que, para redes más grandes y complejas, ReLU [Nair & Hinton, 2010] sigue siendo la opción más adecuada.

Otra mejora que se introduce es la incorporación de las capas Squeeze-and-Excitation layers (SE) [Hu et al., 2018] en los módulos que conforman la red, aunque se emplean selectivamente debido al costo computacional que conlleva su uso. La arquitectura final de la MobileNetV3 se presenta en la Tabla 9.

Tabla 9: Diseño de la MobileNetV3. Tabla de [Howard et al., 2019].

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

2.2 Variaciones a Módulos Convolucionales

Si bien dentro del espacio de búsqueda del método NAS se pueden considerar distintos tipos de bloques o módulos, también existen variaciones adicionales que se pueden aplicar sobre estos módulos. Para este trabajo de FaceNAS se exponen dos tipos de variaciones posibles a estos módulos. Una de estas variaciones es la Squeeze-and-Excitation [Hu et al., 2018], mientras que la segunda variación es la variación denominada effective Squeeze-and-Excitation (eSE) [Lee & Park, 2020] la cual fue presentada en el trabajo de CenterMask [Lee & Park, 2020]. A continuación, se proporcionarán los detalles sobre ambas variaciones.

2.2.1 SE

El bloque Squeeze-and-Excitation (SE) es una estructura diseñada para complementar módulos ya existentes en las redes neuronales. Su función radica en mejorar la capacidad de atención y la representación de características en la red.

La primera implementación de este bloque se llevó a cabo en la arquitectura de la red ResNet [He et al., 2016], donde se demostró que esta ligera modificación añadiendo el bloque SE, que implicaba un pequeño aumento en la carga computacional puede tener un impacto significativo en la precisión del modelo. El bloque SE se constituye de dos componentes principales, denominados Squeeze y Excitation:

- **Squeeze:** En esta fase, se aplica el Global Pooling de tamaño 1x1, reduciendo la dimensionalidad de los mapas de características a un vector de canales 1x1xC. El

objetivo es condensar la información, reteniendo las características clave y más relevantes.

- **Excitation:** Una vez obtenido el vector de características del componente Squeeze, se aplica una capa fully-connected (o convoluciones directas en las versiones más recientes de la SE) con una compresión de canales r (normalmente con valor 4 o 16) y es seguida por una función de activación ReLU [Nair & Hinton, 2010]. Luego, el vector de características se expande a su cantidad original de canales, *C,* mediante otra fully-connected, terminando con una función de activación Sigmoidal. Estas operaciones permiten que el bloque aprenda la contribución relativa de cada canal en relación con la red neuronal.

Una vez procesados estos dos componentes, el resultado del bloque SE se pondera mediante una multiplicación punto a punto con la entrada original del módulo donde se integra el bloque SE, modificando así la importancia de los pesos.

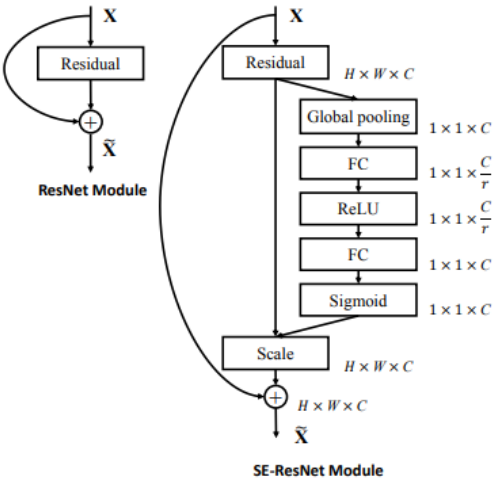

Figura 12: Implementación y diseño de la Squeeze-and-Excitation block. Imagen de [Hu et al., 2018].

La Figura 12 ilustra cómo se integra el bloque SE en una ResNet [He et al., 2016], llamándose la variación SE-ResNet. Los resultados obtenidos para diversas redes con respecto a su modificación e incorporación del bloque SE son los mostrados en la Tabla 10.

Tabla 10: Comparación de resultados de redes y su variación con Squeeze-and-Excitation block. Tabla de [Hu et al., 2018].

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [19]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [19]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [11]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [6]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [21]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

La Tabla 10 presenta una comparativa entre los resultados de diversas redes y sus variantes con el bloque SE. La inclusión de este bloque ha logrado una mejora en la tasa de error de clasificación en estas redes, con un ligero incremento en la cantidad de operaciones de punto flotante por segundo (Giga-FLOPs). Estas mejoras en la precisión se evaluaron en la base de datos de ImageNet [Deng et al., 2009].

Es importante considerar que, a medida que la red es de menor tamaño, la variación y aumento en la carga computacional será más evidente en comparación con redes más grandes, como ResNet50 [He et al., 2016], donde la variación será más marginal.

2.2.2 eSE

El bloque SE [Hu et al., 2018] introdujo mejoras notables en la precisión de redes neuronales al resaltar información relevante en los pesos. Sin embargo, surgía un inconveniente durante la etapa Excitation, donde la reducción de canales (generalmente entre 4 y 16) podía resultar en una pérdida de información debido a la compresión.

Para abordar esta limitación, se propuso una variante denominada effective SE (eSE), que se presentó en el artículo CenterMask. La principal innovación de la eSE radica en el uso de una única capa fully-connected de canales *C*, en lugar de las dos capas utilizadas en la versión original del SE [Hu et al., 2018]. Esta adaptación permite mantener la cantidad de canales de entrada sin comprimir ni expandir, evitando la pérdida de dimensionalidad.

Hay que tener en cuenta que, en aplicaciones actuales de visión computacional, esta capa fully-connected también se ha implementado en forma de convolución de filtro 1x1.

Tabla 11: Resultados y comparativa entre redes con variación SE y eSE. Tabla de [Lee & Park, 2020].

Backbone	Params.	AP^{mask}	AP^{box}	Time (ms)
VoVNetV1-39	49.0M	35.3	39.7	68
+ residual	49.0M	35.5 (+0.2)	39.8 (+0.1)	68
+ SE [13]	50.8M	34.6 (-0.7)	39.0 (-0.7)	70
+ eSE, *ours*	52.6M	35.6 (+0.3)	40.0 (+0.3)	70
VoVNetV1-57	63.0M	36.1	40.8	74
+ residual	63.0M	36.4 (+0.3)	41.1 (+0.3)	74
+ SE [13]	65.9M	35.9 (-0.2)	40.8	77
+ eSE, *ours*	68.9M	36.6 (+0.5)	41.5 (+0.7)	76
VoVNetV1-99	83.6M	31.5	35.3	101
+ residual	83.6M	37.6 (+6.1)	42.5 (+7.2)	101
+ SE [13]	88.0M	37.1 (+5.6)	41.9 (+6.6)	107
+ eSE, *ours*	96.9M	38.3 (+6.8)	43.5 (+8.2)	106

En la Tabla 11 se puede apreciar los experimentos realizados en diferentes configuraciones de redes, incluyendo la red base, la variante con bloques SE [Hu et al., 2018] y la variante con bloques eSE [Lee & Park, 2020]. Se observa una mejora sustancial en la precisión para las tres redes analizadas. Además, los resultados muestran que ambas variantes (SE [Hu et al., 2018] y eSE) presentan una latencia similar, por lo que no existe una clara ventaja de una variante sobre la otra en términos de latencia. Sin embargo, la variante eSE incorpora una mayor cantidad de parámetros en comparación con la variante original SE [Hu et al., 2018] y también muestra una mejora de precisión en cada una de las redes.

2.3 Arquitectura de Detección de Rostros

Habiendo explorado las arquitecturas internas y los bloques que pueden conformar el extractor de características, es esencial comprender cómo se emplea esta información para realizar la detección de rostros. Por lo tanto, lo primero es familiarizarse con la arquitectura FPN (Feature Pyramid Network) [Lin et al., 2017] utilizada de base, seguido de la arquitectura completa y su funcionamiento del detector de rostros, RetinaFace [Deng et al., 2019], el cual es uno de los que se posiciona con mejor precisión en la base de datos WIDER FACE [Yang et al., 2016].

2.3.1 Feature Pyramid Network

La Feature Pyramid Network (FPN) es una arquitectura diseñada para la detección de objetos. Su enfoque consiste en obtener características del extractor de características en diferentes etapas, lo que permite combinar características de alto nivel con características obtenidas en las últimas etapas del extractor.

Estas características obtenidas en diferentes etapas se fusionan mediante interpolación de las capas de menor resolución hacia las capas superiores, como se ilustra en la Figura 13 representando el funcionamiento general de la FPN.

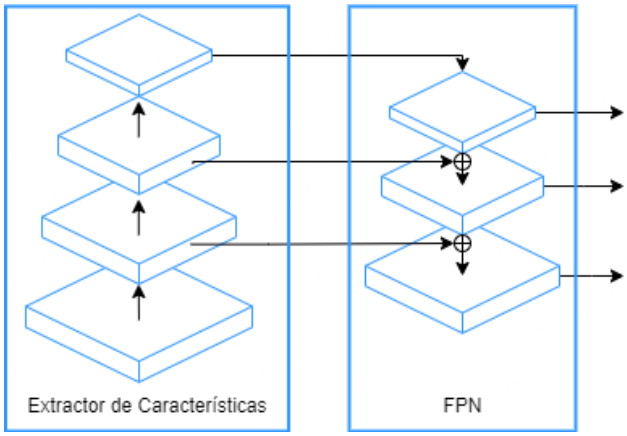

Figura 13: Diseño general de la arquitectura FPN.

Profundizando más en la FPN, las conexiones que fluyen desde el extractor de características hacia la FPN se basan en convoluciones de kernel 1x1. Por otro lado, el aumento de la resolución espacial de las capas inferiores se realiza generalmente a través de interpolaciones. No obstante, existen también variaciones que emplean capas de convolución transpuesta (también conocida como convolución inversa). Esta opción permite una interpolación y un aumento de la dimensionalidad espacial con un mejor rendimiento, gracias a la adaptabilidad de los pesos de estas convoluciones durante el entrenamiento, aunque incorporando un mayor costo computacional. Las conexiones entre la FPN y el extractor de características, junto con la interpolación a utilizar para cada fusión entre cada etapa de la FPN se ilustra en la Figura 14.

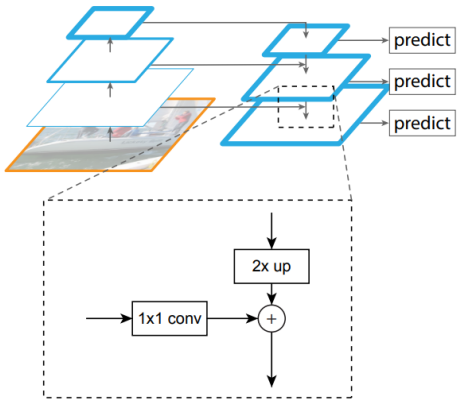

Figura 14: Diseño interno del upsampling y conexión de una arquitectura FPN. Imagen de [Lin et al., 2017].

En resumen, la ventaja clave que tiene este diseño para detección de objetos radica en su capacidad para obtener diferentes escalas con información contextual y espacial, lo que permite abordar la detección de objetos de manera más efectiva, detectando de mejor manera objetos de múltiples tamaños. Este diseño de la FPN es adaptable, ya que

el extractor de características puede ajustarse sobre arquitectura, lo que la hace compatible con la mayoría de las redes extractor de características disponibles.

2.3.2 RetinaFace

RetinaFace es una arquitectura de detección de rostros de gran nivel que forma parte de las pertenecientes al estado del arte de los últimos años [Feng et al., 2022]. El trabajo investigación realizado en RetinaFace introduce una variedad de cambios a los modelos de detecciones de rostros, abordando aspectos como la función de costo, el entrenamiento y el diseño de la arquitectura. La descomposición de la arquitectura de detección obtenida por RetinaFace se presenta en la Figura 15.

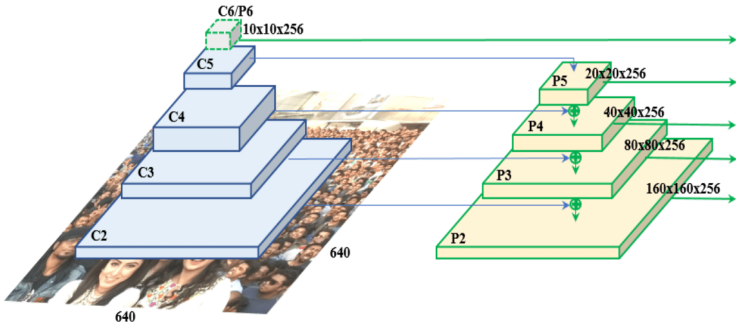

Figura 15: Diseño general de la arquitectura de RetinaFace. Imagen de [Deng et al., 2019].

Los bloques representados en la parte izquierda corresponden a las capas del extractor de características. Mientras que, en la parte derecha de la imagen, se muestra una Feature Pyramid Network (FPN) [Lin et al., 2017], una arquitectura que captura características por escala provenientes del extractor de características. Las nuevas características resultantes de la FPN son dirigidas a través de los cabezales de convolución en la arquitectura completa de RetinaFace. Estos cabezales generan la clasificación y los puntos de bounding box de la imagen procesada.

RetinaFace emplea la red extractor de características ampliamente utilizada para arquitecturas de detección llamado ResNet50 [He et al., 2016], pero también incluye una variante de extractor de características para dispositivos más pequeños, llamado MobileNetV1 0.25, en donde el 0.25 indica la división de cada canal de la red original en 4.

El proceso de entrenamiento y validación se realizan en la base de datos WIDER FACE [Yang et al., 2016], que destaca por su amplitud y diversidad en las imágenes que la componen, convirtiéndose en un punto de referencia para evaluar este tipo de detecciones. Los resultados obtenidos por RetinaFace ejemplificando su alta precisión en detección de rostros son los mostrados en la Figura 16.

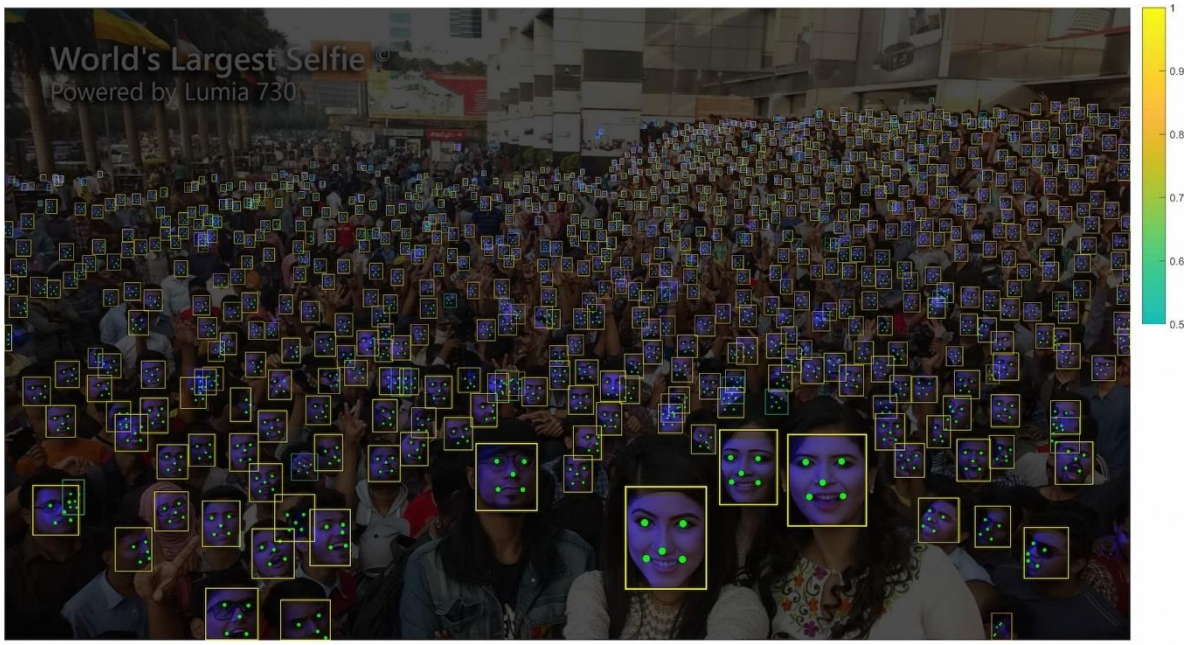

Figura 16: Detección de rostros utilizando RetinaFace. Imagen de [Deng et al., 2019].

2.4 Trabajos en el Área de NAS

Tal como se menciona en la introducción de esta tesis, existe una gran cantidad de investigaciones en el área de procesamiento avanzado de imágenes con Deep Learning y redes convolucionales, pero estas han ido convergiendo a un punto de mejoras cada vez más pequeñas. En consecuencia, se ha vuelto esencial aprovechar la inteligencia artificial para impulsar los avances de manera más eficiente y efectiva.

En este contexto, se utiliza el método de Neural Architecture Search (NAS) para descubrir redes neuronales capaces de superar en distintos aspectos a las ya existentes, siendo los bloques internos de estas redes ya creadas las utilizadas como base para mejora. El método NAS, además de buscar maximizar la precisión, permite la posibilidad de incorporar ciertas limitaciones fundamentales para las redes en distintos ámbitos, como la reducción del costo computacional, disminuir la cantidad de parámetros totales de la red o la restricción a bloques convolucionales específicos. A continuación, se presentan algunas de las investigaciones en el área de NAS que aportan de manera teórica y metodológica a esta tesis.

2.4.1 DARTS: Differentiable Architecture Search

DARTS es un estudio que aborda la búsqueda de arquitecturas mediante un enfoque basado en gradientes. Este método emplea una estrategia de optimización donde los gradientes se utilizan para aprender los pesos de las operaciones y conexiones en una red neuronal. Para lograr esto, introducen una operación de ponderación para cada una de las posibles operaciones en las capas y subcapas de la red. Esta ponderación global en la red permite la búsqueda de hiperparámetros que determinan las operaciones más influyentes, lo que conlleva a la definición de la arquitectura final.

DARTS realiza la búsqueda y entrenamiento de la red en la base de datos CIFAR-10 que consiste en 60.000 imágenes de dimensiones 32x32 con 10 clases de objetos distintas. Posterior a esto realizan la evaluación de la red encontrada en la base de datos ImageNet [Deng et al., 2009]. Si bien los resultados superan a algunas arquitecturas existentes en ese momento, uno de los avances más significativos fue la rapidez con la que se obtuvo esta nueva arquitectura, en comparación con otros enfoques, como se muestra en la Tabla 12.

Tabla 12: Resultados de la red DARTS frente a otras redes. Tabla de [Liu et al., 2018].

Architecture	Test Error (%)		Params	+×	Search Cost	Search
	top-1	top-5	(M)	(M)	(GPU days)	Method
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet 2× ($g = 3$) (Zhang et al., 2017)	26.3	–	∼5	524	–	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000	RL
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	488	2000	RL
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	558	2000	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	588	∼225	SMBO
DARTS (searched on CIFAR-10)	26.7	8.7	4.7	574	4	gradient-based

DARTS logra encontrar una arquitectura en la base de datos CIFAR-10 en sólo 4 días en comparación a otros trabajos donde llegar a durar hasta 3150 días en términos de uso de GPU. Aunque este progreso frente a las investigaciones actuales es importante, es esencial considerar que el método de búsqueda no permite una exploración directa en bases de datos más grandes, como ImageNet [Deng et al., 2009], debido a las demandas computacionales y de parámetros que este enfoque requeriría. Como se mencionó previamente, se requiere emplear una base de datos más pequeña para la búsqueda y luego transferir la arquitectura a bases de datos más grandes.

2.4.2 MnasNet

MnasNet es el primer trabajo de investigación dentro de NAS que busca modificar la forma de encontrar una arquitectura adaptada al hardware, cambiando el objetivo de minimizar la cantidad de FLOPs (operaciones de punto flotante por segundo) a directamente minimizar la latencia, logrando un equilibrio entre estas variables y la precisión de la arquitectura. Este cambio se debe a que, como muestran en su trabajo, los FLOPs no son directamente proporcionales a la latencia o velocidad de la arquitectura. Esto permite una búsqueda de arquitecturas más eficientes y precisas en los valores objetivo esperados.

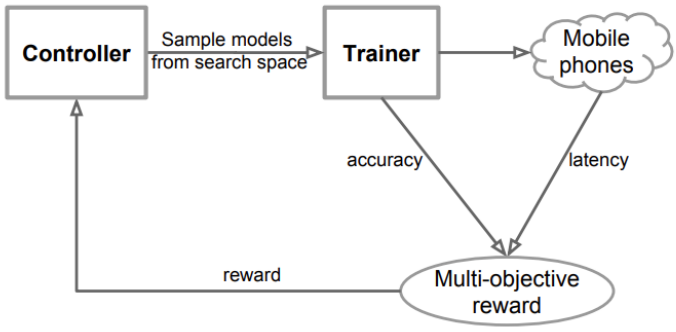

Figura 17: Diseño general del funcionamiento de MnasNet. Imagen de [Tan et al., 2019].

Como se observa en el diagrama de la Figura 17, el método de MnasNet busca construir una arquitectura a través de una búsqueda con recompensa multi-objetivo. Esto consiste en minimizar la latencia del modelo (evaluada en un teléfono móvil) y maximizar su precisión en la base de datos de entrenamiento. Esta búsqueda emplea el enfoque de Reinforcement Learning (RL), donde un controlador (una red neuronal entrenada mediante RL) genera una muestra de arquitecturas que se entrenan y evalúan. Esta evaluación abarca la precisión y la latencia en un teléfono móvil de referencia. La función objetivo para optimizar la recompensa se encuentra en la ecuación 4.

$$maximizar\ m, \qquad ACC(m) * \left[\frac{LAT(m)}{T}\right]^{w} \tag{4}$$

Donde ACC(m) corresponde a la precisión de la red neuronal en la base de datos entrenada, LAT(m) representa la latencia del modelo, y T es la latencia objetivo. El factor de peso w se asigna como alpha si LAT(m) es menor que la latencia objetivo T, y como beta en caso contrario. Estos valores de alpha y beta son constantes definidas por un humano experto, y en los experimentos se establecieron ambos con un valor de -0.07.

Estos resultados son la recompensa multi-objetivo que se le da al controlador como retroalimentación y se itera el proceso hasta obtener una red neuronal óptima o hasta alcanzar la convergencia.

Tabla 13: Resultados y comparativa entre MnasNet y otras redes móviles. Tabla de [Tan et al., 2019].

Model	Type	#Params	#Mult-Adds	Top-1 Acc. (%)	Top-5 Acc. (%)	Inference Latency
MobileNetV1 [11]	manual	4.2M	575M	70.6	89.5	113ms
SqueezeNext [5]	manual	3.2M	708M	67.5	88.2	-
ShuffleNet (1.5x) [33]	manual	3.4M	292M	71.5	-	-
ShuffleNet (2x)	manual	5.4M	524M	73.7	-	-
ShuffleNetV2 (1.5x) [24]	manual	-	299M	72.6	-	-
ShuffleNetV2 (2x)	manual	-	597M	75.4	-	-
CondenseNet (G=C=4) [14]	manual	2.9M	274M	71.0	90.0	-
CondenseNet (G=C=8)	manual	4.8M	529M	73.8	91.7	-
MobileNetV2 [29]	manual	3.4M	300M	72.0	91.0	75ms
MobileNetV2 (1.4x)	manual	6.9M	585M	74.7	92.5	143ms
NASNet-A [36]	auto	5.3M	564M	74.0	91.3	183ms
AmoebaNet-A [26]	auto	5.1M	555M	74.5	92.0	190ms
PNASNet [19]	auto	5.1M	588M	74.2	91.9	-
DARTS [21]	auto	4.9M	595M	73.1	91	-
MnasNet-A1	**auto**	**3.9M**	**312M**	**75.2**	**92.5**	**78ms**
MnasNet-A2	**auto**	**4.8M**	**340M**	**75.6**	**92.7**	**84ms**
MnasNet-A3	**auto**	**5.2M**	**403M**	**76.7**	**93.3**	**103ms**

Sus resultados mostrados en la Tabla 13 demuestran que este nuevo enfoque logra su objetivo de encontrar una nueva arquitectura con mejor precisión en la base de datos de clasificación ImageNet [Deng et al., 2009], mientras que su latencia sigue siendo inferior a estas otras arquitecturas móviles, incluso cuando la cantidad de parámetros en algunos casos es mayor.

2.4.3 ProxylessNas

ProxylessNas destaca como uno de los primeros trabajos de investigación que logra realizar búsquedas directamente en bases de datos de gran tamaño como ImageNet [Deng et al., 2009]. Como su nombre lo indica, se caracteriza por su capacidad de realizar la búsqueda y entrenamiento de arquitecturas de redes neuronales en una base de datos de gran tamaño sin necesidad de utilizar un proxy o base de datos pequeña para la búsqueda inicial. Esto se debe principalmente a su forma de realizar la búsqueda y carga de los bloques convolucionales que la componen, siendo su diferencia de diseño general la mostrada en la Figura 18.

Figura 18: Diseño general de funcionamiento de método NAS vs Diseño General de ProxylessNas. Imagen de [Cai et al., 2018].

Un aspecto clave de ProxylessNas es su flexibilidad para aplicar restricciones como FLOPs, latencia en CPU, latencia en GPU e incluso latencia en dispositivos móviles. Esto

lo convierte en una base importante para investigaciones futuras debido a que puede personalizarse para diversos tipos de hardware con recursos limitados.

Su funcionamiento se basa en un enfoque de búsqueda y optimización basado en gradientes. Este enfoque consiste en la asignación de un peso ponderado a cada operación candidata en los módulos de la red que se busca. Esto es similar a DARTS [Liu et al., 2018] tanto en el enfoque mediante gradientes como en la utilización de un peso ponderado para cada operación.

Para llevar a cabo este método, primero se construye una red sobre parametrizada que incluye todos los bloques candidatos pertenecientes al espacio de búsqueda. Con el fin de reducir el consumo de memoria y agilizar la búsqueda, se utiliza una arquitectura de parámetros binarios. Esta arquitectura de parámetros binarios permite dejar activo sólo uno de los bloques candidatos por capa y desactivar el resto de los bloques candidatos durante una iteración específica del entrenamiento y la búsqueda de la red neuronal. La selección del bloque activo se determina en función de su peso ponderado con respecto a los demás bloques de la misma capa. En otras palabras, se aplica una distribución multinomial a estos valores ponderados por bloque para decidir cuál de ellos permanece activo. En la ecuación 5 se presenta la fórmula correspondiente.

$$binarizar(p_1, \dots, p_n) = \{ \begin{cases} [1, 0, \dots, 0] \; con\ una\ probabilidad\ de\ p_1 \\ \dots \\ [0, 0, \dots, 1] con\ una\ probabilidad\ de\ p_n \end{cases} \tag{5}$$

La probabilidad para que un bloque específico, denotado como *Pn*, se active se encuentra directamente relacionada con su peso ponderado en la capa correspondiente. Para determinar la importancia de este bloque y ajustar sus pesos a lo largo del entrenamiento y la búsqueda, se implementa un proceso de retropropagación y aprendizaje en las capas actuales. Esto permite evaluar continuamente la relevancia de cada bloque dentro de la misma red neuronal y, en última instancia, seleccionar el bloque óptimo para minimizar la función de pérdida del método NAS.

La función de pérdida tiene como objetivo aumentar la precisión durante el proceso de entrenamiento, manteniéndose dentro de los límites de FLOPs o latencia predefinidos por un experto humano. Un esquema completo de estos procesos se muestra en la Figura 19.

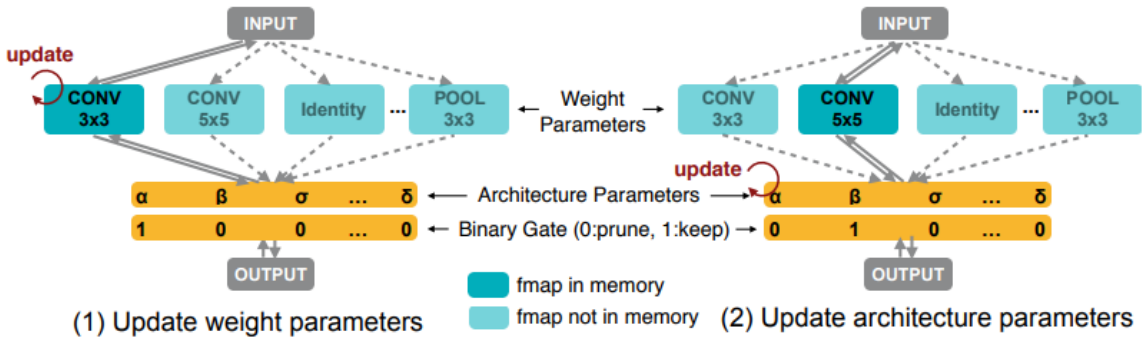

Figura 19: Esquema completo del funcionamiento de búsqueda y entrenamiento de los bloques y las Binary Gates. Imagen de [Cai et al., 2018].

De esta manera, durante los procesos de entrenamiento y búsqueda, el uso de memoria de la red se reduce significativamente. Esto se logra al cargar en memoria únicamente los bloques activos, excluyendo los bloques desactivados. Esta mejora permite la aplicación directa del método en bases de datos más grandes, como ImageNet [Deng et al., 2009], tal como se mencionó inicialmente.

En cuanto a la función de costo utilizada, aunque la precisión es diferenciable y, por lo tanto, admite la retropropagación y la optimización mediante gradientes, las restricciones relacionadas con FLOPs y latencia no lo son de manera intrínseca. Para incorporar estas restricciones, se realizan modificaciones específicas. Estas consisten en multiplicar la probabilidad del peso ponderado de cada bloque por su respectiva latencia o FLOPs y luego sumar estos valores para cada capa, obteniendo así la fórmula mostrada en la Figura 20.

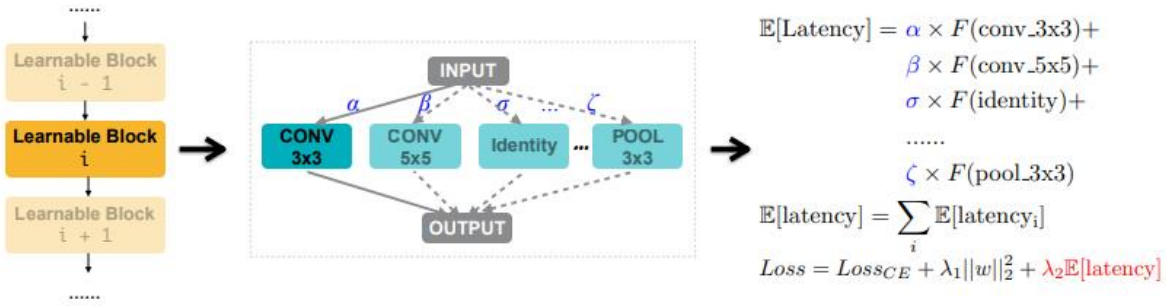

Figura 20: Fórmula de obtención del loss de latencia/FLOPs. Imagen de [Cai et al., 2018].

También se agrega un factor de escala λ_2, que es una constante utilizada para equilibrar la importancia entre la precisión y la latencia, ajustándola según se estime necesario. Mientras que $\lambda 1||w||_2^2$ representa el término de decaimiento de peso, comúnmente conocido como *weight decay* el cual se utiliza para evitar el sobreajuste del modelo. Los resultados obtenidos para una GPU Tesla V100 por este método luego de

utilizarse directamente sobre la base de datos ImageNet [Deng et al., 2009] son los mostrados en la Tabla 14.

Tabla 14: Resultados y comparativa de red Proxyless (GPU). Tabla de [Cai et al., 2018].

Model	Top-1	Top-5	GPU latency
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	6.1ms
ShuffleNetV2 (1.5) (Ma et al., 2018)	72.6	-	7.3ms
ResNet-34 (He et al., 2016)	73.3	91.4	8.0ms
NASNet-A (Zoph et al., 2018)	74.0	91.3	38.3ms
DARTS (Liu et al., 2018c)	73.1	91.0	-
MnasNet (Tan et al., 2018)	74.0	91.8	6.1ms
Proxyless (GPU)	**75.1**	**92.5**	**5.1ms**

Obteniendo mejores resultados en precisión y latencia que trabajos anteriores a ProxylessNas como los ya mencionados MnasNet [Tan et al., 2019] y DARTS [Liu et al., 2018]. Mientras que los resultados para distintos hardware son los presentados en la Tabla 15.

Tabla 15: Resultados para distintos dispositivos con el método de ProxylessNas. Tabla de [Cai et al., 2018].

Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

Demostrando que la red si logra diferenciar en su búsqueda de red neuronal entre distintos hardware, haciendo una red con bloques específicos para cada requerimiento y limitación.

2.4.4 GENet

La investigación de GENet, realizado por Alibaba Group, se centra en la búsqueda de diseño de arquitecturas, pero a diferencia de otros enfoques que tienen un espacio de búsqueda limitado, GENet incorpora una variedad de bloques de diferentes familias de redes neuronales. Esto significa que no se restringen a un particular tipo de bloque en su espacio de búsqueda, lo que les brinda más flexibilidad en la exploración y permite encontrar arquitecturas óptimas para diversos propósitos.

En esta investigación de la GENet, se utilizan tres tipos de bloques de convolución básicos que son comunes en muchas redes neuronales de gran desempeño. Estos bloques son:

- **XX-Block:** Bloque que consiste en dos convoluciones básicas de manera secuencial con kernel kxk. Su diseño se encuentra en la Figura 21.

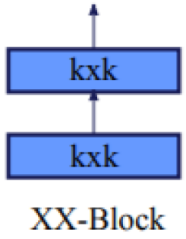

Figura 21: Diseño XX-Block. Imagen de [Lin et al., 2020].

- **BL-Block:** Bloque proveniente del diseño BottleNeck creado en la arquitectura ResNet [He et al., 2016]. Su diseño se encuentra en la Figura 22 y se compone de la siguiente manera:

    1. Primero se utiliza una convolución inicial de kernel 1x1, con el fin de reducir la dimensionalidad de los canales.
    2. Posteriormente, se encuentra una convolución de kernel kxk con una expansión de canales de factor 1/r.
    3. Finalmente, una convolución de kernel 1x1.

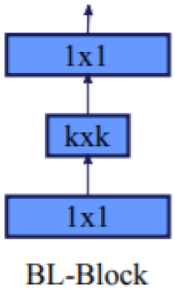

Figura 22: Diseño BL-Block. Imagen de [Lin et al., 2020].

- **DW-Block:** El bloque DW-Block se basa en los bloques Depth-Wise utilizados en la familia de las MobileNets [Howard et al., 2017, Howard et al., 2019, Sandler et al., 2018]. Estos bloques son comúnmente utilizados en redes diseñadas para dispositivos móviles o de menor recursos computacionales. Su diseño se presenta en la Figura 23 y su composición es la siguiente:

    1. Utiliza una convolución inicial de kernel 1x1.
    2. Luego, sigue una convolución de kernel kxk, que se realiza de forma Depth-wise, lo que significa que se aplica una convolución separada para cada canal de entrada sin mezclarlos. Además, se utiliza una técnica de expansión de canales en la que la cantidad de canales se aumenta en un factor r.

3. Finalmente, se aplica una convolución de kernel 1x1 que se utiliza para restaurar la dimensionalidad de los canales a su valor original.

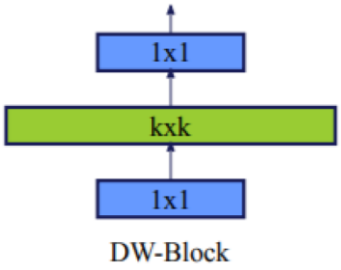

Figura 23: Diseño DW-Block. Imagen de [Lin et al., 2020].

Estos bloques básicos son el esqueleto de la red y pueden tener conexiones residuales de la manera mostrada en la Figura 24.

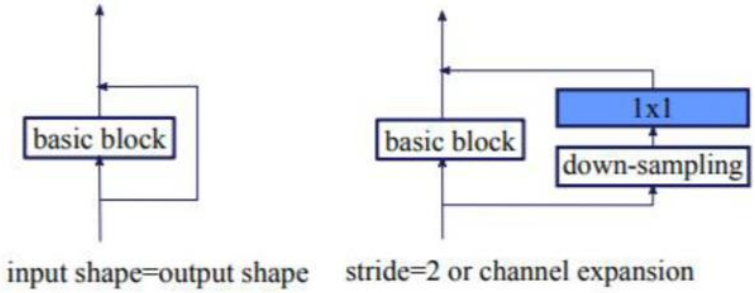

Figura 24: Diseño para conexiones residuales en GENet. Imagen de [Lin et al., 2020].

La Figura 24 muestra que al utilizar un stride de valor 2, se incorpora una convolución de kernel 1x1 (o simplemente se realiza una operación de pooling antes de la convolución) en la conexión residual como parte del diseño.

El trabajo de investigación de GENet, antes de comenzar a diseñar arquitecturas específicas o a hablar sobre el método de búsqueda NAS, se llevaron a cabo múltiples experimentos para comprender el comportamiento de los tres tipos de bloques (XX-Block, BL-Block y DW-Block) en diferentes configuraciones de batches, canales y cantidad de parámetros. A partir de estos experimentos, se llegaron a las siguientes conclusiones:

- La latencia de inferir en una GPU no se encuentra directamente relacionada con la cantidad de operaciones realizadas por la red neuronal (FLOPs) o del tamaño del modelo (cantidad de parámetros).
- Se encuentra que el bloque XX-Block, que consiste en convoluciones directas, es el más costoso en términos de latencia. Sin embargo, este bloque es eficiente en términos de capacidad de adaptación de la red neuronal para aprender funciones complejas.

- Se concluye que, para maximizar la capacidad de la red, es beneficioso utilizar bloques XX-Block en las etapas de nivel bajo (las primeras etapas) y bloques BL-Block y DW-Block en las etapas de nivel alto (las últimas etapas) para aprovechar al máximo la eficiencia de la GPU.

Basándose en estas conclusiones, se diseñaron 20 redes neuronales con una restricción de latencia de 0.34ms de latencia para un batch de 64 y se entrenaron en la base de datos ImageNet [Deng et al., 2009] durante 120 épocas. Durante este diseño, se mantuvo el BL-Block con una expansión de ¼ y los DW-Block con una expansión de 6. Los resultados y el rendimiento de estas redes se muestran en la Tabla 16.

Tabla 16: Diseños de los experimentos y resultados realizados manualmente. Tabla de [Lin et al., 2020].

Model	Block Type	# Layers	# Channels	Stride	Acc
Net1	C,X,X,B,D,D,C	1,1,4,8,4,2,1	32,64,96,512,320,320,1280	2,2,2,2,2,1,1	77.6%
Net2	C,X,X,D,D,D,C	1,1,4,8,4,2,1	32,48,64,160,320,320,1280	2,2,2,2,2,1,1	77.6%
Net3	C,X,X,D,D,C	1,1,4,8,6,1	32,48,64,160,320,1280	2,2,2,2,2,1	77.5%
Net4	C,X,B,B,D,C	1,1,4,8,6,1	32,64,256,512,320,1280	2,2,2,2,2,1	77.4%
Net5	C,X,B,D,D,D,C	1,1,4,8,4,2,1	32,32,256,144,288,288,1280	2,2,2,2,2,1,1	77.4%
Net6	C,X,X,B,D,C	1,1,4,8,6,1	32,64,96,512,320,1280	2,2,2,2,2,1	77.3%
Net7	C,X,B,D,D,C	1,1,4,8,6,1	32,32,256,144,288,1280	2,2,2,2,2,1	77.3%
Net8	C,D,D,D,D,D,C	1,1,4,8,4,2,1	32,24,64,128,256,256,1280	2,2,2,2,2,1,1	77.2%
Net9	C,X,X,D,D,D,C	1,1,4,4,4,4,1	32,48,64,160,160,320,1280	2,2,2,2,1,2,1	77.1%
Net10	C,D,D,D,D,C	1,1,4,8,6,1	32,24,64,128,256,1280	2,2,2,2,2,1	77.0%
Net11	C,X,X,D,D,D,C	1,1,4,6,2,4,1	32,48,64,160,160,320,1280	2,2,2,2,1,2,1	76.9%
Net12	C,X,X,B,B,D,C	1,1,4,6,2,4,1	32,64,96,512,512,320,1280	2,2,2,2,1,2,1	76.9%
Net13	C,X,X,B,B,D,C	1,1,4,4,4,4,1	32,64,96,512,512,320,1280	2,2,2,2,1,2,1	76.8%
Net14	C,X,B,D,D,D,C	1,1,4,6,2,4,1	32,32,256,144,144,288,1280	2,2,2,2,1,2,1	76.7%
Net15	C,X,X,B,B,B,C	1,1,4,8,4,2,1	32,64,96,512,1024,1024,1280	2,2,2,2,2,1,1	76.6%
Net16	C,D,D,D,D,D,C	1,1,4,6,2,4,1	32,24,64,128,128,256,1280	2,2,2,2,1,2,1	76.5%
Net17	C,X,B,B,B,C	1,1,4,8,6,1	32,64,256,512,1024,1280	2,2,2,2,2,1	76.3%
Net18	C,X,X,B,B,B,C	1,1,4,6,2,4,1	32,64,96,512,512,1024,1280	2,2,2,2,1,2,1	76.0%
Net19	C,X,X,B,B,C	1,1,4,8,6,1	32,64,96,512,1024,1280	2,2,2,2,2,1	76.0%
Net20	C,X,X,B,B,B,C	1,1,4,4,4,4,1	32,64,96,512,512,1024,1280	2,2,2,2,1,2,1	75.8%

En la tabla se muestran los diferentes tipos de bloques utilizados en las redes, donde *C* representa una convolución directa, *X* son los bloques XX-Block, *B* son los bloques BL-Block y *D* son los bloques DW-Block. Se destaca que las redes de mejor rendimiento tienen bloques XX-Block en sus primeras etapas y bloques BL-Block y DW-Block en las últimas etapas.

La red designada como MasterNet es el modelo Net1, aunque mencionan que Net2 y Net3 también tienen un rendimiento similar. La diferencia entre estas tres redes radica en la inclusión de cada uno de los tipos de bloques (XX, BL y DW). Con la MasterNet seleccionada como base, se procede a utilizar el método NAS, que parametriza cada uno de los bloques de esta red y realiza los siguientes pasos:

- **Destilación:** Cada bloque se reemplaza con un bloque aleatorio y se realiza fine-tuning hasta que alcance su convergencia. Obteniendo así la precisión en validación de esta nueva arquitectura, mientras que este proceso se repite hasta tener N precisiones de bloques distintos.
- **Regresión:** Teniendo la precisión correspondiente a las N muestras, para cada bloque y tipo de bloque se utiliza regresión cuadrática para estimar un pseudo-gradiente que sea capaz de predecir la precisión de cualquier arquitectura a partir de sus parámetros estructurales (que corresponden a la dimensión, cantidad de canales y tipo de bloque).
- **Selección:** Genera de manera aleatoria M arquitecturas y predice su precisión usando el pseudo-gradiente obtenido anteriormente. Finalmente, se elige la arquitectura que tiene la mejor precisión dentro de los límites impuestos para la cota máxima de latencia.

Utilizando este método semi-automático de NAS para diseñar redes neuronales eficientes para GPU, se obtienen tres tipos de diseños de redes: GENet-light, GENet-normal y GENet-large. Estas redes se entrenan en la base de datos ImageNet [Deng et al., 2009], y los resultados de precisión y latencia en comparación con otras redes se presentan en la Tabla 17.

Tabla 17: Resultados y comparativa de redes finales. Tabla de [Lin et al., 2020].

		V100 FP16		T4 FP16		T4 INT8	
model	acc	latency	speed-up	latency	speed-up	latency	speed-up
GENet-light	**75.7%**	**0.1**	**1.0x**	**0.14**	**1.0x**	**0.1**	**1.0x**
ResNet-34	74.4%	0.22	2.2x	0.33	2.3x	0.19	1.9x
RegNetY-600MF	75.5%	0.31	3.1x	0.28	2.0x	0.22	2.2x
MobileNetV2-1.4	74.7%	0.24	2.4x	0.30	2.1x	0.20	2.0x
DNANet-2a	76.0%	0.19	1.9x	0.24	1.7x	0.16	1.6x
OFANet-9ms	75.3%	0.14	1.4x	0.21	1.5x	0.15	1.5x
GENet-normal	**80.0%**	**0.20**	**1.0x**	**0.25**	**1.0x**	**0.15**	**1.0x**
ResNet-152	79.2%	0.94	4.7x	1.04	4.2x	0.55	3.7x
EfficientNet-B2	79.8%	0.64	3.2x	1.32	5.3x	1.02	6.8x
OFANet-595M	80.0%	0.41	2.1x	0.71	2.8x	0.63	4.2x
GENet-large	**81.3%**	**0.34**	**1.0x**	**0.44**	**1.0x**	**0.26**	**1.0x**
EfficientNet-B3	81.1%	1.12	3.3x	2.36	5.4x	1.67	6.4x

2.5 Base de Datos

2.5.1 WIDER FACE

WIDER FACE [Yang et al., 2016] es una base de datos de rostros creada a partir de la base de datos pública WIDER [Xiong et al., 2015]. Esta nueva base de datos WIDER

FACE [Yang et al., 2016] incluye un total de 32.203 imágenes y 393.703 rostros etiquetados. Estos rostros presentan una gran variabilidad en términos de escala, pose y oclusión, lo la convierte en una base de datos ampliamente utilizada en diversos trabajos de investigación como benchmark. Algunos ejemplos de la base de datos WIDER FACE [Yang et al., 2016] y sus tipos de variaciones existentes se encuentran en la Figura 25.

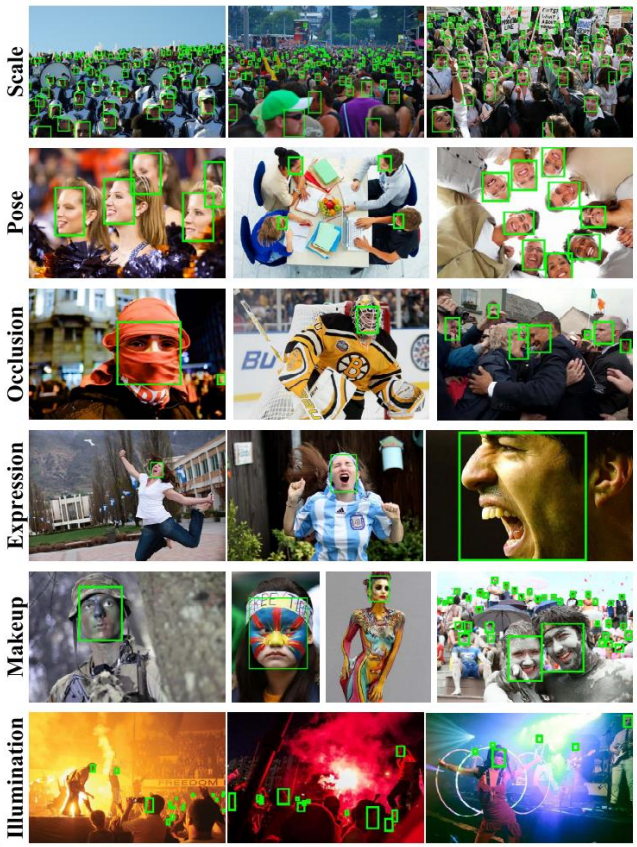

Figura 25: Imágenes de ejemplo y variaciones posibles en base de datos WIDER FACE. Imagen de [Yang et al., 2016].

La base de datos WIDER FACE [Yang et al., 2016] se divide tres conjuntos: un conjunto de entrenamiento que abarca el 40% de los datos, un conjunto de validación que representa el 10% y un conjunto de prueba que consiste en el 50% restante. Para evaluar la dificultad de la detección de rostros, la base de datos categoriza los rostros en tres niveles de dificultad: Fácil (Easy), Medio (Medium) y Difícil (Hard).

Además, WIDER FACE [Yang et al., 2016] está organizada en 61 clases de eventos, lo que proporciona una mayor variabilidad en comparación con bases de datos que pueden estar limitadas a entornos o escenarios específicos. Esta base de datos se utilizará en el trabajo de FaceNAS, lo que permitirá llevar a cabo un análisis exhaustivo y preciso en el contexto de la detección de rostros.

Existen investigaciones recientes, como la de [Luo et al., 2022], que demuestran la capacidad de mejorar la precisión de los modelos convolucionales mediante ligeras correcciones de las etiquetas de los cuadros delimitadores (bounding box). Sin embargo, dado que esta tesis se centra en la comparativa del uso de NAS, realizar esta modificación no permitiría una comparación directa entre los resultados. Por lo tanto, este avance se considera como una oportunidad de investigación futura, más allá del alcance de esta tesis.

Capítulo 3

Metodología

Esta tesis tiene como objetivo crear un método NAS que funcione directamente con una base de detección de rostros y se adapte de manera eficiente al hardware objetivo. De esta manera lograr encontrar un extractor de características óptimo a los recursos computacionales y restricciones establecidas por el humano experto. Para realizar esto, se aprovecharán los trabajos previos, en particular RetinaFace [Deng et al., 2019] y ProxylessNas [Cai et al., 2018], como punto de partida para la creación del método NAS que opere directamente con la base de datos de detección de rostros WIDER FACE [Yang et al., 2016].

En las secciones siguientes, se proporcionará una explicación detallada de cómo se llevará a cabo la adaptación y reestructuración del método NAS basado en ProxylessNas [Cai et al., 2018] con el fin de implementar la arquitectura de detección de RetinaFace [Deng et al., 2019]. Además, se describirá en detalle el proceso de búsqueda y entrenamiento directo en la base de datos WIDER FACE [Yang et al., 2016], así como la incorporación de una amplia variedad de bloques básicos y complementos, como los módulos SE [Hu et al., 2018] y eSE [Lee & Park, 2020], en el espacio de búsqueda.

3.1 Diseño General del Sistema

El diseño general del sistema se basa en el método NAS ProxylessNas [Cai et al., 2018] para aprovechar su posibilidad de trabajar con bases de datos de gran tamaño, pero modificando su funcionamiento e implementando la arquitectura propuesta por RetinaFace [Deng et al., 2019] para la detección de rostros. En este contexto, el extractor de características de la RetinaFace [Deng et al., 2019] será reemplazado por la arquitectura de red neuronal que se busca encontrar a través del método NAS. El extractor de características se creará a partir de los bloques básicos de los que se compone el espacio de búsqueda y buscará optimizar la precisión del modelo con las restricciones definidas para otros parámetros como puede ser la latencia.

Esta nueva arquitectura del extractor de características estará conectada a una FPN de 3 niveles, siguiendo el diseño original de RetinaFace [Deng et al., 2019]. Además, se mantendrá la función de costo utilizada por RetinaFace [Deng et al., 2019] durante su entrenamiento y se incorporará la restricción de latencia del modelo, que se obtendrá para cada bloque interno del extractor de características.

La latencia de la arquitectura se puede calcular para diferentes tipos de hardware objetivo. Para realizar este cálculo, se ejecutará un script que creará un diccionario con información detallada sobre los bloques básicos del espacio de búsqueda y sus respectivas latencias. Este diccionario, que contiene información sobre la latencia de

cada bloque básico en distintos contextos de hardware, se cargará en el sistema donde se llevará a cabo el entrenamiento del método y la búsqueda de la arquitectura deseada.

Al tener información precisa sobre las latencias de cada bloque básico en el hardware objetivo, se podrá estimar de manera más precisa la función de costo y aplicar penalizaciones a los bloques en función de su latencia con respecto al objetivo establecido. Los detalles específicos de estos procesos se explicarán con mayor profundidad en secciones posteriores.

El entrenamiento y búsqueda de la arquitectura se llevará a cabo utilizando la base de datos WIDER FACE [Yang et al., 2016]. Esta elección se basa en el hecho de que WIDER FACE [Yang et al., 2016] es una base de datos ampliamente reconocida y utilizada como punto de referencia o benchmark en el área de detección de rostros. Además, RetinaFace [Deng et al., 2019], que es una de las referencias principales para esta tesis, ha obtenido sus resultados de precisión utilizando esta base de datos. Por lo tanto, el uso de WIDER FACE [Yang et al., 2016] como base de datos de entrenamiento permitirá una comparación directa entre los resultados obtenidos en esta tesis y los logrados por RetinaFace [Deng et al., 2019], lo que es fundamental para evaluar la eficacia y el rendimiento de la arquitectura de detección de rostros obtenida a partir del método NAS propuesto.

Es importante destacar que, aunque existen diversas técnicas de entrenamiento y modificaciones posibles para mejorar el rendimiento de la arquitectura final encontrada, esta tesis se centra en el método NAS y las mejoras que se pueden alcanzar a través de él. Cualquier modificación externa al método NAS que afecte la arquitectura final no se considerará en esta investigación, ya que dificulta una comparación directa entre la mejora de usar el método NAS propuesto por esta tesis y la arquitectura original de detección de rostros, RetinaFace [Deng et al., 2019]. Por lo tanto, el enfoque principal se mantendrá en realizar modificaciones en la arquitectura del extractor de características, preservando los hiperparámetros y las arquitecturas posteriores de la red, que se basa principalmente en mantener una FPN de 3 niveles, tal como se describe en el artículo de RetinaFace [Deng et al., 2019].

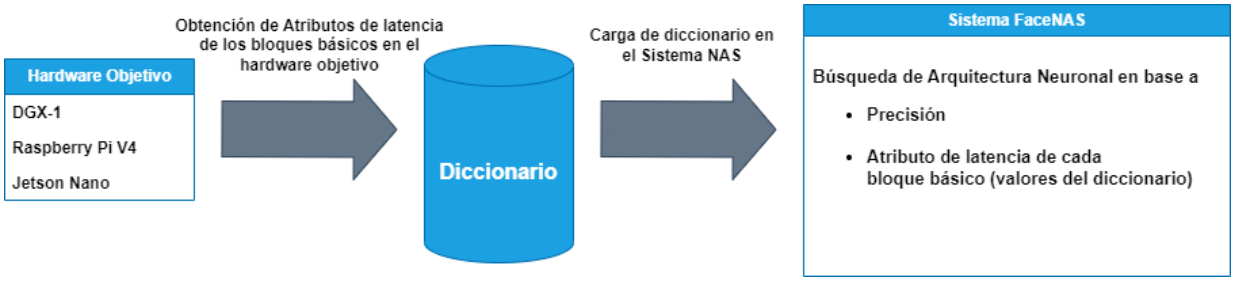

Figura 26: Esquema general método FaceNAS.

Como se presenta en la Figura 26, el método en términos generales se basa en utilizar un diccionario que contiene los atributos de latencia paca cada bloque básico

existente en el espacio de búsqueda. Este diccionario puede ser obtenido antes del proceso de búsqueda NAS mediante un script que realiza pruebas de latencia en el hardware objetivo o durante la búsqueda a medida que se evalúan diferentes arquitecturas.

Este método busca encontrar una arquitectura que optimice tanto la precisión como las restricciones de latencia definidas por el experto humano. La importancia relativa de la precisión y la cercanía a la latencia objetivo se puede ajustar mediante una variable configurable por el experto.

Una vez que se ha obtenido la arquitectura final a través del método NAS, se procede a cargar esta arquitectura en el proceso de entrenamiento directo. Durante este entrenamiento, se utilizan los mismos hiperparámetros que RetinaFace [Deng et al., 2019] para evaluar la precisión del modelo. Los resultados de precisión se comparan con los obtenidos en la validación de WIDER FACE [Yang et al., 2016] en las categorías Easy, Medium y Hard. Además, se evalúa la latencia de inferencia en el hardware objetivo para obtener una comprensión completa del rendimiento del modelo.

3.2 Arquitectura diseñada para Detección de Rostros

La arquitectura de detección de rostros propuesta se compone de los siguientes elementos:

1. **Extractor de Características:** Esta parte de la red se encarga de procesar la imagen de entrada y extraer características relevantes para la detección de rostros. Puede estar compuesto por una red convolucional como una ResNet [He et al., 2016] u otra arquitectura similar.
2. **Feature Pyramid Network (FPN)** [Lin et al., 2017]**:** La FPN es una estructura que consta de múltiples niveles de características, diseñada para capturar información a diferentes escalas espaciales. La salida del extractor de características se conecta a una FPN, lo cual se encuentra representado en la Figura 13. Esto es esencial para detectar rostros de diferentes tamaños en la imagen.
3. **Cabezal de Detección:** La FPN está conectada a un cabezal de detección, que tiene varias tareas importantes:
    - **Detección de Bounding Box:** Predice las coordenadas del cuadro delimitador (bounding box) que rodea al rostro en la imagen.
    - **Puntuación de Detección:** Proporciona una puntuación que indica la confianza del modelo de que un rostro está presente en la región delimitada por el bounding box.
    - **Detección de Landmarks:** Predice las ubicaciones de puntos clave en el rostro, los cuales son: los ojos, la nariz y los extremos de los labios.

En este contexto, el extractor de características es el componente que puede variar en la arquitectura, ya que el método NAS buscará crear y optimizar su estructura para equilibrar la precisión y la restricción de latencia. La FPN [Lin et al., 2017] y el cabezal de

detección se mantienen consistentes en la arquitectura, y sólo se modifican los pesos que la componen.

La arquitectura del extractor de características puede evolucionar a medida que el método NAS identifica bloques básicos óptimos en diversas capas. No obstante, esta arquitectura debe mantener una estructura mínima para asegurar una conexión constante entre el extractor de características y la FPN [Lin et al., 2017]. Esto implica que existe un número mínimo de capas en cada etapa encargadas de esta conexión. Si bien estas capas mínimas pueden modificarse entre los distintos tipos de bloques básicos, no pueden ser eliminadas. Esto garantiza un flujo apropiado de información. Por otro lado, las capas adicionales pueden variar según lo que el método NAS determine como más eficiente para mejorar la precisión de la detección de rostros sin sobrepasar la restricción de latencia.

3.3 Método NAS

El método NAS propuesto en esta tesis busca diseñar el extractor de características utilizando una combinación de bloques básicos. De esta manera optimizar la precisión de la arquitectura mientras se mantiene dentro de la restricción de latencia del modelo neuronal, un valor que es definido por el experto humano. Para comprender mejor la composición de este extractor de características, se proporciona una descripción general de cada bloque que lo conforma y de los módulos internos en cada una de estas etapas.

3.3.1 Espacio de Búsqueda

El espacio de búsqueda se refiere a los bloques básicos disponibles con los que el método NAS puede trabajar. En esta tesis, el espacio de búsqueda comprende bloques básicos de diversas familias de redes convolucionales, incluyendo ResNet [He et al., 2016], la familia de MobileNets [Howard et al., 2017, Howard et al., 2019, Sandler et al., 2018], DarkNet [Redmon & Farhadi, 2018] y varias variaciones desarrolladas en el trabajo de investigación GENet [Lin et al., 2020]. Estos bloques básicos han sido utilizados y adaptados en numerosos trabajos recientes de diferentes maneras. Por ejemplo, EfficientNet [Tan & Le, 2019] utiliza los bloques de MobileNetV2 [Sandler et al., 2018] como base para lograr resultados del estado del arte en su año de investigación.

A continuación, enumeraremos estos bloques y describiremos su arquitectura interna, y más adelante, abordaremos algunas variaciones que se pueden aplicar a estos bloques, como SE (Squeeze-and-Excitation) [Hu et al., 2018] y eSE (Effective Squeeze-and-Excitation) [Lee & Park, 2020].

3.3.1.1 Arquitectura del Bloque

Es importante destacar que cada uno de estos bloques básicos tendrá una conexión residual o shortcut, que varía según si el bloque pertenece al inicio de su etapa correspondiente o no. Estas conexiones residuales desempeñan un papel clave en la mejora de la precisión y el entrenamiento de las redes neuronales al reducir el problema del desvanecimiento del gradiente durante la retropropagación.

- **Conexión Residual Base:** Estas conexiones residuales se utilizan cuando el bloque básico tiene la misma cantidad de canales de entrada que de salida, es decir, no hay una variación en la cantidad de canales en este bloque. Esta conexión ayuda a mejorar la precisión y el entrenamiento al mitigar el problema del desvanecimiento del gradiente durante la retropropagación. Este tipo de conexión se ilustra en la Figura 27.

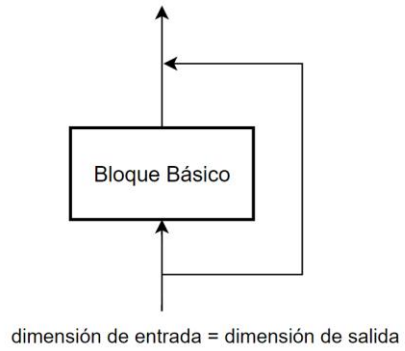

Figura 27: Diseño de conexión residual base.

- **Conexión Residual para primer bloque de etapa:** Dentro de la conexión residual, tal como se representa en la Figura 28, se aplica una convolución 1x1 con un stride de valor 1 o 2 según el valor definido para la etapa por el humano experto. Esta conexión se basa en el trabajo realizado en “Bag of Tricks” [He et al., 2018] para la ResNet [He et al., 2016] y se mencionó en el punto 3.1.2. Por otro lado, debido a la pérdida de información que puede ocurrir al utilizar convoluciones de kernel 1x1 con stride 2, estas conexiones se reemplazan por una operación de Average Pooling de kernel 2x2 con un stride de 2, seguido de una convolución de kernel 1x1 con un stride de 1.

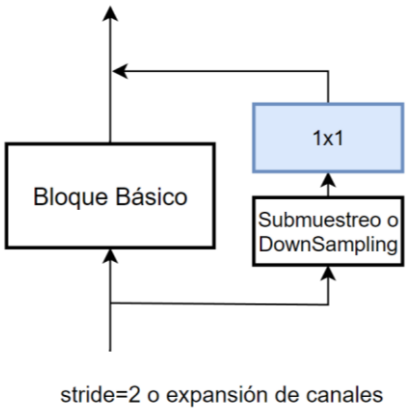

Figura 28: Diseño de conexión residual para primer bloque de etapa.

Es importante considerar el costo computacional asociado con el uso de estas conexiones residuales. Dependiendo del hardware objetivo, es posible que se eliminen estas conexiones residuales y se mantengan solo las conexiones

residuales base utilizadas para bloques que tengan una cantidad de canales de entrada y salida similar. Estos suelen ser los segundos bloques en adelante de cada etapa de la red.

A continuación, se detallarán los tipos de bloques básicos a utilizar.

3.3.1.2 Bloques Básicos

MBV1-Block

El bloque MBV1 (MobileNetV1 Block) se basa en los bloques utilizados en MobileNetV1 [Howard et al., 2017]. Este bloque consta de dos partes principales:

1. **Convolución Depthwise:** En esta etapa, se aplica una convolución 3x3 con un número de canales de entrada y salida similares. Además, se utiliza un grupo que es igual al número de canales de entrada, lo que significa que cada canal de entrada se somete a su propia convolución interna con su correspondiente canal de salida. Luego, los resultados de estas convoluciones se concatenan para obtener los canales de salida finales. Este proceso se conoce como *Depthwise Convolution*.
2. **Convolución Pointwise:** Se aplica una convolución 1x1 que ajusta la salida del bloque a la cantidad deseada de canales de salida. La relación entre los canales de entrada y salida puede variar según la configuración del bloque.

Este bloque es útil para dispositivos móviles y de recursos limitados. También es importante recalcar que su versión de MobileNetV1 0.25 fue utilizada como extractor de características para RetinaFace [Deng et al., 2019] en su versión de detector para dispositivos embebidos. Las variaciones posibles para este bloque son los posibles tipos de kernel {3x3, 5x5}. Su arquitectura general como bloque sería la siguiente.

El bloque MBV1 fue diseñado para dispositivos móviles y sistemas con recursos computacionales limitados debido a su eficiencia computacional. Es importante destacar que incluso una versión más ligera de MobileNetV1 [Howard et al., 2017], conocida como MobileNetV1 0.25, se ha utilizado con buenos resultados como extractor de características en la implementación de RetinaFace [Deng et al., 2019] destinada a dispositivos embebidos.

En esta tesis, se tiene la flexibilidad de emplear dos tamaños de kernel diferentes para este bloque, específicamente 3x3 o 5x5, en base a las necesidades particulares del usuario. Esto permite ajustar el espacio de búsqueda para diferentes aplicaciones. La arquitectura general del bloque MBV1 se ilustra en la Figura 29.

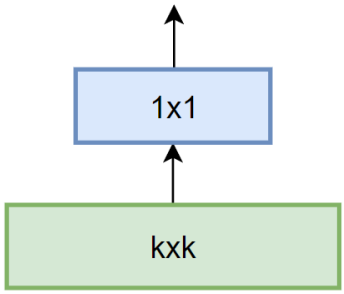

Figura 29: Arquitectura general como bloque para MBV1.

MBV2-Block

El MBV2-Block se compone de las Inverted Residual creadas en la MobileNetV2 [Sandler et al., 2018]. Se pueden configurar con diferentes valores de expansión (r = {3, 4, 6}) y tamaños de kernel (k = {3x3, 5x5}). Su arquitectura como bloque básico se muestra en la Figura 30.

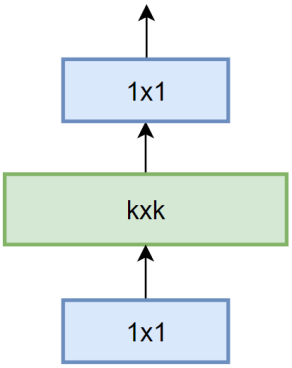

Figura 30: Arquitectura general como bloque para MBV2.

Estos bloques han demostrado un gran potencial debido a sus excelentes resultados en las MobileNets [Sandler et al., 2018] y su uso en redes actuales de gran desempeño como la EfficientNet [Tan & Le, 2019].

BL-Block

Los BL-Block son los bloques tipo BottleNeck pertenecientes a las ResNet [He et al., 2016]. En este caso su variación de expansión se fija en r=1/4 y los tamaños de kernel disponibles son k={3x3,5x5}. La arquitectura general de este bloque se muestra en la Figura 31.

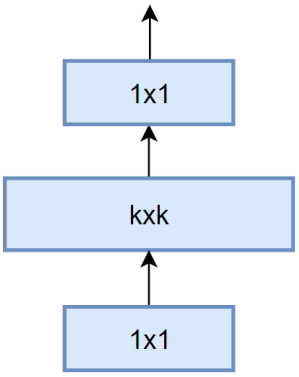

Figura 31: Arquitectura general como bloque de BL.

Este tipo de bloques permite utilizar una mayor cantidad de canales en la red sin que esto genere un alto costo computacional debido a su diseño y compresión en la etapa intermedia.

KK-Block

Los KK-Block provienen del diseño realizado como bloque básico en el paper de GENet [Lin et al., 2020]. Estos bloques consisten en dos convoluciones básicas secuenciales con tamaños de kernel por definir. Sin embargo, debido al alto costo computacional de estos bloques, se permite una variación que solo utiliza la mitad de este bloque o, en otras palabras, solo una convolución directa llamado KK_Mid-Block. Las arquitecturas generales de estos bloques se muestran en la Figura 32.

Figura 32: Arquitectura general como bloque de KK_Mid y KK.

Las variaciones posibles para estos bloques incluyen el tamaño del kernel, que es {3x3}, y la opción de utilizar el bloque completo o la mitad, representada como mid = {True, False}. Es importante destacar que en este caso se elimina la variación de kernel 5x5 debido a su elevado costo computacional.

DK-Block

Los DK-Block pertenecen a los bloques de DarkNet [Redmon & Farhadi, 2018] y consisten en una convolución de kernel 1x1 seguida de una convolución de kernel 3x3. Su arquitectura general se muestra en la Figura 33.

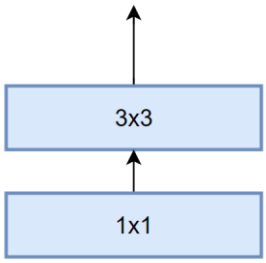

Figura 33: Arquitectura general como bloque de DK.

Debido a su diseño, el tamaño del kernel se fija el kernel en 3x3 y no existe mayor variación. Sin embargo, si este bloque tiene un stride de valor 2, se modifica el bloque reemplazándolo por una convolución directa, es decir, se elimina el primer bloque de kernel 1x1 utilizado y solo se mantiene la siguiente convolución. Esto se realiza para mantener el diseño utilizado en la red de detección YOLO [Redmon & Farhadi, 2018].

3.3.1.3 Complementos a Bloques SE y eSE

Los complementos SE (Squeeze-and-Excitation) [Hu et al., 2018] y eSE (Effective Squeeze-and-Excitation) [Lee & Park, 2020] se aplicarán a los bloques básicos en diferentes etapas de la arquitectura. Estos complementos se implementarán según los experimentos realizados por TResNet [Ridnik et al., 2020], que sugieren que, para optimizar la precisión y el costo computacional, solo se utilizarán en las dos primeras etapas de la red a crear mediante el método NAS.

Para los bloques básicos, la implementación de SE [Hu et al., 2018] o eSE [Lee & Park, 2020] se realizará en las siguientes partes del bloque:

- **MBV1-Block:** Se aplicará después de la primera convolución, que pertenece a la convolución Depthwise.
- **MBV2-Block:** Al igual que en el caso anterior, se aplicará después de la convolución Depthwise, que en este caso corresponde a implementar el SE [Hu et al., 2018] o eSE [Lee & Park, 2020] posterior a la segunda convolución.
- **BL-Block:** Para la BL-Block, se aplicará después de la segunda convolución, que implica implementarlo luego de la etapa de compresión en donde los canales se dividen en 4.
- **KK-Block:** En este caso, se aplicará al final de la segunda convolución. Si se utiliza la variante *mid* (solo uno de los dos bloques convolucionales), no se aplicará SE [Hu et al., 2018] o eSE [Lee & Park, 2020].
- **DK-Block:** De manera similar al caso anterior, este se aplicará posterior a la segunda convolución. Sin embargo, si se trata de un bloque con un valor de stride igual a 2 y, por lo tanto, solo consta de una convolución, no se aplicará el uso de SE [Hu et al., 2018] o eSE [Lee & Park, 2020].

El uso de SE [Hu et al., 2018] o eSE [Lee & Park, 2020] se limitará a los bloques que tengan el mismo canal de entrada que el de salida, similar a la limitación para las conexiones residuales base.

3.3.1.4 Nomenclatura

La nomenclatura de cada bloque básico es fundamental para diferenciarlos y referenciarlos de manera única. En general, esta nomenclatura se basa en varios elementos clave, como el tipo de kernel, la expansión r (cuando corresponda), los canales de entrada y salida, y si se utiliza la variante Mid, como en el caso de las KK-Block.

Además, se incorporan indicadores como *S* o *SE* para señalar si el bloque contiene un stride de valor 2 o si incluye las transformaciones Squeeze-and-Excitation (SE) [Hu et al., 2018] o Effective Squeeze-and-Excitation (eSE) [Lee & Park, 2020].

La estructura general de la nomenclatura se compone de la siguiente manera:

[*S*]_[*SE*]_[*k*]x[*k*]BloqueBásico[*r*]_[*Mid*]_[*channel_out*]_[*input*]

- Los valores *k* hacen referencia al tamaño del kernel del bloque.
- *r* se incluye cuando corresponde según la expansión del bloque, que se aplica en los BL-Block y los MBV2-Block.
- *Mid* se agrega a la nomenclatura para indicar la variante en los KK-Block, cuando se utiliza solo una convolución.
- *channel_out* se refiere directamente a los canales de salida del bloque.
- *input* representa la dimensionalidad de entrada al bloque, incluyendo los 3 canales, siendo el primero el canal de entrada, seguido de dos canales adicionales que dependen de la cantidad de stride 2 o reducción de dimensionalidad aplicados y las dimensiones de la imagen o entrada al comienzo de la red.

La Tabla 18 proporciona un resumen de las nomenclaturas mencionadas para cada uno de los bloques básicos con los que se trabajará y sus posibles variaciones, considerando que se aplica S_SE como base para generalizar al igual que Mid en las KK-Block.

Tabla 18: Resumen de bloques básicos y sus variaciones.

Bloque Básico	Variaciones de Kernel	Variaciones Específicas	Nomenclatura general para Bloque
MBV1-Block	3x3, 5x5	-	S_SE_kxk_MBV1Conv_channelout_input
MBV2-Block	3x3, 5x5	Expansión r = 3, 4, 6	S_SE_kxk_ MBConv{3,4,6}_channelout_input

BL-Block	3x3, 5x5	Expansión 1/r = 1/4	S_SE_kxk_BottleNeckConv{4}_channelout_input
KK-Block	3x3	Variación Mid	S_SE_kxk_KKConv_Mid_channelout_input
DK-Block	3x3	-	S_SE_kxk_DKConv_channelout_input

3.3.2 MixedEdge

Existen cinco tipos de bloques básicos a utilizar. Para el diseño interno del método NAS, se crea un bloque principal que contiene estos cinco bloques básicos, llamado MixedEdge. Este se compone de dos factores principales, los cuales son:

- **Bloque Básico:** Este bloque puede estar compuesto por varios tipos de bloques básicos, como el MBV1-Block, MBV2-Block, BL-Block, KK-Block, DK-Block y Zero. El bloque Zero es un bloque identidad que indica que no se debe utilizar ningún bloque básico en esa capa de la red. El MixedEdge puede estar compuesto de solo algunos de estos bloques básicos, definidos por el humano experto. Por ejemplo, es posible agregar solo el bloque KK-Block, lo que permitirá variar entre el KK-Block y el bloque vacío (Zero) en el MixedEdge a excepción de los bloques mínimos que debe contener la arquitectura (para la conexión con la FPN [Lin et al., 2017] que en este caso su espacio de búsqueda queda de solo KK-Block.
- **Conexión Residual:** Corresponde al tipo de conexión residual a utilizar. Como se detalla en la sección 3.3.1.1, esta conexión residual puede ser del tipo Identity (conexión residual directa) o del tipo 1x1_Conv, con su correspondiente operación de Average Pool. La conexión residual de nomenclatura 1x1_Conv se aplica solo en los cambios de etapa del extractor de características para optimizar el costo computacional que este conlleva. También existe la posibilidad de no utilizar conexión residual (None), pero esto será fijado en los hiperparámetros por el humano experto dependiendo del hardware objetivo.

El diseño interno del MixedEdge con los dos factores principales de los que se compone, se ilustra en la Figura 34.

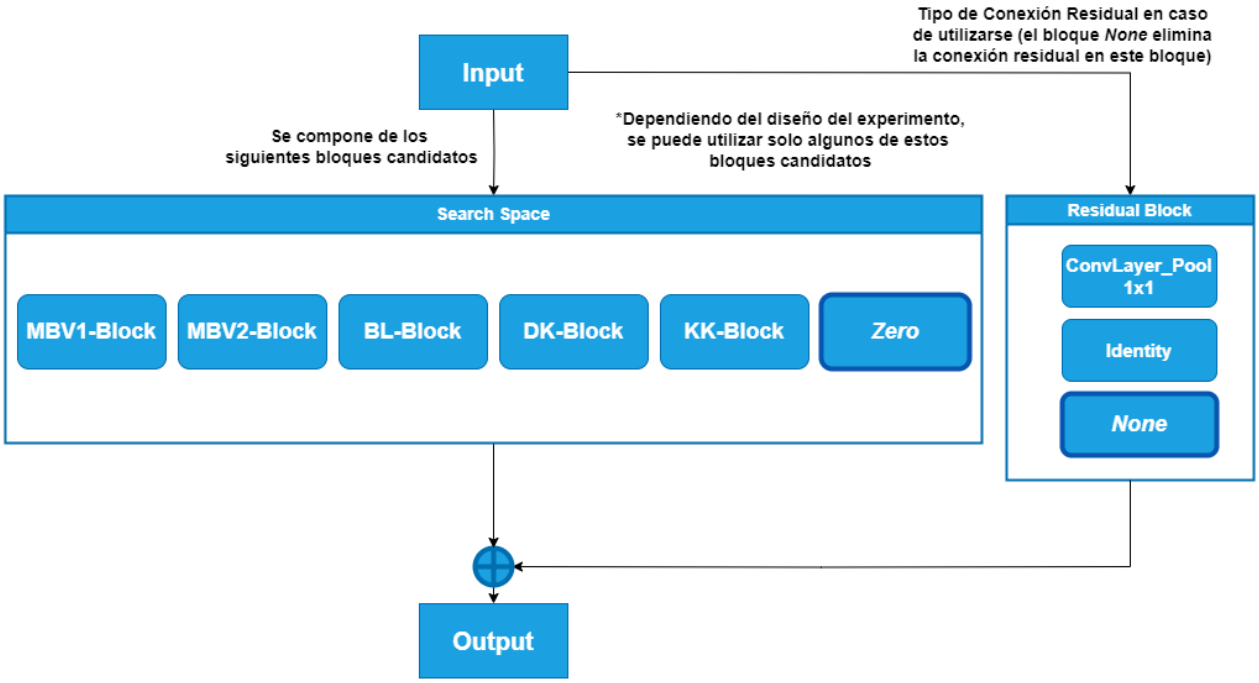

Figura 34: Composición interna de MixedEdge.

3.3.3 Diseño de Arquitectura NAS Principal

La arquitectura del extractor de características se asemeja en gran medida a la estructura general de una ResNet [He et al., 2016], ya que cuenta con una convolución de entrada y luego sigue con cuatro etapas de distintas convoluciones. Sin embargo, se diferencia en el uso del bloque interno llamado MixedEdge en cada capa de las etapas de la red, así como una capa de salida convolucional de 1x1 que actúa como un conector hacia el último nivel de la FPN [Lin et al., 2017]. Las etapas 2 y 3 del extractor de características también actúan como conector, en este caso para el primer y segundo nivel de la FPN [Lin et al., 2017], respectivamente.

En la parte inicial de la red, las convoluciones de entrada comienzan con una capa de entrada de 3 canales, seguida por una capa de MixedEdge. La configuración de esta capa MixedEdge puede ser fijada por un humano experto a un tipo de bloque en particular, lo que permite que la búsqueda se centre en las cuatro etapas siguientes de la arquitectura.

La arquitectura el extractor de características NAS se puede observar en la Figura 35.

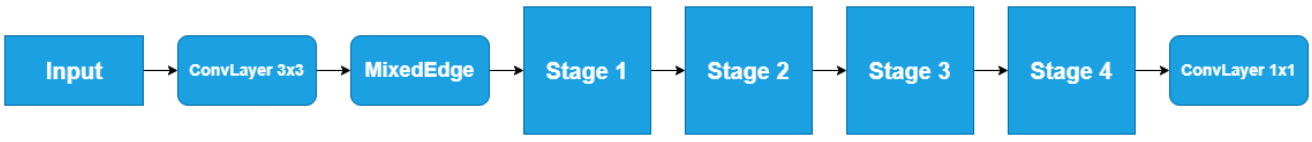

Figura 35: Arquitectura completa del Extractor de características NAS.

Dentro de cada una de estas cuatro etapas, hay un número N de bloques MixedEdge y un número de canales de entrada C. Ambos parámetros son definidos por el humano experto, en colaboración con el espacio de búsqueda que se define para cada etapa. En este espacio de búsqueda, se pueden especificar qué tipo de bloques básicos se utilizarán en las etapas correspondientes. La arquitectura se asemeja a la que se muestra en la Figura 36. Esto proporciona flexibilidad para ajustar la arquitectura de la red de acuerdo con las necesidades específicas y las limitaciones de hardware.

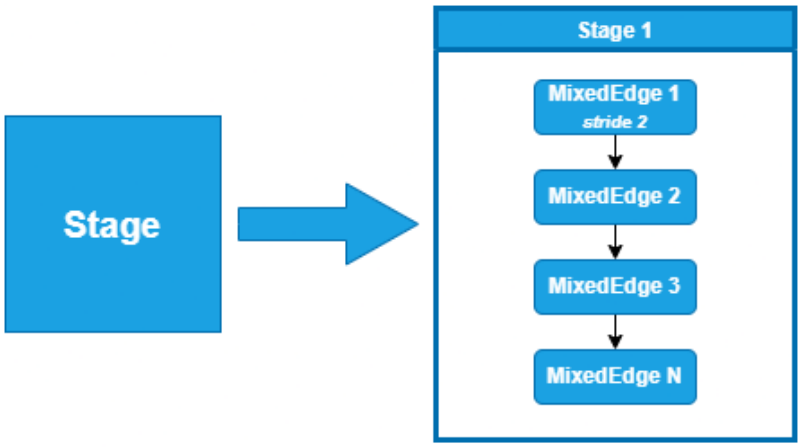

Figura 36: Composición interna de un Stage.

En cada una de estas etapas, se incorpora un bloque fijo que no puede ser reemplazado por un bloque de tipo Zero (bloque vacío o identidad). Esta característica proporciona una arquitectura base que permite la conexión con la FPN [Lin et al., 2017]. Este bloque particular contendrá la operación de stride 2 si es necesario en esa etapa y ajustará el número de canales de acuerdo con la variación respecto a la etapa anterior. Por ejemplo, si la etapa anterior tenía 32 canales y esta nueva etapa requiere 64 canales, el primer bloque tendrá 32 canales de entrada y 64 canales de salida, y luego se mantendrán los 64 canales para el resto de los bloques de esa etapa.

3.3.3.1 Integración de variación MaxPool similar a ResNet

Se ha implementado la opción de replicar las primeras etapas de reducción de dimensionalidad similar a lo que se encuentra en varias redes como la familia de las ResNet [He et al., 2016] y sus variantes. Para esto, se ha introducido una variable que, cuando se activa, cambia el primer bloque de la Etapa 1 de stride 2 a stride 1, y se agrega una operación MaxPool de 3x3 con un stride de 2 antes de este bloque. Esta operación de MaxPool se utiliza para reducir la dimensionalidad y, si bien ahorra costos computacionales, inicialmente puede resultar en una ligera disminución de la precisión. La razón es que esta operación es fija y no se adapta a los datos como lo harían los pesos de una convolución con stride 2. La ilustración gráfica del funcionamiento con esta variación activada se muestra en la Figura 37.

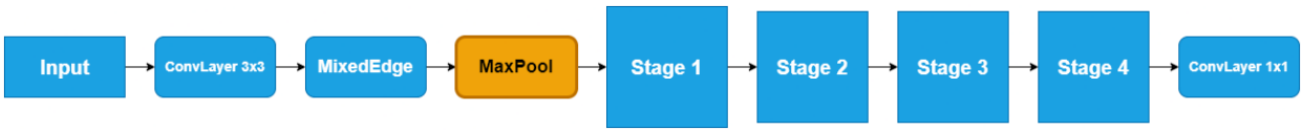

Figura 37: Arquitectura completa del Extractor de características NAS con variación de MaxPool.

3.3.3.2 Variación de bloques iniciales por diseño de ResNet50d

Inicialmente, como se aprecia en la Figura 37, en un inicio de la arquitectura del extractor de características NAS se emplea una convolución de 3x3, seguido de un bloque MixedEdge que está compuesto de solo una convolución 3x3 (lo que es equivalente a usar directamente una convolución 3x3). Sin embargo, tomando como referencia los experimentos y mejoras presentados en el artículo mencionado en la sección 2.1.2 sobre ResNet-D [He et al., 2018], se modificará el valor utilizado en el MixedEdge de convolución 3x3, reemplazándolo por un KK-Block que contendrá las dos convoluciones de 3x3 utilizadas en dicho trabajo.

3.4 Función de Pérdida

La función de pérdida se compone de dos factores. Por un lado, se encuentra la función de pérdida relacionada a la precisión del modelo neuronal, mientras que, por otro lado, se tiene la función de pérdida basada en la latencia de referencia para el modelo. A continuación, se explican ambas funciones en detalle.

3.4.1 Función de costo complementada

La función de costo, que combina ambas funciones (tanto precisión como latencia), se formula de la siguiente manera:

$$L_{total} = L_{train} + L_{lat} \tag{6}$$

Para comprender cómo se desglosan tanto la función de pérdida de precisión como la de latencia, se presentan los detalles en las secciones siguientes.

3.4.1 Función de pérdida en precisión

La función de pérdida en precisión se compone de la misma manera que la utilizada en RetinaFace [Deng et al., 2019], esta se constituye de los siguientes elementos:

$$L_{train} = L_{cls} + \alpha * L_{bbox} + L_{land} \tag{7}$$

$\boldsymbol{L_{cls}}$**:** Este componente se calcula a partir de la clasificación de los rostros utilizando la función de pérdida Cross Entropy.

$\boldsymbol{L_{bbox}}$**:** Para los puntos de los bounding box obtenidos y los valores reales de la base de datos, se utiliza la función de pérdida suavizada Smooth L1. El Smooth L1 es una versión suavizada de la función de pérdida L1, que mide la diferencia absoluta entre el valor obtenido y el valor esperado. La ventaja de utilizar esta versión suavizada radica en su menor sensibilidad a valores atípicos u outliers que puedan encontrarse en los datos.

Alpha (*α*)**:** Este valor es configurable y se utiliza para enfocar la pérdida de entrenamiento, lo que permite asignar una mayor importancia a la localización del bounding box o a la clasificación de los rostros. En este trabajo, se fija en 2.0, manteniendo la configuración utilizada en el trabajo de RetinaFace [Deng et al., 2019].

$\boldsymbol{L_{land}}$**:** Al igual que en Lbbox, se utiliza Smooth L1 para los puntos de landmarks esperados en comparación con los puntos obtenidos.

3.4.2 Función de pérdida en latencia

La función de pérdida de latencia experimentó varias iteraciones durante los experimentos realizados. Inicialmente, se utilizaba la función de pérdida empleada en el trabajo de ProxylessNas [Cai et al., 2018], que se define de la siguiente manera:

$$L_{lat} = \frac{\lambda * (Latencia_{esperada} - Latencia_{referencia})}{Latencia_{referencia}} \quad (8)$$

A continuación, se explican los conceptos clave de cada valor en esta fórmula:

Lambda (*λ*): Este valor permite ajustar la importancia relativa de la función de pérdida de latencia en comparación con la función de pérdida de precisión de la red. También se utiliza para controlar cuánto se penaliza la diferencia entre la latencia de referencia definida por el usuario y la latencia actual del modelo.

$\boldsymbol{Latencia_{esperada}}$**:** Este valor corresponde a la suma de las latencias de cada bloque activo seleccionado por la red en el momento en que se calcula esta función de pérdida.

$\boldsymbol{Latencia_{referencia}}$**:** Se refiere a la latencia inicialmente definida para el método NAS. El objetivo del método NAS es acercarse o converger hacia este valor durante su búsqueda.

La función de pérdida original generaba problemas al principio, ya que, si la latencia esperada resultaba ser menor que la de referencia, se obtenía un valor negativo, lo que contrarrestaba la función de pérdida de precisión. Esto llevaba a que el método siempre seleccionara la red más simple, dejando todos los bloques activos como Zero (vacío) y,

en consecuencia, no lograba maximizar la precisión, ya que siempre se centraba en minimizar la latencia a expensas de la precisión.

Para abordar este problema, se realizó una modificación en la función de pérdida, resultando en la siguiente fórmula:

$$L_{lat} = \text{Max}(\frac{\lambda * \left(Latencia_{esperada} - Latencia_{referencia}\right)}{Latencia_{referencia}}, 0) \quad (9)$$

Con este cambio, la red ya no buscaba minimizar o eliminar bloques de la red neuronal para contrarrestar la pérdida, ya que, al llegar al punto negativo, el valor se mantenía en cero. Esta modificación permitió que la red iterara de manera más efectiva en la búsqueda de bloques que mejoraran la precisión sin converger únicamente en el esqueleto básico de la red de búsqueda NAS.

Sin embargo, esta modificación aún no lograba penalizar adecuadamente el método por utilizar menos bloques y no aprovechar al máximo el valor de referencia para maximizar la precisión. Como resultado, se realizaron más iteraciones en la función de pérdida de latencia, obteniendo finalmente la siguiente fórmula:

$$L_{lat} = |\frac{\lambda * \left(Latencia_{esperada} - Latencia_{referencia}\right)}{Latencia_{referencia}}| \quad (10)$$

En esta versión final, se reemplaza el valor máximo entre la fórmula interna y cero por el valor absoluto de la fórmula interna. De esta manera, el método siempre busca acercarse al valor de la latencia de referencia, independientemente de si la latencia esperada es mayor o menor. Esto permite controlar el grado de penalización que se aplica a la red en función de su diferencia con la latencia de referencia, basándose únicamente en el valor de λ, que indica cuán costoso resulta alejarse de la latencia de referencia.

Con la fórmula de pérdida definida de esta manera, se realizó otra modificación en la forma de calcular la latencia esperada en comparación con el enfoque utilizado en el trabajo de ProxylessNas [Cai et al., 2018]. Inicialmente, se tenía una fórmula que realizaba una suma ponderada de cada bloque candidato de cada capa, con la suma total de estos ponderadores igual a 1, como se muestra en la ecuación 11.

$$E[Latency] = \alpha * F(conv_{3x3}) + \beta * F(conv_{5x5}) + \sigma * F(identity) + \dots + \delta * F(pool_{3x3}) \quad (11)$$

$$E[Latency] = \sum_i E[latency_i]$$

Diversas pruebas han demostrado que la elección del bloque activo no es siempre óptima, en especial cuando se enfrenta a un espacio de búsqueda extenso, donde hay numerosas opciones de bloques básicos para elegir. Esto se debe a que la ponderación y el valor asignado al bloque activo tienden a ser bajos en tales escenarios.

Para abordar esta situación, se implementa una modificación y se aplica la siguiente fórmula:

$$Latencia_{ponderada} = alpha * F(b1) + betha * F(b2) + \cdots$$

$$Latencia_{bloque\ activo} = betha * F(b2)$$

$$E[latencia_{bloque\ i}] = latencia_{ponderada} * p1 + latencia_{bloque\ activo} * (1 - p1) \quad (12)$$

$$E[latencia] = \sum_{i=1}^{n} E[latencia_{bloque\ i}]$$

Como resultado de esta modificación, se ha incrementado la importancia del bloque activo en el proceso de selección. Esto se logra al considerar no solo la latencia ponderada, sino también la latencia directa del bloque activo, ambas ponderadas para obtener un valor final. Después de llevar a cabo una serie de experimentos, se ha determinado que la elección de un valor de 0.5 tanto para la latencia ponderada como para la latencia del bloque activo (valor fijado en p1) produce los mejores resultados en comparación con otras variaciones, como 0.6 y 0.4 o 0.7 y 0.3, por ejemplo.

3.5 Data Augmentation

Para llevar a cabo el entrenamiento tanto durante la ejecución del método NAS como en la fase final de entrenamiento (realizada en la red ya encontrada por el método NAS),

se emplea una técnica llamada Data Augmentation. Esta técnica tiene como objetivo incrementar la cantidad de información disponible para la red neuronal, utilizando la misma base de datos. Las modificaciones o transformaciones realizadas en las imágenes pueden ser constantes (aplicadas de manera regular) o probabilísticas (con una cierta probabilidad de aplicación). Durante el entrenamiento, se aplican las siguientes transformaciones:

- **Crop:** Se realiza un recorte aleatorio en una región de la imagen que puede variar entre el 30%, 45%, 60%, 80% y 100% de la escala de la imagen original. Si este recorte no resulta en al menos una cara visible de 16x16 píxeles, se repite el proceso de recorte hasta que se cumpla este criterio.
- **Distort:** En este caso, la distorsión se refiere a la modificación del brillo, contraste, saturación y tono (hue) de la imagen. Cada una de estas modificaciones tiene una probabilidad del 50% de ser aplicada.
- **Mirror:** La imagen se refleja horizontalmente con una probabilidad del 50%.
- **Pad to square:** Se agrega un relleno de píxeles negros a la imagen para que esta tenga una resolución cuadrada, manteniendo el lado más largo y expandiendo el otro.
- **Resize Subtract Mean:** La imagen se redimensiona a una resolución específica definida en los hiperparámetros de entrenamiento. Luego, se aplica una normalización sobre la imagen. Este proceso es comúnmente utilizado en entrenamientos tanto para la detección como para la clasificación de objetos.

En la etapa de validación, se aplican solo las siguientes modificaciones:

- **Mirror:** También se aplica con una probabilidad del 50%.
- **Resize Subtract Mean.**

Se utiliza un conjunto más limitado de transformaciones en las imágenes de validación debido a que variaciones excesivas podrían dificultar la evaluación precisa de la red neuronal en comparación con iteraciones anteriores.

El empleo de esta técnica de Data Augmentation permite que la red neuronal adquiera una mayor diversidad de información y aprenda características más sólidas e invariantes frente a nuevas imágenes que puedan presentarse en diversas situaciones y condiciones, lo que a su vez mejora la precisión de la red.

3.6 Obtención de Latencias y Diccionario

La obtención de los atributos de latencia para cada bloque que conforma el espacio de búsqueda en cada iteración del método NAS propuesto no es una estrategia óptima, ya que esto incrementa significativamente el tiempo requerido, extendiéndolo a varios días en términos del uso de recursos de GPU. Además, en el escenario en el que el hardware objetivo no coincide con el utilizado durante la búsqueda de la arquitectura NAS, obtener estos valores de latencia sería aún más complicado y podría ralentizar

significativamente el proceso, ya que requeriría una conexión entre ambos entornos para obtener la latencia de los bloques, lo que agregaría complejidad al método.

Por esta razón, se ha implementado una solución basada en un diccionario JSON que almacena los atributos de cada bloque, específicamente el atributo relacionado con la latencia para un hardware en particular.

Sin embargo, la cantidad de bloques a iterar para obtener sus atributos en un espacio de búsqueda grande podría ralentizar la obtención de todas las combinaciones posibles, especialmente en hardware con recursos limitados. Por lo tanto, se han desarrollado distintas estrategias para abordar tanto el caso en el que se realiza la búsqueda de la arquitectura NAS en el mismo hardware objetivo como el caso en el que se realiza en un hardware diferente, con el objetivo de optimizar y simplificar esta tarea.

3.6.1 Obtención de Diccionario en hardware objetivo interno

Cuando el hardware objetivo coincide con el hardware utilizado para la búsqueda, se permite obtener el atributo de latencia para cada bloque durante el mismo proceso del método NAS propuesto. Estos valores de latencia se almacenan en un diccionario JSON, lo que permite disponer de ellos en futuras iteraciones sobre este mismo hardware.

El funcionamiento de este método interno se basa en la búsqueda de los atributos de los bloques utilizados en la iteración actual del proceso de búsqueda de la arquitectura NAS. Si el valor de latencia para el bloque actual aún no existe en el diccionario, se procede a medir la latencia del bloque. Una vez obtenida, esta información se guarda en el diccionario y se continúa con el siguiente bloque. Se verifica si el atributo del bloque siguiente ya se encuentra en el diccionario o no, y se repite este proceso hasta que todos los bloques utilizados en la iteración actual cuenten con sus atributos de latencia registrados. De esta manera, solo se obtienen los atributos de los bloques que se están utilizando durante esta iteración, optimizando la obtención de estos valores y generando un diccionario para futuras iteraciones y experimentos.

3.6.2 Obtención de Diccionario en hardware objetivo externo

En el escenario en el que el hardware objetivo difiere del hardware utilizado durante la búsqueda y el entrenamiento, se requiere la creación de un script flexible que pueda ser ejecutado en el hardware objetivo. Este script permite modificar una serie de parámetros que definen el espacio de búsqueda, con el fin de generar un diccionario JSON que contenga información detallada sobre los atributos de los bloques. La ventaja de esta flexibilidad de parámetros es que se evita la obtención de atributos para bloques que no se utilicen en el hardware objetivo, lo que optimiza el proceso de generación del diccionario.

Los hiperparámetros que se pueden ajustar en este script para la generación del diccionario JSON incluyen:

- **Bloques Básicos:** Se especifica qué tipo de bloque básico se desea obtener y las posibles variaciones que puede tener, como variaciones de kernel, expansión *r* o la variación *Mid* en el caso de los bloques KK-Block. Esto también incluye componentes de bloque residual para el primer bloque de cada etapa, como se mencionó en el punto 3.3.2.
- **Batch size:** Se define el tamaño del batch con el que se creará el tensor de entrada para el bloque.
- **Variaciones complementarias sobre los bloques básicos:** Consiste en el uso o no uso de SE [Hu et al., 2018] o eSE [Lee & Park, 2020] según se busque utilizar.
- **Tipo de Hardware:** En caso de que el hardware objetivo cuente con GPU, se especifica si se busca obtener una arquitectura de red neuronal optimizada para su CPU o GPU y en base a esto obtener las latencias de cada bloque.
- **Strides:** Se indica si se desea aplicar variaciones en los strides utilizados en los bloques. Normalmente, se consideran las variaciones de valores {1,2}.
- **Cantidad de etapas**: Aunque por defecto se utilizan 4 etapas para la conexión con la arquitectura FPN [Lin et al., 2017] y para las reducciones de dimensionalidad, se permite agregar etapas intermedias que, aunque no incluyan reducción de dimensionalidad, pueden usarse para introducir variaciones en los canales intermedios.
- **Canales de entrada para cada etapa:** Se especifica la variación de canales que se desea utilizar para cada etapa de la red neuronal. Esto se logra al definir listas de canales para la entrada y la salida de cada etapa.
- **Canales para bloque inicial:** El primer bloque, que recibe el tensor de entrada, puede tener diversas variaciones en sus canales de salida. Estas variaciones dependen de la complejidad o el tamaño de la red que se pretende construir y pueden variar desde 8 canales (similar a MobileNetV1 0.25) hasta 64 canales de salida (similar a ResNet50 [He et al., 2016]).
- **Canales para bloque final:** El bloque final que conecta la última etapa de FPN [Lin et al., 2017], consiste en una convolución de kernel 1x1. Recibe como entrada la lista de canales de la última etapa y tiene una salida que depende del hardware objetivo. Los valores típicos son 2560 para DGX, 256 para Raspberry Pi 4 y 256 canales para Jetson Nano.
- **Resolución de entrada de imagen:** Se especifica la resolución de la imagen o tensor de entrada que se utilizará, de esta manera variar la dimensionalidad que pueden tener estos bloques. Esta dimensionalidad dependerá de su etapa en la que se encuentre el bloque y su cantidad de canales correspondiente. La dimensionalidad se compone de la siguiente manera:

$$(\text{batch\_size}, \frac{resolución\ de\ imagen}{2^{\text{etapa}+1}}, \frac{resolución\ de\ imagen}{2^{\text{etapa}+1}})$$

Para optimizar la obtención de las latencias, se han implementado importantes funcionalidades en el proceso de generación del diccionario de atributos de los bloques. Estas funcionalidades permiten ingresar nomenclaturas directas o listas de nomenclaturas al script, lo que posibilita obtener el atributo de latencia de uno o varios bloques específicos sin necesidad de realizar un espacio de búsqueda completo para obtenerlos.

Además, se aplican filtros que permiten omitir ciertas combinaciones de bloques que carecen de factibilidad de implementación o elección durante la búsqueda. Estos filtros son los siguientes:

- **Umbral de latencia:** Se verifica si la latencia de un bloque supera cierto umbral. Esto se hace teniendo en cuenta que en algunos hardware el recurso computacional puede ser limitado, lo que podría generar configuraciones de bloques muy costosas en términos de latencia. Si el promedio de las primeras 5 muestras de latencia (después del período de calentamiento o warm-up para evitar valores atípicos) supera el umbral fijado por el humano experto, se omite ese bloque y se continúa con el siguiente.
- **Combinaciones atípicas:** Se evitan combinaciones de canales de entrada y salida que son atípicas según los estándares actuales de diseño de redes neuronales. Por ejemplo, si el canal de entrada es 16 y el canal de salida es 128, esto podría considerarse una combinación poco lógica (debido a una expansión de 4 veces los canales de la etapa anterior), y se omite para evitar la generación de configuraciones de bloques que no se utilizarán en el diseño final de la red. De igual manera, este valor es configurable por el humano experto para darle mayor libertad en caso de querer incluir estas combinaciones.

Toda esta información, además de ser registrada en el diccionario JSON de atributos de bloques, se almacena en un archivo de registro de texto (logs). Esto permite analizar los datos obtenidos, así como los bloques que fueron omitidos por los filtros y las razones detrás de esas omisiones.

Para garantizar la integridad de los datos y evitar interferencias en la toma de valores de latencia, especialmente en hardware más limitado, se ha incorporado un tiempo de descanso después de cada cambio de bloque básico y su conjunto correspondiente. Este período de descanso ayuda a aliviar la carga en la CPU, reducir la temperatura y garantizar que no se produzcan perturbaciones en la toma de valores de latencia, lo que asegura la calidad de los datos recopilados.

3.6.3 Mecánica de obtención de latencia

La mecánica para obtener la latencia de cada bloque de manera precisa se realiza siguiendo los siguientes pasos:

1. Se ingresa la nomenclatura del bloque para el cual se desea obtener la latencia. Esta nomenclatura descrita en la sección 3.3.1.4 contiene toda la información necesaria para saber los parámetros y cómo cargar este bloque.
2. A la nomenclatura se le incluye la dimensionalidad del tensor de entrada requerido para el bloque. Se crea este tensor con valores aleatorios utilizando la función torch.randn().
3. Dependiendo del hardware objetivo (CPU o GPU), se carga el bloque y el tensor de entrada en el dispositivo correspondiente.
4. Con ambos componentes listos, se realizan inferencias como parte de un proceso de calentamiento (warm-up) para el bloque. Esto se hace para eliminar los primeros resultados de latencia, que suelen ser más lentos debido a la optimización interna que realiza PyTorch para el bloque en específico.
5. Después de los procesos de calentamiento, se realizan inferencias adicionales del bloque con el tensor de entrada, y se registran los tiempos de latencia en una lista de valores numpy. La toma de latencia se realiza midiendo el tiempo inicial, realizando la inferencia del bloque con el tensor de entrada y midiendo el tiempo final. Estos tiempos están sincronizados con la GPU en caso de que se esté utilizando, lo que proporciona una medición más precisa.
6. Para obtener un promedio de latencia más robusto y eliminar valores atípicos, se ordena la lista de tiempos de latencia y se descartan el 10% de los valores más bajos y el 10% de los valores más altos.

Los valores específicos de warm-up y la cantidad de muestras para obtener la latencia varían según si el hardware objetivo es una CPU o una GPU, los valores son los mostrados en la Tabla 19.

Tabla 19: Cantidad de Warm-Up y muestreo por tipo de hardware.

Hardware	Warm-Up	Muestreo
GPU	40	200
CPU	20	50

Es relevante mencionar que para las pruebas en CPU se utiliza un batch size de 1, lo que significa que las inferencias se realizan para una sola entrada a la vez. Por otro lado, para las pruebas en GPU se utiliza un batch size de 8 como referencia. Esto se debe a que en GPU es común realizar inferencias para múltiples entradas al mismo tiempo, lo que proporciona una medida de escalabilidad base para el bloque. Esta elección del batch size también se alinea con las prácticas comunes en el área y permite comparar los resultados con una referencia común.

3.7 Hiperparámetros

El método NAS propuesto es semi-automático, ya que necesita de definiciones hechas por el humano experto para algunas de las configuraciones. Esto genera sus ventajas, ya que logra acotar ciertos parámetros que pueden complejizar el método más de lo necesario.

A continuación, se describen los hiperparámetros que deben ser definidos por el humano experto:

- **Tipo de Bloque(s):** Se elige el tipo de bloque(s) que se utilizará en la arquitectura. Esto puede incluir bloques como KK-Block, BL-Block, DK-Block, MBV1-Block y MBV2-Block, junto con sus respectivas configuraciones. Estas elecciones determinan el espacio de búsqueda para cada una de las etapas de la red NAS.
- **Canales en cada Etapa:** Se especifica la cantidad de canales que tendrá cada una de las cuatro etapas de la red. Estas etapas se encuentran conectadas entre sí, lo que significa que el valor de entrada del primer bloque de cada etapa proviene de la etapa anterior, y el valor de salida se convierte en el canal de entrada para la etapa actual.
- **Cantidad de Bloques máximo por Etapa:** Indica la cantidad máxima de bloques que se permitirán por cada etapa. Esto permite dar mayor margen de utilización a ciertas etapas en caso de ser necesario, ayudando a la búsqueda NAS del método.
- **Latencia de Referencia:** Este valor sirve como referencia para la latencia que se buscará en la arquitectura NAS. Ayuda a optimizar la red para maximizar la precisión dentro del límite de latencia establecido.
- **GPU por utilizar:** En el caso de que el hardware en donde se realice la búsqueda y entrenamiento cuente con múltiples GPUs, se puede indicar cuales utilizar.
- **Learning Rate:** Corresponde a la tasa de aprendizaje para el entrenamiento de la red neuronal NAS.
- **Arch Learning Rate:** La tasa de aprendizaje para el entrenamiento de los pesos que ponderan cada bloque, lo que define a la arquitectura NAS.
- **Batch Size:** Cantidad de imágenes por batch para la búsqueda y entrenamiento, esto varía dependiendo del tamaño de la red y la capacidad de la(s) GPU(s).
- **Uso de MaxPool:** Se decide si se utilizará la operación MaxPool antes de la primera etapa de la red. Esto también afecta el valor del stride en la primera etapa (cambiando de stride 2 a 1).
- **Strides a utilizar en cada etapa:** Se define el stride a utilizar en cada etapa. Esto es más relevante cuando se utilizan más de 4 etapas en la red, ya que deben haber por lo menos 4 strides con valor 2 en esta configuración (o 3 en caso de utilizar MaxPool).
- **Épocas totales de warm-up:** Antes de la búsqueda e iteración de los valores ponderados, se realizan épocas de warm-up para que los pesos de los bloques

converjan un poco y tengan una mayor precisión antes de medir su importancia en la red.

- **Épocas totales de búsqueda**: Indica la cantidad de épocas en las que se llevará a cabo el entrenamiento y la búsqueda de la red neuronal NAS final.
- **Valor de lambda y alpha para las funciones de pérdida:** Estos valores determinan la importancia relativa de cada componente de la función de pérdida (descrito en la sección 3.4). Ayudan a definir en qué se enfocará más el método NAS durante la búsqueda.

Es importante recalcar la posibilidad de retomar el entrenamiento, desde una época específica de búsqueda o desde el inicio posterior a un warm-up de los pesos. Esto permite realizar experimentos adicionales sobre la misma base o probar variaciones en algunos de los hiperparámetros para comprender mejor el comportamiento del método NAS.

3.8 Hardware Objetivos

Una vez definido el método NAS y su funcionamiento, es importante destacar y definir los hardware objetivos para esta tesis. Los siguientes dispositivos han sido seleccionados con el propósito de abarcar un rango variado de hardware a utilizar. El primero de ellos posee una gran cantidad de recursos computacionales, el segundo cuenta con una GPU de gama media-baja en un dispositivo embebido, y el último caso involucra un hardware embebido sin GPU. A partir de los resultados obtenidos en estos dispositivos, se pretende demostrar la eficacia del enfoque NAS propuesto para esta tesis. Los hardware objetivos son los descritos a continuación.

3.8.1 DGX-1

La DGX-1 es una supercomputadora de IA diseñada por Nvidia. Su alta capacidad de cómputo tanto en CPU como en GPU le permite poder realizar entrenamientos a gran escala y acelerar los entrenamiento y experimentos que se buscan llevar a cabo. Por lo tanto, será el hardware encargado de realizar el entrenamiento y búsqueda NAS para los 3 hardware objetivos elegidos, siendo la DGX-1, uno de ellos. Visualmente, en la Figura 38 se muestra la DGX-1.

Figura 38: Supercomputadora DGX-1 (https://www.amax.com/wp-content/uploads/2020/01/DGX-1-IMG.png).

3.8.1.1 Especificaciones del Hardware

Para las especificaciones del hardware se mencionarán las más relevantes para el óptimo desempeño del método. En base a esto, la DGX-1 tiene como componentes internos más relevantes:

- **CPU:** Cuenta con una Intel Xeon E5-2698 v4 con una frecuencia base de 2.2GHz y 20 núcleos físicos por CPU. Esta CPU también cuenta con la tecnología Hyper-Threading de Intel, permitiendo que cada núcleo físico pueda manejar dos hilos de ejecución de manera simultánea mejorando aún más la capacidad de procesamiento de la DGX-1.
- **RAM:** 512 GB DDR4 de capacidad, permitiendo cargar grandes cantidades de volúmenes de datos para realizar una búsqueda y entrenamiento eficiente sin importar del tamaño o complejidad que tenga la red neuronal.
- **GPU:** Uno de los aspectos más potentes de esta Supercomputadora es su GPU, ya que cuenta con 8 GPUs Tesla V100 de 32GB de capacidad cada una. Esta GPU se basa en la arquitectura Volta y cuenta con 5120 núcleos CUDA, por lo que aparte de su gran capacidad de memoria también cuenta con la tecnología necesaria para realizar entrenamientos de la manera más eficiente posible.

Una tabla resumen de la información descrita anteriormente es la mostrada en la Tabla 20.

Tabla 20: Especificaciones de hardware DGX-1.

CPU	RAM	GPU
Dual 20-Core Intel Xeon E5-2698 v4 2.2GHz.	512 GB 2133MHz DDR4.	8xNvidia Tesla V100 32GB.

3.8.1.2 Parámetros de Búsqueda

Al ser una supercomputadora de IA, en este caso no se limita a ciertos bloques en específico, por lo que se hará prueba con cada uno de ellos. El espacio de búsqueda a utilizar se encuentra en la Tabla 21.

Tabla 21: Parámetros de búsqueda para DGX-1.

Bloque Básico	Variaciones de Kernel	Variaciones a Utilizar
MBV1-Block	3x3, 5x5	-
MBV2-Block	3x3, 5x5	Expansión r = 3, 4, 6
BL-Block	3x3, 5x5	Expansión 1/r = 1/4
KK-Block	3x3	Variación Mid
DK-Block	3x3	-

3.8.2 Raspberry Pi v4

La Raspberry Pi v4 representa la cuarta generación de dispositivos embebidos de bajo costo desarrollados por la Raspberry Pi Foundation. Visualmente se encuentra en la Figura 39.

Figura 39: Raspberry Pi v4 (https://res.cloudinary.com/rsc/image/upload/b_rgb:FFFFFF,c_pad,dpr_1.0,f_auto,q_auto,w_700/c_pad,w_700/R1822096-01).

Entre sus características más destacadas se encuentra su alto rendimiento en términos de CPU, especialmente para ser un dispositivo embebido. Esto le otorga versatilidad para su aplicación en diversos campos. La Raspberry Pi v4 ha logrado una amplia adopción en múltiples industrias debido a su sólido desempeño, su activa comunidad de desarrolladores y su eficiencia energética. Por estas razones, se ha seleccionado como uno de los hardware objetivos para esta tesis.

3.8.2.1 Especificaciones del Hardware

La Raspberry Pi v4 cuenta con los siguientes componentes principales:

- **CPU:** Esta nueva generación de Raspberry Pi mejora con respecto a su predecesora con relación a la CPU. Cuenta con un procesador Quad core Cortex-A72 del tipo ARM v8, con cuatro núcleos físicos a una frecuencia de 1.8GHz.
- **RAM:** Existen múltiples variaciones en capacidad de RAM. Se encuentran desde 1GB hasta 8GB de RAM de capacidad (las variaciones posibles son 1GB, 2GB, 4GB y 8GB). Para efectos de esta tesis, se utilizará la Raspberry Pi que cuenta con 4GB de RAM.
- **GPU:** Este dispositivo no cuenta con GPU.

El resumen de esta información se encuentra en la Tabla 22.

Tabla 22: Especificaciones de hardware Raspberry Pi v4.

CPU	RAM	GPU
Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz.	4GB LPDDR4-3200 SDRAM.	No contiene.

3.8.2.2 Parámetros de Búsqueda

Para este caso debido a las limitantes de no contar con GPU, se quitan los kernel 5x5 para MBV2-Block y BL-Block. Para el caso de MBV1-Block se mantiene el uso de kernel 5x5 debido a su bajo coste computacional, pero como se verá en los experimentos y resultados, también se quitará posteriormente. En la Tabla 23 se puede apreciar el resumen de los parámetros de búsqueda a utilizar.

Tabla 23: Parámetros de búsqueda para Raspberry Pi v4.

Bloque Básico	Variaciones de Kernel	Variaciones a Utilizar
MBV1-Block	3x3, 5x5	-
MBV2-Block	3x3	Expansión r = 3, 4, 6
BL-Block	3x3	Expansión 1/r = 1/4
KK-Block	3x3	Variación Mid
DK-Block	3x3	-

3.8.3 Jetson Nano

La Jetson Nano es una plataforma de desarrollo embebida similar a una Raspberry Pi, pero con la diferencia de su enfoque en el procesamiento de algoritmos de inteligencia artificial. Este dispositivo está equipado con una GPU que permite el procesamiento en tiempo real de una amplia gama de algoritmos de IA. Diseñada por Nvidia y lanzada al público en 2019, la Jetson Nano marcó un hito importante en la oferta de hardware embebido de Nvidia. A diferencia de sus predecesoras, como la Jetson TX1 y TX2, la Jetson Nano se destacó por su accesibilidad y un menor consumo de energía sin sacrificar significativamente su rendimiento. Este dispositivo embebido se muestra visualmente en la Figura 40.

Figura 40: Jetson Nano (https://m.media-amazon.com/images/I/51jt-o2nDAL._AC_UF1000,1000_QL80_.jpg).

3.8.3.1 Especificaciones del Hardware

A diferencia de la Raspberry Pi, como se puede apreciar en las especificaciones de la Tabla 24, la Jetson Nano tiene una CPU de menor rendimiento que una Raspberry Pi v4. Esto se debe a que, en su diseño, la Jetson Nano está orientada a procesar cargas de trabajo más intensivas en su GPU, que es el componente comúnmente utilizado para procesar algoritmos de inteligencia artificial. Por lo tanto, la elección entre estos dos dispositivos para diversas aplicaciones dependerá del tipo de proyecto a desarrollar, ya que cada uno tiene sus propias ventajas y desventajas correspondientes.

Tabla 24: Especificaciones de hardware Jetson Nano.

CPU	RAM	GPU
Quad-core ARM Cortex-A57 MPCore processor.	RAM 4GB DDR4.	GPU Nvidia Maxwell, 128 CUDA cores.

3.8.3.2 Parámetros de Búsqueda

La Jetson Nano, en contraste con la Raspberry Pi, se beneficia de su GPU para procesar bloques convolucionales y redes de manera paralela. Sin embargo, dado su enfoque como dispositivo embebido, que busca mantener costos bajos y un consumo eficiente de energía, su GPU presenta limitaciones. En comparación con la DGX, que cuenta con GPU V100 y sus 5120 núcleos CUDA, la Jetson Nano ofrece un rendimiento significativamente menor con sus 128 núcleos CUDA. Esta diferencia en rendimiento y limitaciones es especialmente notable cuando se trata de bloques que requieren un alto grado de paralelización para lograr un desempeño óptimo. Por lo tanto, se plantea la hipótesis de que el rendimiento de estos bloques será diferente en este hardware.

Con base en estas consideraciones, el espacio de búsqueda definido se presenta en la Tabla 25.

Tabla 25: Parámetros de búsqueda para Jetson Nano.

Bloque Básico	Variaciones de Kernel	Variaciones a Utilizar
MBV1-Block	3x3, 5x5	-
MBV2-Block	3x3, 5x5	Expansión r = 3, 4, 6
BL-Block	3x3	Expansión 1/r = 1/4
KK-Block	3x3	Variación Mid
DK-Block	3x3	-

Si bien es posible que se eliminen algunos de estos bloques o variaciones por los resultados de latencia obtenidos en la Jetson Nano, este espacio de búsqueda presentado en la Tabla 25 será la base inicial con la que se trabajará e iterará en base a los resultados obtenidos posteriormente.

3.9 Uso de FP16

Con el fin de optimizar el aprovechamiento de la capacidad y la velocidad de la GPU durante el entrenamiento, lo que permite realizar iteraciones más rápidas y manejar conjuntos de datos más grandes, se implementa una tecnología que en su momento surgió como resultado de la investigación llevada a cabo por Nvidia, conocida como Apex. La ventaja clave de esta tecnología radica en la conversión de ciertos parámetros o pesos de la red neuronal de precisión de punto flotante de 32 bits (FP32) a precisión de punto flotante de 16 bits (FP16).

Esta modificación conlleva a almacenar un menor número de decimales, lo que, a su vez, acelera las operaciones matemáticas, como multiplicaciones y otras operaciones entre estos valores. En promedio, esta mejora puede aumentar la velocidad de entrenamiento en un rango de 2 a 8 veces, al tiempo que reduce el uso de la GPU en hasta un 50% para un modelo de red neuronal dado.

Aunque esta mejora se implementa utilizando la biblioteca y el repositorio de Nvidia, debido a su significativo impacto, se ha integrado directamente en la biblioteca de Pytorch. Esto se logra reimplementando la biblioteca Apex de Nvidia y adaptándola a su contraparte de Pytorch, conocida como Automatic Mixed Precision (AMP). Este enfoque simplifica la implementación en el código y permite aprovechar los beneficios de la precisión de FP16 de manera más directa.

Capítulo 4

Experimentos, Resultados y Análisis

Para cada uno de estos tres hardware objetivos definidos, se diseñaron diferentes enfoques, modificando, de ser necesario, los parámetros y bloques a utilizar a medida que se obtenían resultados con el fin de obtener una red neuronal definitiva que cumpliera con los requisitos establecidos.

Es importante destacar que, aunque se contaba con un diccionario de atributos de latencia para los bloques obtenidos en su hardware correspondiente, los entrenamientos y búsquedas se llevaron a cabo en la supercomputadora DGX-1 debido a su capacidad y potencia de cómputo. Los resultados obtenidos para cada caso se detallan a continuación.

4.1 Hardware Supercomputadora DGX-1

Como este fue el primer hardware objetivo en utilizar, se llevaron a cabo una mayor cantidad de experimentos base con el propósito de comparar diversas combinaciones y comprender de manera más profunda el funcionamiento de cada bloque, variación y posible comportamiento.

Por otro lado, es importante destacar que la red de referencia para este caso es la RetinaFace base, la cual utiliza la ResNet50 [He et al., 2016] como extractor de características. Para garantizar una comparación justa con respecto al método NAS, se entrenó la RetinaFace [Deng et al., 2019] con el extractor de características ResNet50 [He et al., 2016] utilizando los valores por defecto. Los resultados obtenidos para esta red en las tres categorías de precisión de WIDER FACE [Yang et al., 2016] se muestran en la Tabla 26.

Tabla 26: Precisión y latencia de RetinaFace Base.

Nombre Modelo	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características[[8,640,640]
RetinaFace Base (ResNet50)	95,27	93,83	84,33	91,14	5,905ms

En esta tabla, también se incluye la latencia de la red neuronal ResNet50 [He et al., 2016] en la DGX-1, la cual servirá como referencia para comprender el costo computacional de esta red y lo que se busca mejorar con lo propuesto en esta tesis. La

latencia del extractor de características se obtuvo utilizando un batch size de 8 y una resolución de entrada de 640x640.

Como primer paso para comprender el funcionamiento de los bloques básicos en el espacio de búsqueda y su eficiencia en este hardware objetivo, se generó un diccionario con sus valores, y se crearon gráficos que se dividen según su dimensionalidad. La dimensionalidad indica en qué etapa de la red se encuentran realizando inferencias y se relaciona con los tensores de entrada. La reducción de la dimensionalidad se produce cuando hay un cambio de etapa y se procesa el tensor con un tipo de bloque con stride 2 o en el caso de MaxPool con stride 2. Las diferentes dimensionalidades se describen de la siguiente manera:

- **Dimensionalidad 320:** Se refiere a las inferencias en las etapas previas (de las cuatro etapas) de la arquitectura base del extractor de características NAS.
- **Dimensionalidad 160:** Corresponde a las inferencias en la primera etapa del extractor de características NAS.
- **Dimensionalidad 80:** Corresponde a la inferencia de la segunda etapa del extractor de características NAS.
- **Dimensionalidad 40:** Corresponde a la inferencia de la tercera etapa del extractor de características NAS.
- **Dimensionalidad 20:** Corresponde a la inferencia de la cuarta etapa del extractor de características NAS.

Después de comprender este concepto, se presentarán gráficos agrupados que permiten realizar un análisis general del comportamiento de estos bloques. La totalidad de los gráficos se encuentran en el apéndice para un análisis más detallado.

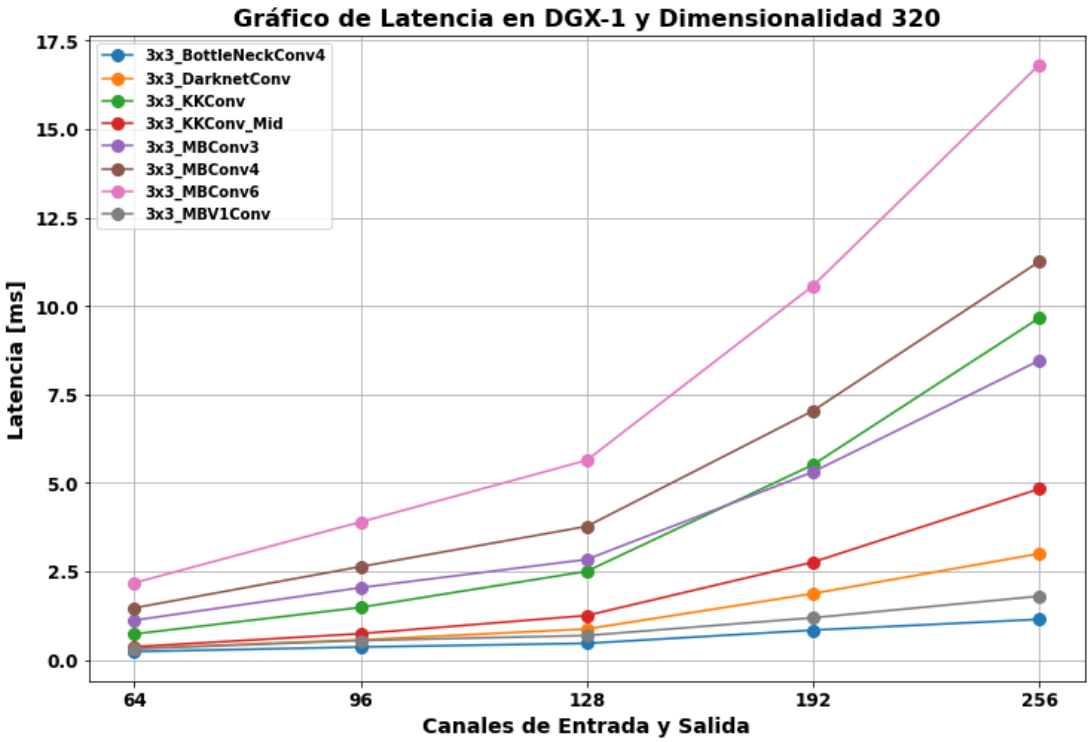

Figura 41: Gráfico de Latencia en DGX-1, Dimensionalidad 320.

En el primer gráfico, presentado en la Figura 41, se observa el funcionamiento de los tipos de bloques para las etapas previas de la arquitectura extractor de características NAS. Teniendo en cuenta que la referencia a utilizar es 5,5 ms, a partir de los 96 canales ya no son factibles una gran variedad de estos bloques básicos analizados para su inclusión en el espacio de búsqueda.

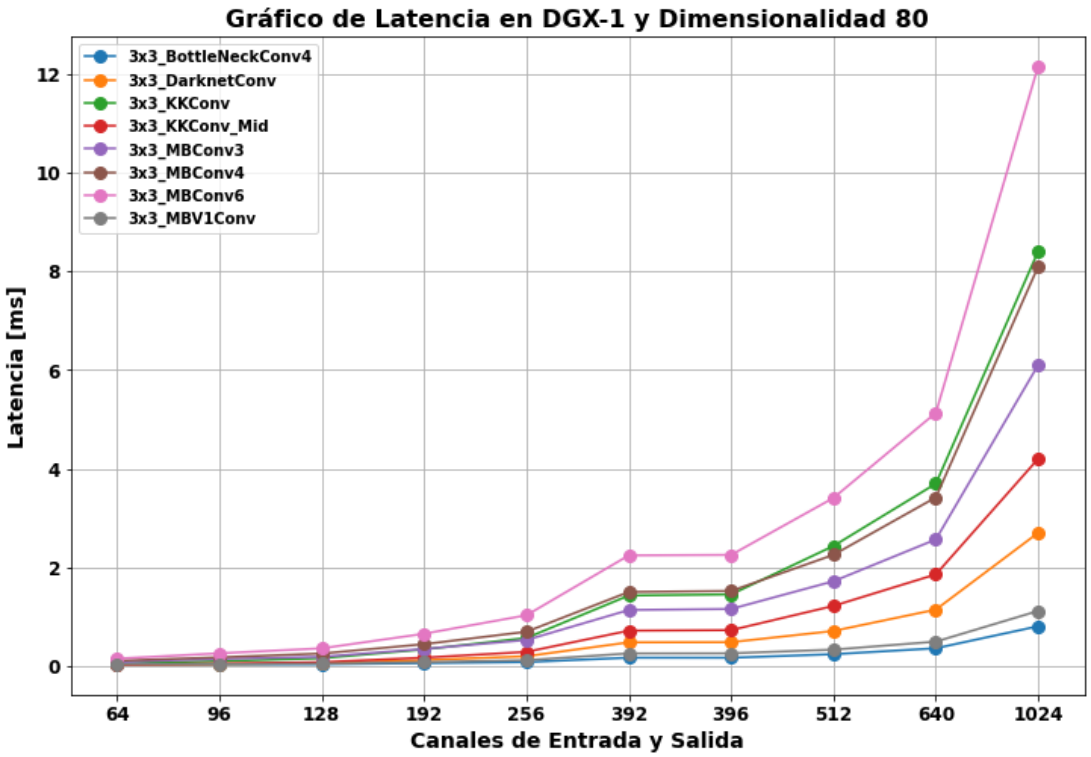

Figura 42: Gráfico de Latencia en DGX-1, Dimensionalidad 80.

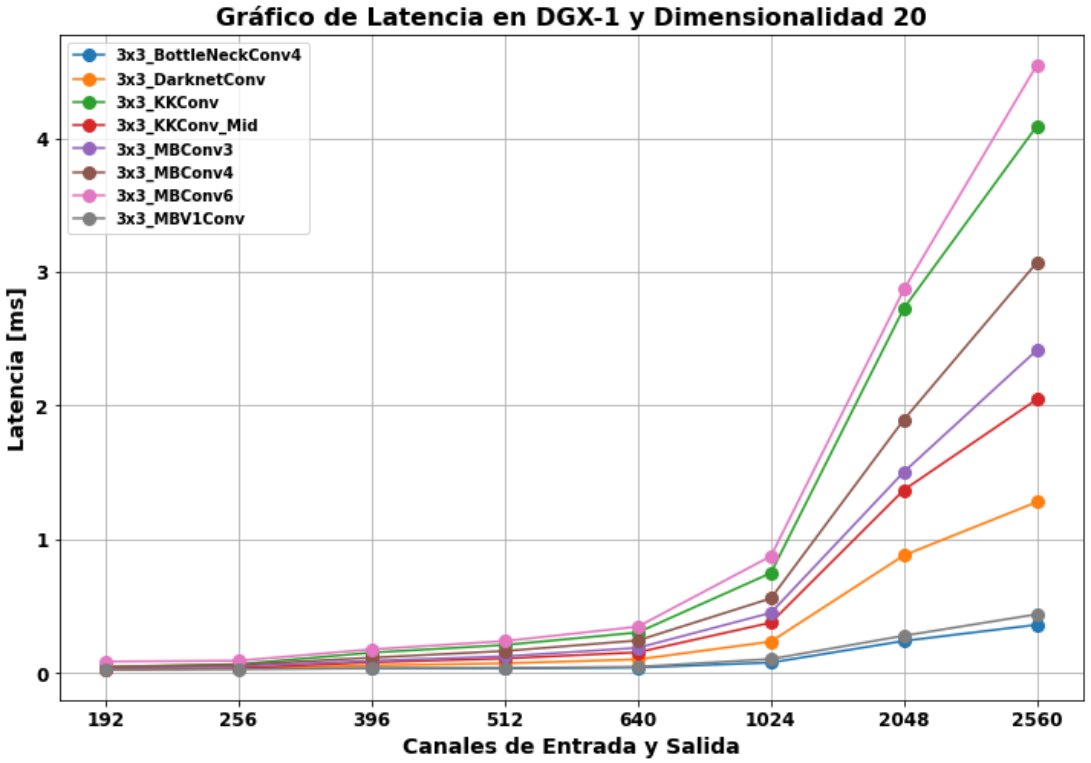

Figura 43: Gráfico de Latencia en DGX-1, Dimensionalidad 20.

En las Figuras 42 y 43, se presenta a gran escala la forma exponencial en la que las latencias de los tipos de bloques aumentan con respecto a la cantidad de canales. Esto es especialmente notorio en el gráfico de la dimensionalidad 20, donde el factor exponencial de la curva comienza a crecer de manera significativa a partir de 640 canales. Esta tendencia se aprecia en la mayoría de los bloques, excepto en los tipos BottleNeck (BL) y MBV1, donde el aumento de canales no representa un impacto tan significativo en la latencia o no se da de manera tan exponencial como en otros tipos de bloques.

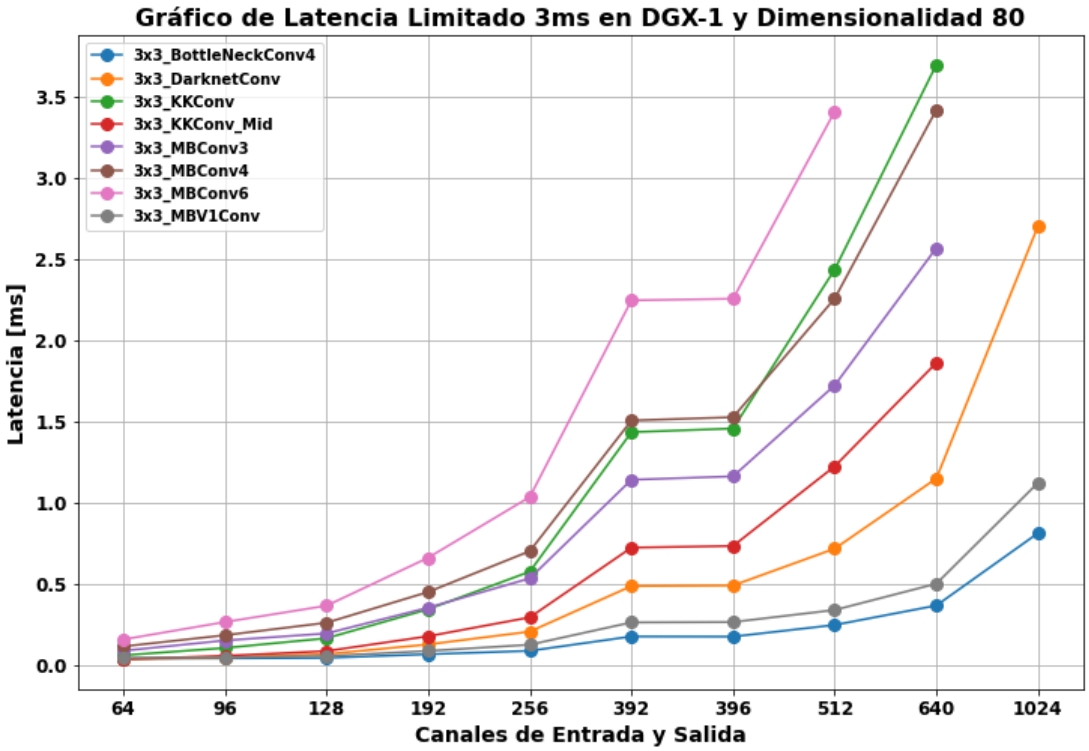

Figura 44: Gráfico de Latencia en DGX-1, Dimensionalidad 80, Acotado a 3ms.

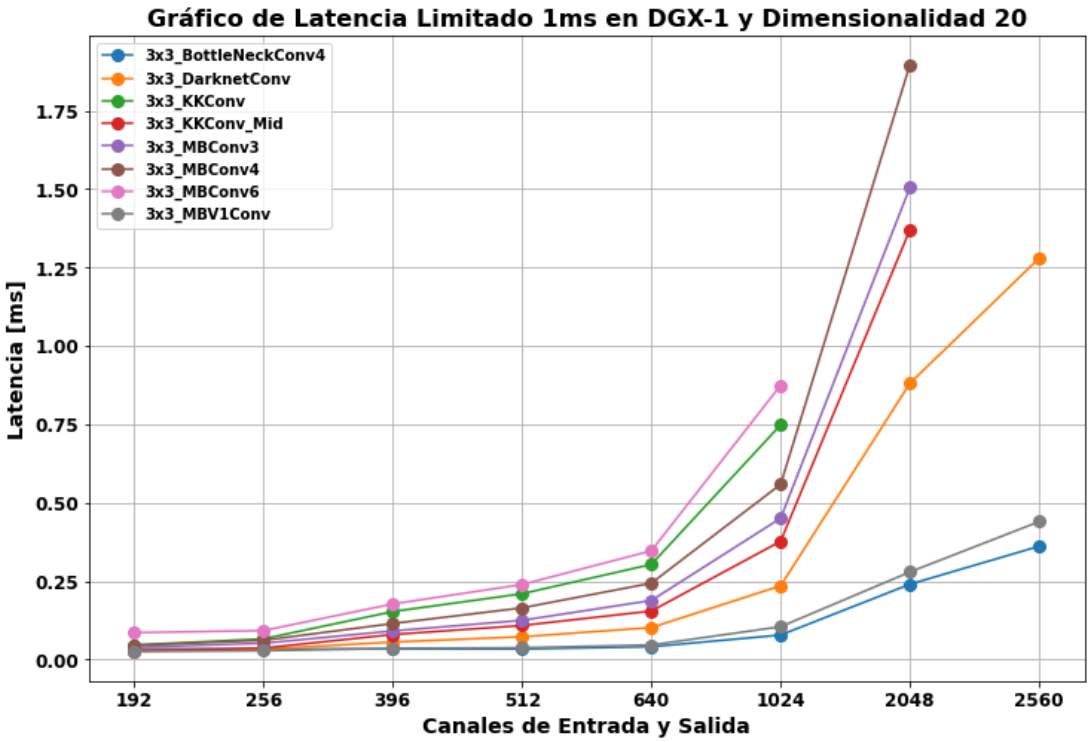

Figura 45: Gráfico de Latencia en DGX-1, Dimensionalidad 20, Acotado a 1ms.

Para un análisis más específico en las últimas etapas de la red (baja dimensionalidad), se acotaron los gráficos a 3 ms y 1 ms, como se muestra en las Figuras 44 y 45. Se observa que, en general, los bloques MBV2 que contienen un factor de expansión en su bloque intermedio presentan las mayores latencias en este hardware, y estas latencias aumentan de manera exponencial a medida que se incrementan los canales. Esto indica que, si bien este tipo de bloque puede aportar mejoras en precisión, su alto costo computacional lo convierte en una opción no factible para este hardware objetivo.

Los bloques que presentan un mejor desempeño en términos de latencia para este hardware son los BottleNeck, los MBV1, la DarkNet (DK), y las KK/KK_mid. Aunque no se restringirá el uso de otros tipos de bloques básicos, se tendrán en consideración estos resultados tanto en el diseño de los experimentos como en las futuras iteraciones.

Posterior a este análisis, se realiza una comparación entre los bloques de la familia MobileNets [Howard et al., 2017, Howard et al., 2019, Sandler et al., 2018] en sus variaciones de kernel. Los gráficos obtenidos son:

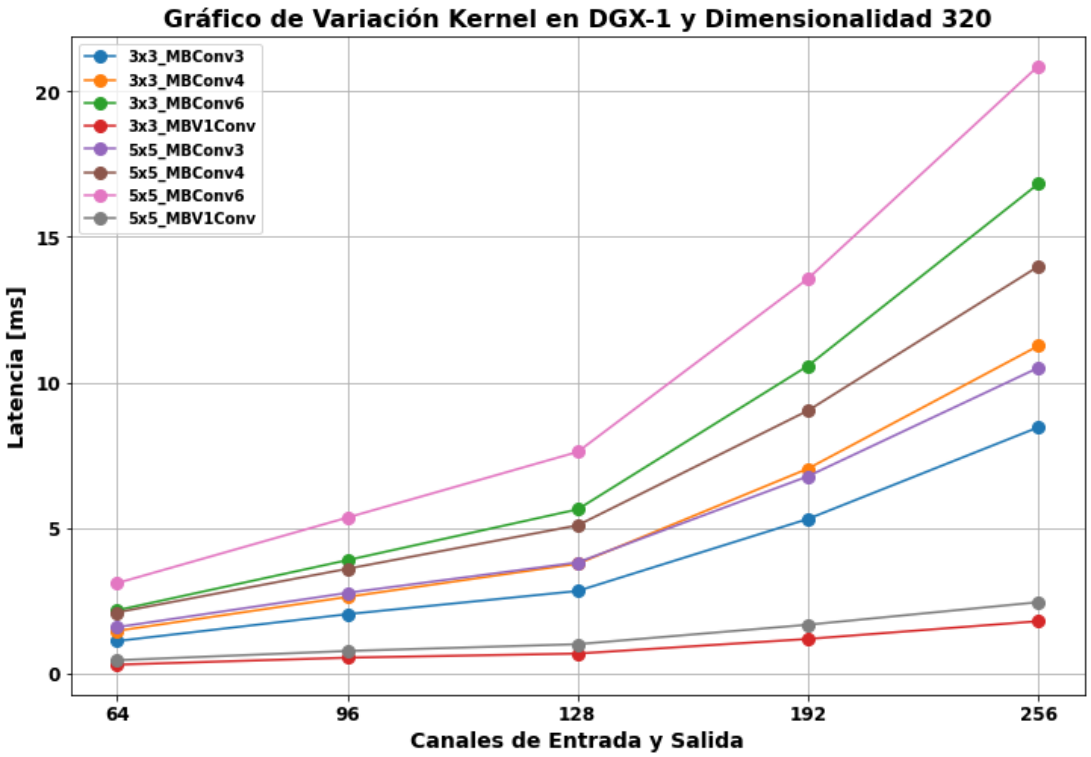

Figura 46: Gráfico de Latencia en DGX-1, Dimensionalidad 320, Variación de kernel 3x3 y 5x5.

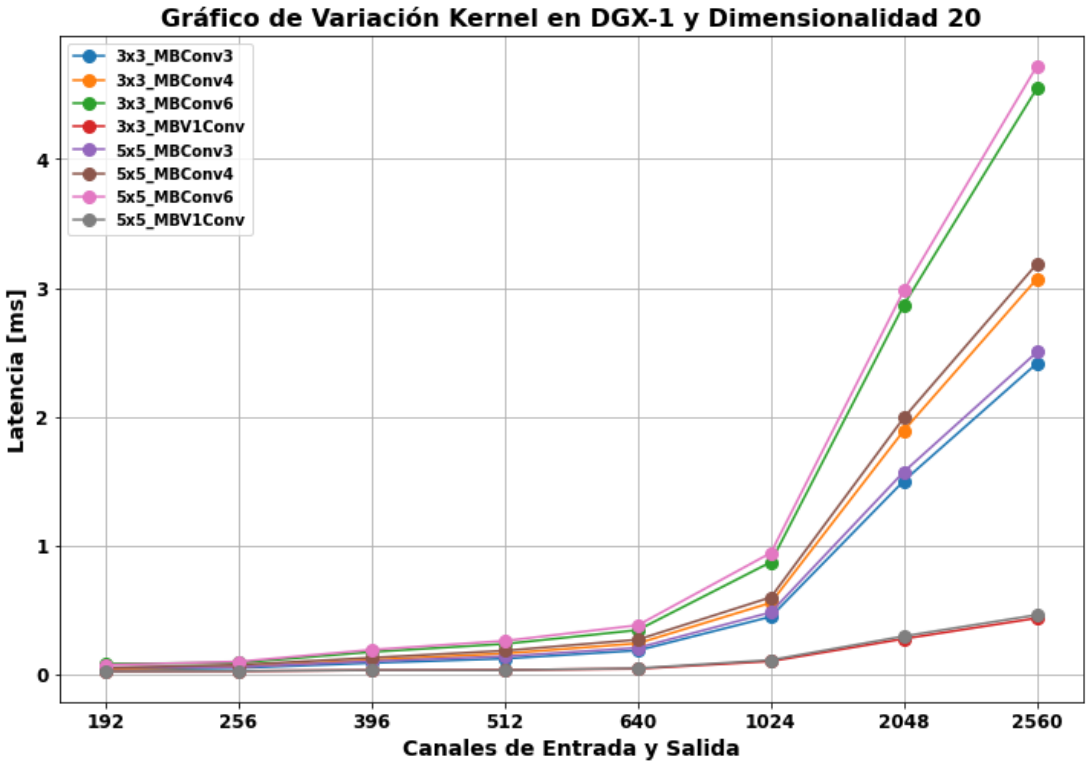

Figura 47: Gráfico de Latencia en DGX-1, Dimensionalidad 20, Variación de kernel 3x3 y 5x5.

Los resultados se muestran en las Figuras 46 y 47. Se evidencia que las diferencias de latencia entre las versiones de kernel 3x3 y 5x5 de sus respectivos bloques básicos son lo suficientemente significativas como para no considerar estas variaciones de kernel de 5x5, ya que resultarían contraproducentes en el modelo debido a su costo computacional en relación con la precisión que pueden aportar.

Con estos primeros acercamientos al funcionamiento de los bloques en este hardware objetivo, se procede al diseño de los primeros experimentos para obtener una idea más clara del desempeño de estos.

Se toma como punto de partida el valor de latencia de referencia y la precisión obtenida por el modelo RetinaFace Base. A partir de estos valores, se realizan variaciones en los tipos de bloques básicos, manteniendo una cantidad de canales similar al diseño de la ResNet50 [He et al., 2016], excepto en los casos en que se utilizan bloques MBV2, que requieren una cantidad menor de canales debido a su valor de expansión interno.

Los parámetros iniciales fijados para las pruebas son los siguientes:

- **Latencia esperada de la red neuronal NAS:** 5,5 ms.
- **Uso de SE/eSE:** Se aplica eSE [Lee & Park, 2020] en todos los experimentos en las dos primeras etapas.
- **Resolución de entrenamiento y búsqueda:** 640x640 píxeles.
- **Valor de lambda:** 7,5 x 10^-1
- **Alpha:** Se fija en 2.0 para mantener la consistencia con el entrenamiento diseñado para RetinaFace [Deng et al., 2019] en Pytorch, permitiendo así comparaciones precisas.
- **Batch de Entrenamiento y Búsqueda:** 24.
- **Bloque Básico NAS Inicial (primer MixedEdge en la etapa previa):** KK_Mid.

Los experimentos realizados se detallan en la Tabla 27.

Tabla 27: Diseño de experimentos para primera iteración de Método FaceNAS en DGX-1.

	Tipo de Bloque y Cantidad de Bloques				Canales			
Nombre Test	Stage1	Stage2	Stage3	Stage4	S1	S2	S3	S4
T1	KK+DK|3	KK+DK|8	BL|8	MBV2|8	128	256	512	640
T2	KK+DK+BL|3	KK+DK+BL|8	BL|8	MBV2|8	128	256	512	640
T3	KK+DK|3	KK+DK+BL|8	KK+DK+BL|8	KK+DK+BL|8	128	256	512	1024
T4	KK+DK+BL|3	KK+DK+BL|8	KK+DK+BL|8	KK+DK+BL|8	128	256	512	1024
T5	BL|2	BL|8	BL|8	BL|8	128	256	512	1024
T6	KK+DK|3	KK+DK|8	KK+DK|8	KK+DK|8	128	256	512	1024

T7	MBV2|5	MBV2|8	MBV2|8	MBV2|8	32	64	128	256
T8	KK+DK+BL|3	KK+DK+BL|8	MBV2|8	MBV2|8	128	256	192	320
T8_v2	KK+DK|2	BL|4	BL|12	MBV2|12	128	256	640	640
T9	KK+DK+BL|3	KK+DK+BL|4	MBV2|12	MBV2|12	64	128	256	384
T10	MBV1|3	MBV1|5	MBV1|12	MBV1|8	256	512	1024	2048
T11	KK+DK|2	KK+DK|4	MBV1|12	MBV1|8	64	128	640	1280
T12	MBV1|3	MBV1|5	BL|12	BL|8	256	512	1024	2048

Esta tabla se muestra las configuraciones iniciales para el diseño de estos experimentos. En la primera sección de la tabla, se enumeran los tipos de bloques con su etapa correspondiente, donde puede haber más de un conjunto o tipo de bloque básico. El método tiene la capacidad de combinar estos conjuntos de miniespacios de búsqueda para iterar sobre ambos y determinar cuál tiene mayor relevancia para la etapa de diseño específica.

Después de indicar el tipo o tipos de bloques, se muestra la cantidad de capas o bloques secuenciales que tendrá esta etapa en particular (separado por un 'I'). Basado en diseños de diferentes redes como TResNet [Ridnik et al., 2020], ResNet [He et al., 2016] y GENet [Lin et al., 2020], se mantiene la práctica de utilizar pocas capas en las primeras etapas y se asigna un mayor número de capas a las etapas 3 y 4. El método NAS tomará decisiones sobre qué bloques prefiere utilizar y eliminará los otros mediante el uso del bloque Zero (identidad o vacío).

En la segunda sección se definen los canales de entrada y salida de la etapa correspondiente. Estos canales buscan mantener un diseño o cantidad de canales similares a los utilizados en redes neuronales existentes y probando distintas variaciones.

Teniendo las redes encontradas para cada uno de los experimentos, se realiza un entrenamiento de estas redes neuronales encontradas y se obtienen las siguientes precisiones.

Tabla 28: Resultados de experimentos para primera iteración de Método FaceNAS en DGX-1.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[8,640,640]
T1	93,02	91,33	83,19	89,18	5,2ms
T2	92,60	91,09	83,70	89,13	4,94ms
T3	93,37	92,19	85,22	**90,26**	5,117ms
T4	93,00	91,60	84,10	89,57	6,12ms
T5	92,5	91	83,2	88,90	5,56ms
T6	91,1	89,6	82,29	87,66	**3,367ms**
T7	93,40	91,90	83,60	89,63	5,169ms

T8	93,48	92	83,87	89,78	5,483ms
T8_v2	92,83	91,54	83,40	89,26	5,79ms
T9	92,34	90,74	81,90	88,33	6,24ms
T10	93,2	90,98	78,84	87,67	6,517ms
T11	93,3	91,08	79,45	87,94	4,211ms
T12	93,77	91,44	79,18	88,13	6,048ms

Frente a estos resultados obtenidos en la Tabla 28, se puede apreciar que las precisiones se encuentran en un rango similar, pero con una ligera predominancia del experimento T3 en la precisión promedio. Sin embargo, el experimento T6 destaca por tener la latencia más baja, aunque su precisión también es más baja en comparación con los otros experimentos.

En estos experimentos, se observa una mayor utilización de bloques KK y DK en las primeras etapas en comparación con los bloques BottleNeck (BL). En contraste, para las últimas etapas, los bloques BL tienen un papel más importante en el diseño. Además, se observa que el experimento T10, que utiliza bloques básicos MBV1, no es eficiente en términos de precisión, ya que muestra una precisión cercana a la más baja, acompañada del tiempo de latencia más alto. Esto se refleja de manera similar en el experimento T9, que utiliza bloques MBV2, aunque en menor medida que el experimento T10.

Basándose en estos resultados, se decide trabajar con la configuración realizada en el experimento T3 para realizar una segunda iteración de experimentos, la cual se denominará TT3.

El experimento TT3 mantiene los espacios de búsqueda de KK y DK para las dos primeras etapas, mientras que utiliza bloques BL para las dos últimas etapas. Como modificación en la cantidad de canales, se reduce el primer canal a 64 canales, el segundo canal a 128 canales, y se aumenta a 1024 y 2048 canales respectivamente en las últimas dos etapas de bloques BL. La tabla resumen de estas modificaciones se presenta en la Tabla 29.

Tabla 29: Diseño de nuevo experimento para segunda iteración de Método FaceNAS.

	Tipo de Bloque y Cantidad de Bloques				Canales			
Nombre Test	Stage1	Stage2	Stage3	Stage4	S1	S2	S3	S4
TT3	KK+DK|2	KK+DK|3	BL|12	BL|5	64	128	1024	2048

Basándose en el diseño de la ResNet [He et al., 2016], se realiza otra modificación en la arquitectura y se implementa la variación de usar MaxPool previo a las cuatro etapas, como se menciona en la sección 3.3.3.1. Aplicando esta variación se procesa el experimento en el método NAS obteniendo la siguiente red mostrada en la Tabla 30.

Tabla 30: Arquitectura interna del experimento TT3.

Arquitectura Final, Experimento TT3	
Bloques Básicos	Bloques Residuales
SE_3x3_KKConv_32	None
3x3_MAXPool	None
SE_3x3_KKConv_64	1x1_Conv_64
3x3_KKConv_Mid_64	Identity
S_3x3_DKConv_128	1x1_Conv_Pool_128
SE_3x3_KKConv_128	Identity
SE_3x3_KKConv_128	Identity
S_SE_3x3_BottleNeckConv4_1024	1x1_Conv_Pool_1024
SE_3x3_BottleNeckConv4_1024	Identity
SE_3x3_BottleNeckConv4_1024	Identity
SE_3x3_BottleNeckConv4_1024	Identity
S_3x3_BottleNeckConv4_2048	1x1_Conv_Pool_2048
3x3_BottleNeckConv4_2048	Identity
3x3_BottleNeckConv4_2048	Identity
3x3_BottleNeckConv4_2048	Identity
3x3_BottleNeckConv4_2048	Identity
1x1_Conv_2560	-

Obteniendo este nuevo extractor de características NAS llamado TT3. Posteriormente se entrena en la base de datos WIDER FACE [Yang et al., 2016] obteniendo como resultado lo mostrado en la Tabla 31.

Tabla 31: Resultados de experimento TT3.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características[[8,640,640]
TT3	94,04	92,21	82,64	89,63	4,45ms

Teniendo como resultado una precisión cercana a la obtenida en el experimento T3, pero con menor latencia.

Por otro lado, como se define en los primeros experimentos, todas estas arquitecturas consistían en utilizar eSE [Lee & Park, 2020] en las primeras dos etapas del extractor de características. Sobre esta base se realizarán variaciones en TT3 para ver su comportamiento tanto en el uso de SE [Hu et al., 2018], como en el no uso de SE [Hu et al., 2018] o eSE [Lee & Park, 2020]. Los resultados obtenidos son los mostrados en la Tabla 32 presentada a continuación.

Tabla 32: Resultados de variaciones de experimento TT3 con eSE, NoSE y SE.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características[8,640,640]
TT3_eSE(base)	94,04	92,21	82,64	89,63	4,45ms
TT3_NoSE	94,52	92,8	83,18	90,17	**4,28ms**
TT3_SE	94,57	92,79	83,31	**90,22**	4,46ms

La tabla anterior muestra que el experimento TT3_eSE muestra un comportamiento contraproducente al hacer que la arquitectura disminuya su precisión y aumente su latencia. La variación de usar SE [Hu et al., 2018] aumenta ligeramente su precisión, pero aumentando su latencia en un 10%, por lo que se define como extractor de características final el TT3_NoSE que no utiliza las variaciones ni de SE [Hu et al., 2018] ni de eSE [Lee & Park, 2020].

Tabla 33: Resultados finales y comparación con RetinaFace base para DGX-1.

Nombre Modelo	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características[8,640,640]
RetinaFace Base (ResNet50)	95,27	93,83	84,33	**91,14**	5,905ms
TT3_NoSE	94,52	92,8	83,18	90,17	**4,28ms**

En la Tabla 33 se puede ver que el extractor de características encontrado presenta una disminución del 1% de precisión promedio con respecto a la red RetinaFace [Deng et al., 2019], pero presentando una mejora de aproximadamente un 38% en tiempo de latencia. Esto presenta una mejora importante para su funcionamiento en tiempo real y escalabilidad frente a la RetinaFace [Deng et al., 2019] original.

De esta manera el método logra obtener una red que logra maximizar la precisión mientras que disminuye la latencia del modelo, encontrando en este caso una red con precisión similar, pero con una mejora sustancial en su tiempo de latencia. La arquitectura gráfica de la red final es la mostrada en la Tabla 34.

Tabla 34: Arquitectura de TT3_NoSE.

Modelo TT3_NoSE	Bloques Básicos	Bloques Residuales
First Conv	3x3_Conv_8	-
Pre-Stage	3x3_KKConv_32 3x3_MAXPool	- -
Stage 1	3x3_KKConv_64 3x3_KKConv_Mid_64	1x1_Conv_64 Identity
Stage 2	S_3x3_DKConv_128 [3x3_KKConv_128] x 2	1x1_Conv_Pool_128 [Identity] x 2
Stage 3	S_3x3_BottleNeckConv4_1024 [3x3_BottleNeckConv4_1024] x 3	1x1_Conv_Pool_1024 [Identity] x 3
Stage 4	S_3x3_BottleNeckConv4_2048 [3x3_BottleNeckConv4_2048] x 4	1x1_Conv_Pool_2048 [Identity] x 4
Last Conv	1x1_Conv_2560	-

4.2 Hardware CPU embebida, Raspberry Pi v4

Para el hardware objetivo de Raspberry Pi v4, debido a sus limitaciones en términos de latencia y capacidad de procesamiento, se obtuvieron los valores de latencia y el diccionario de atributos de latencia de los bloques básicos en una configuración de batch 1 y una resolución de 320x320. Como punto de referencia para comparar con una red ya existente, utilizamos la versión móvil de RetinaFace [Deng et al., 2019]. Los valores de precisión y latencia de RetinaFace Mobile (Mobile0.25) en Raspberry Pi v4 se muestran en la Tabla 35.

Tabla 35: Precisión y latencia de RetinaFace mobile en Raspberry Pi v4.

Nombre Modelo	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
RetinaFace Mobile (Mobile0.25)	89,71	86,83	71,47	82,67	204ms

Con estos resultados como punto de referencia, se trabaja con un valor de latencia de referencia de 210 ms. Como se mencionó en la metodología, es esencial obtener un diccionario con los atributos de latencia de cada bloque y sus variaciones para utilizarlos en la DGX-1 para la búsqueda y el entrenamiento basados en estos valores. Inicialmente, dado que este método es semi-automático y requiere la fijación de ciertos hiperparámetros por parte del humano experto, se realizan gráficos de comparación entre estos bloques básicos para evaluar su rendimiento en este hardware.

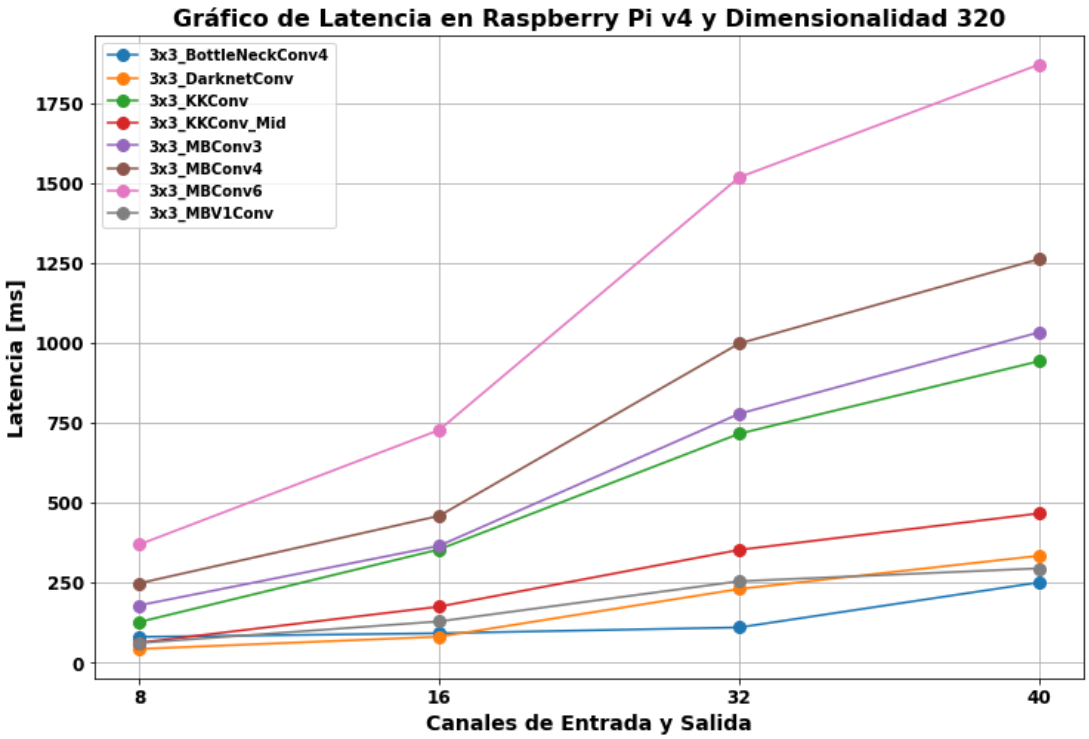

Figura 48: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 320.

Considerando los resultados de latencia presentados en la Figura 48 y teniendo en cuenta que el valor de referencia total para el extractor de características debe ser de 210 ms, se ha determinado que los valores de canal para los bloques pre-etapas deben ser establecidos en 8. Además, se busca minimizar el uso de esta dimensionalidad para todos los bloques básicos, ya que se ha observado que representa un costo elevado en términos de latencia para este hardware objetivo. Estas decisiones se basan en la necesidad de optimizar el rendimiento del extractor de características en la Raspberry Pi v4 y garantizar que se ajuste dentro de los límites de latencia establecidos.

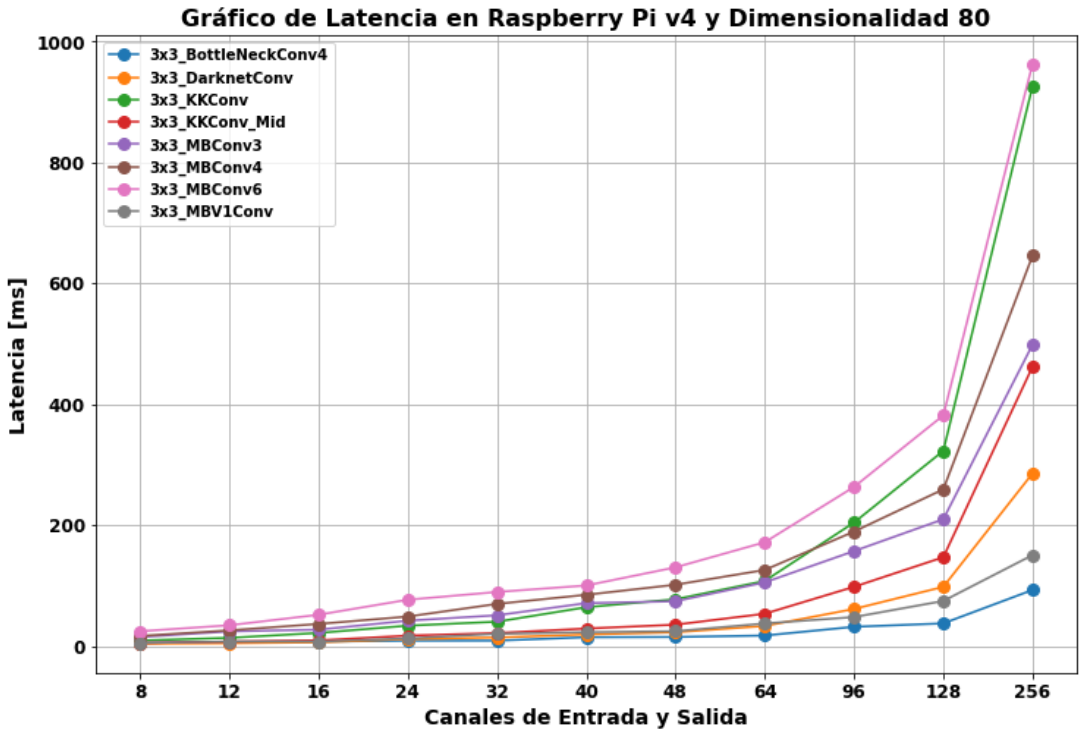

Figura 49: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 80.

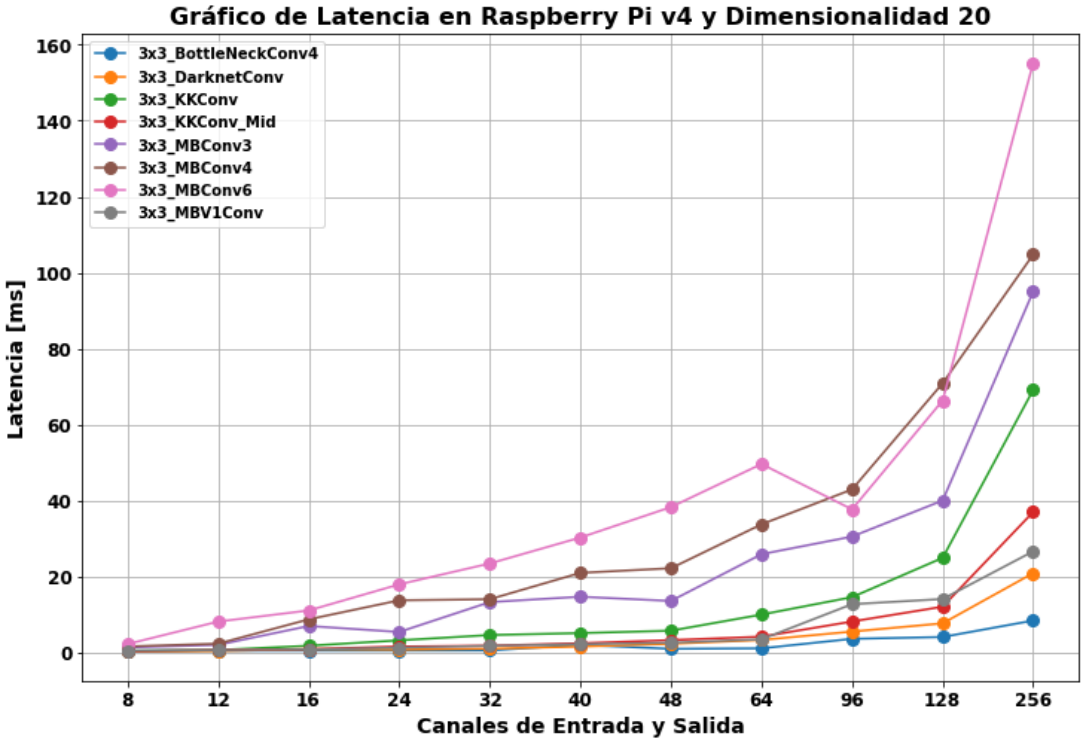

Figura 50: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 20.

Los gráficos presentados en las Figuras 49 y 50 muestran que se mantienen las tendencias exponenciales en los bloques básicos a medida que aumenta la cantidad de

canales. Los tipos de bloques BottleNeck y MBV1 siguen siendo los más eficientes en términos de latencia, especialmente en las etapas de canales más altos. En este hardware objetivo, que utiliza una CPU con una capacidad de paralelización menor en comparación con una GPU, se beneficia más de bloques con un número reducido de canales, ya que se pueden utilizar en mayor cantidad de bloques secuenciales (lo cual será definido por el método NAS).

Basándonos en esta observación y en los datos de los gráficos, se hace evidente la necesidad de utilizar una baja cantidad de canales para cada una de las etapas que componen el extractor de características NAS en este hardware. Sin embargo, es importante destacar que el aumento significativo en la latencia a medida que se incrementan los canales altos indica que no es eficiente utilizar bloques con una alta cantidad de canales, ya que esto limitaría el paralelismo en la CPU.

Por lo tanto, se acota la latencia máxima para representar en los gráficos a 200 ms y 50 ms para las dimensionalidades más bajas, lo que nos permite obtener una mejor visualización del comportamiento de los bloques básicos en canales bajos y tomar decisiones más acertadas sobre la configuración adecuada para el extractor de características en la Raspberry Pi v4.

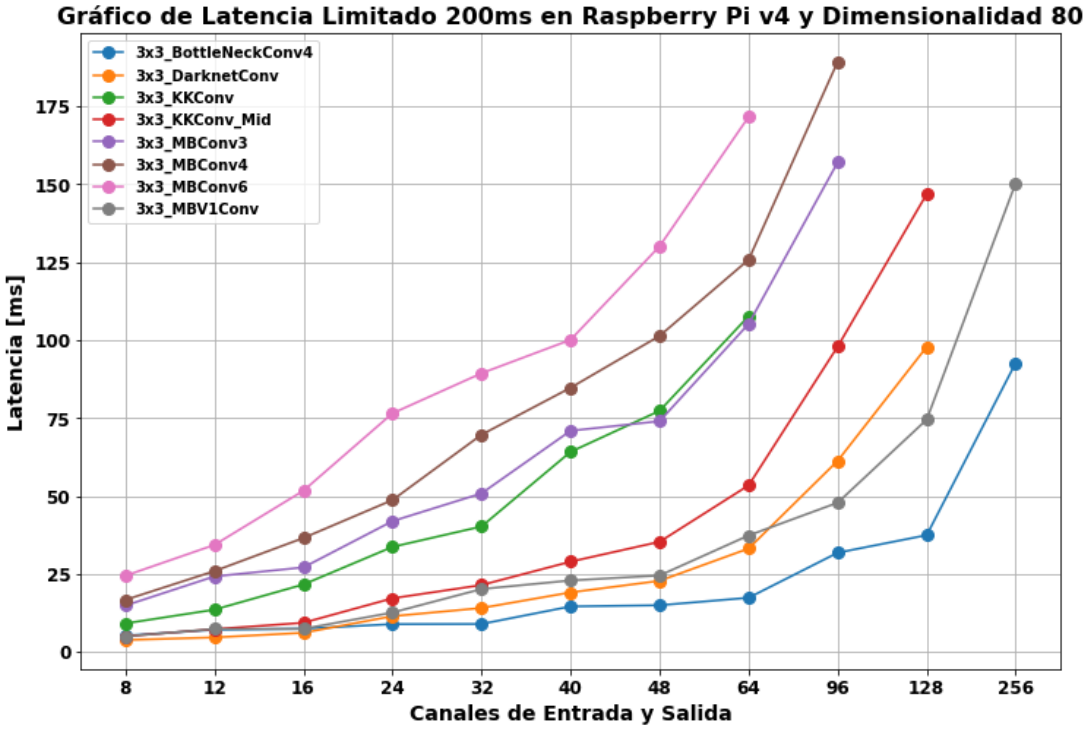

Figura 51: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 80, Acotado a 200ms.

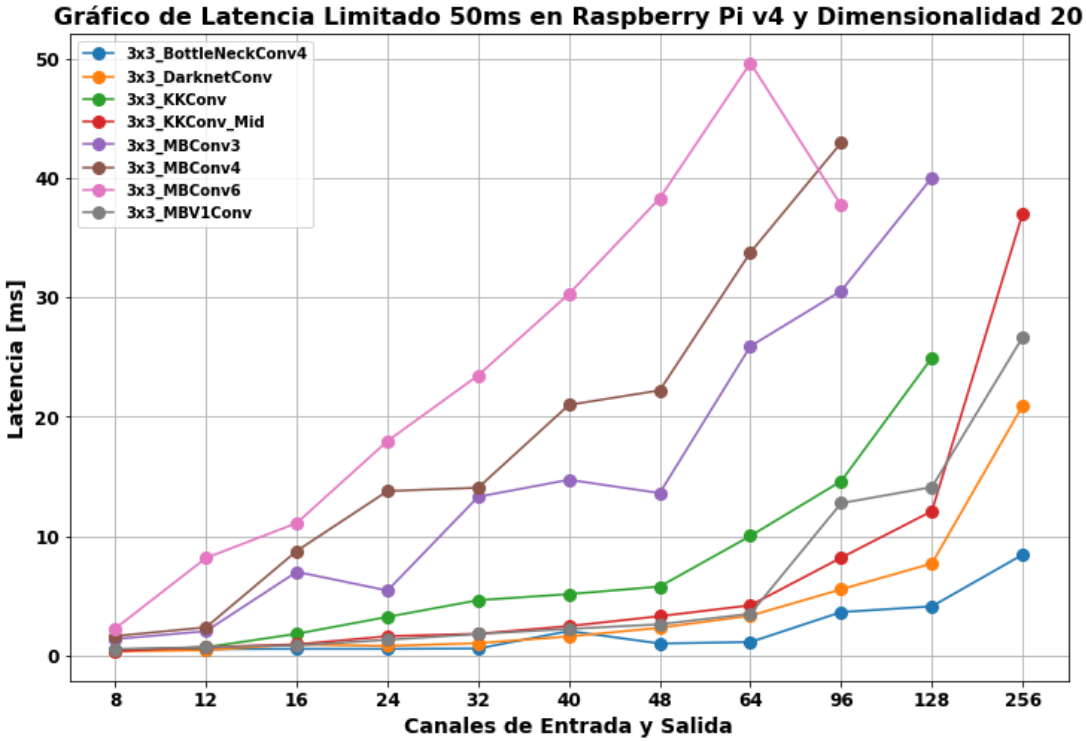

Figura 52: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 20, Acotado a 50ms.

Los gráficos presentados en las Figuras 51 y 52 muestran una similitud en los valores de latencia para los bloques tipo KK_Mid, DK, BL y MBV1 en canales bajos. Sin embargo, los bloques MBV2 resultan ser los más costosos en términos de latencia en todas las dimensionalidades, lo que confirma que no son eficientes para este hardware específico, que utiliza una CPU con limitaciones en paralelización.

Estos resultados también sugieren que los bloques KK y DK son opciones factibles para todas las dimensionalidades en las etapas, ya que son eficientes en términos de latencia, especialmente en configuraciones con pocos canales.

Por el lado de la variación de kernel para los bloques de la familia MBV1 Y MBV2, si bien se menciona en el punto 3.8.2.2 que se omitirá el uso de kernel 5x5 para los bloques de tipo MBV2, se incorporan en este análisis para validar esta idea en conjunto a ver el funcionamiento de los kernel 5x5 en MBV1, obteniendo los siguientes gráficos.

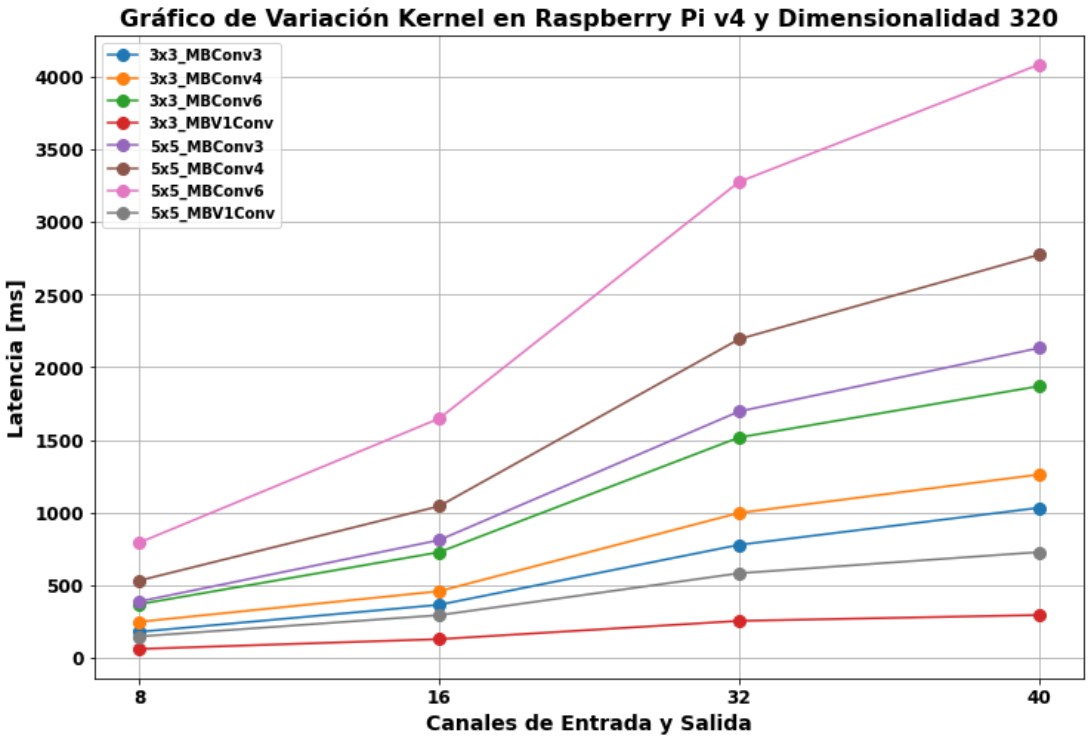

Figura 53: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 320, Variación de kernel 3x3 y 5x5.

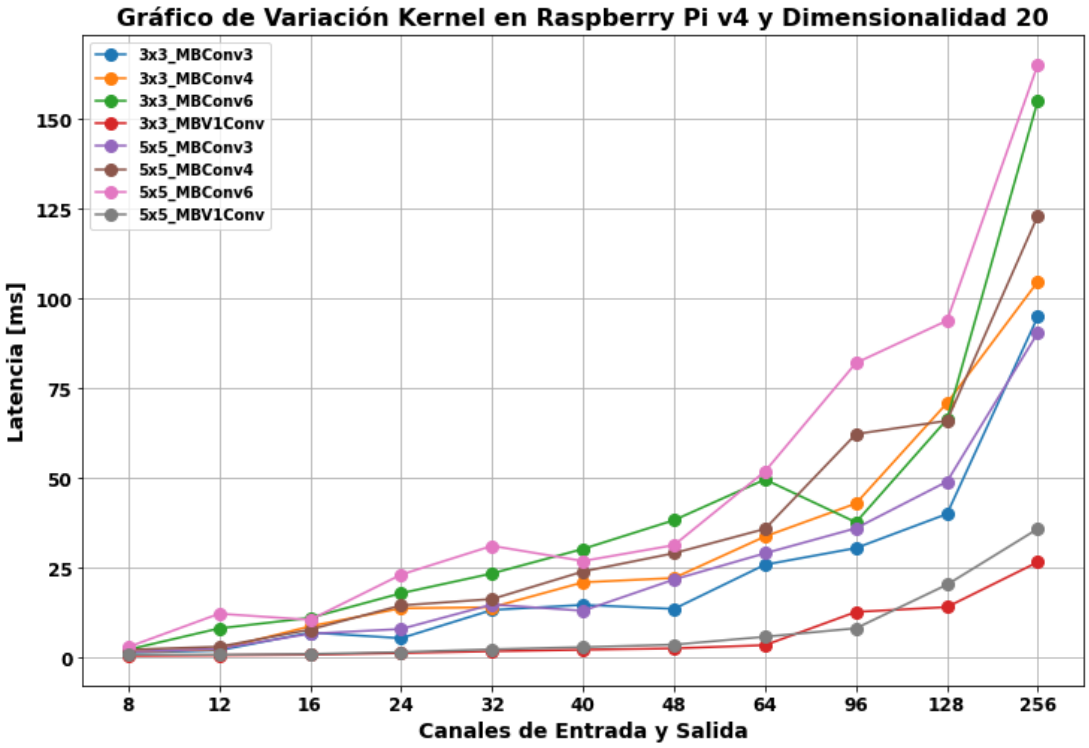

Figura 54: Gráfico de Latencia en Raspberry Pi v4, Dimensionalidad 20, Variación de kernel 3x3 y 5x5.

En las figuras 53 y 54 se observa un leve aumento en latencia para variaciones de kernel, siendo estas más elevadas en las dimensionalidades más altas, pero no presentan un cambio brusco o significativo lo suficientemente elevado para descartar su uso, de esta manera quedan incluidas en el espacio de búsqueda y el método NAS será el encargado de definir qué tipo de variación de kernel utilizar.

Ya teniendo un primer acercamiento y análisis de los tipos de bloques en el hardware objetivo Raspberry Pi v4 se fijan los hiperparámetros a utilizar:

- **Latencia esperada de la red neuronal NAS:** 210 ms.
- **Uso de SE/eSE:** No se aplica ninguna variación debido a sus costes computacionales, en especial para dispositivos embebidos.
- **Resolución de entrenamiento y búsqueda:** 640x640 píxeles.
- **Valor de lambda:** 7.
- **Alpha:** Se fija en 2.0 para mantener la consistencia con el entrenamiento diseñado para RetinaFace [Deng et al., 2019] en Pytorch, permitiendo así comparaciones precisas.
- **Batch de Entrenamiento y Búsqueda:** 32.
- **Bloque Básico NAS Inicial (primer MixedEdge en la etapa previa):** MBV1. Se utiliza en un principio este tipo de bloque básico para aprovechar su bajo coste computacional.

Finalmente, se tiene el diseño de los experimentos mostrados en la Tabla 36.

Tabla 36: Diseño de experimentos para primera iteración de Método FaceNAS en Raspberry Pi v4.

	Tipo de Bloque y Cantidad de Bloques				Canales			
Nombre Test	Stage1	Stage2	Stage3	Stage4	S1	S2	S3	S4
R1	MBV1+MBV2|4	MBV1+MBV2|6	BL+MBV1+MBV2|9	BL+MBV1+MBV2|6	32	64	128	256
R2	MBV1+MBV2|4	MBV1+MBV2|6	MBV1+MBV2|9	MBV1+MBV2|6	32	64	128	256
R3	MBV1|4	MBV1|6	MBV1|9	MBV1|6	32	64	128	256
R4	KK+DK|4	MBV1+KK+DK|6	MBV1|9	MBV1|6	32	64	128	256
R5	MBV1+KK+DK|4	MBV1+KK+DK|6	MBV1+KK+DK |9	MBV1+KK+DK |6	32	64	128	256
R6	KK+DK|4	KK+DK|6	KK+DK|9	KK+DK|6	32	64	128	256
R7(3x3)	MBV1|4	MBV1|6	MBV1|9	MBV1|6	32	64	128	256
R8	KK+DK|4	KK+DK|6	MBV1|9	MBV1|6	32	64	128	256
R9	MBV1|4	MBV1|6	MBV2|9	MBV2|6	32	64	64	96
R10	MBV2|4	MBV2|6	MBV1|9	MBV1|6	24	32	128	256

Se plantea el experimento R7 como una variación del experimento R3, con la diferencia principal de que en R7 se restringe el uso de bloques MBV1 al kernel 3x3, mientras que en R3 se permitía que el método NAS variara entre kernel 3x3 y 5x5. Esta

restricción se basa en la observación de que en el hardware de Raspberry Pi v4 no se evidencia una diferencia significativa en términos de latencia entre las variaciones de kernel, lo que sugiere que permitir esta variación podría ser beneficioso.

Tabla 37: Resultados de experimentos para primera iteración de Método FaceNAS en Raspberry Pi v4.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
R1	87,48	83,92	64,58	78,66	203ms
R2	88,98	85,98	71,45	82,14	240ms
R3	89,19	86,66	71,84	82,56	206ms
R4	89,29	86,95	73,10	83,11	191ms
R5	91,77	89,36	76,75	**85,96**	243ms
R6	91,04	88,3	75,63	84,99	237ms
R7	90,10	87,22	72,15	83,16	321ms
R8	89,27	86,53	73,04	82,95	184ms
R9	87,49	84,46	67,64	79,86	**137ms**
R10	88,44	85,92	71,20	81,85	212ms

Tal como se muestra en la Tabla 37, como primer análisis de los resultados se puede observar que el experimento R7 si bien tiene un ligero aumento de precisión promedio con respecto a R3, su latencia es 50% más alta, por lo que a nivel general se tiene un mejor equilibrio en el experimento R3.

Excluyendo el experimento R7, todos los experimentos tienen una latencia similar a la de referencia utilizada de 210ms. El experimento R5 es el de mayor precisión seguido por el experimento R6 con una precisión un poco más baja, la semejanza entre ambos experimentos es el uso de los bloques básicos KK y DK a diferencia de que en el R6 se cuenta con MBV1 incluido en el espacio de búsqueda. Viendo la arquitectura del experimento R5 se tiene la arquitectura definida en la Tabla 38.

Tabla 38: Arquitectura interna del experimento R5.

Arquitectura Final, Experimento R5	
Bloques Básicos	Bloques Residuales
3x3_MBV1Conv_16	None
3x3_MAXPool	None
3x3_KKConv_32	None
3x3_KKConv_32	Identity

3x3_KKConv_Mid_32	Identity
S_3X3_KKConv_Mid_64	None
3x3_KKConv_Mid_64	Identity
3x3_DKConv_64	Identity
S_3X3_KKConv_Mid_128	None
3x3_DKConv_128	Identity
3x3_DKConv_128	Identity
3x3_DKConv_128	Identity
S_3x3_DKConv_256	None
3x3_KKConv_Mid_256	Identity
3x3_KKConv_Mid_256	Identity
1x1_Conv_256	-

Demostrando que, si bien estaba incluido el uso de MBV1, este no fue incorporado por el método NAS en la arquitectura para su uso y solo utilizó los bloques básicos KK y DK.

Frente a estos resultados, se trabaja con el experimento R5, que logra la mayor precisión. Se mantiene la arquitectura del experimento R5 y solo se modifica su primer MixedEdge que en su configuración original empleaba un bloque MBV1. En este contexto, se exploran dos alternativas para reemplazar el bloque MBV1: los bloques KK_Mid y DK. Se eligen estos dos tipos de bloques debido a su eficiencia en términos de latencia cuando se aplican en canales bajos. La finalidad de estos experimentos es valuar su capacidad para mejorar la precisión aprovechando de que no aumentan en gran medida la carga computacional. Los experimentos por realizar son:

- **R5_V2:** Consiste en modificar el primer bloque básico de MBV1 por un bloque KK_Mid.
- **R5_V3:** Consiste en modificar el primer bloque básico de MBV1 por un bloque básico del tipo DK.

Los resultados obtenidos fueron los mostrados en la Tabla 39.

Tabla 39: Resultados de variaciones de experimento R5.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
R5_V2	92,65	90,43	79,45	**87,51**	265ms
R5_V3	92,65	90,24	78,87	87,25	**226ms**

Se aprecia en los resultados de la tabla anterior una mejora importante con esta modificación sobre la red, mostrando la efectividad que tienen estos tipos de bloques para este dispositivo.

En base a estos resultados, se propone realizar dos experimentos más que corresponden a los diseños mostrados en la Tabla 40.

Tabla 40: Diseño de experimentos para segunda iteración de Método FaceNAS en Raspberry Pi v4.

	Tipo de Bloque y Cantidad de Bloques				Canales			
Nombre Test	Stage1	Stage2	Stage3	Stage4	S1	S2	S3	S4
R11	DK|4	DK|6	DK|9	DK|6	32	64	128	256
R12	KK|4	KK|6	KK|9	KK|6	32	64	128	256

Teniendo en cuenta que se modifica desde ahora el MBV1 que se utilizaba al inicio por las dos variaciones de KK_Mid y DK. Los resultados obtenidos posterior a la obtención de su arquitectura NAS, variación V2 y V3, y entrenamiento de estos, son los mostrados en Tabla 41.

Tabla 41: Resultados de experimentos y su variación para segunda iteración de método FaceNAS en Raspberry Pi v4.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
R11_V2	92,97	90,7	78,73	**87,47**	232ms
R11_V3	92,74	90,43	78,43	87,20	223ms
R12_V2	91,56	89	76,32	85,63	130ms
R12_V3	91,55	88,78	75,95	85,43	**126ms**

Luego de obtener los resultados mostrados en la tabla anterior, se aplicó una modificación basada en el trabajo de investigación Designing Network Design Spaces [Radosavovic et al., 2020] de FAIR. Este estudio menciona que las activaciones Swish [Ramachandran et al., 2017] y su variante HardSwish [Howard et al., 2019], creada en la investigación de la MobileNetV3 [Howard et al., 2019], funcionan de manera más efectiva en redes pequeñas, como las obtenidas en estos experimentos. Por otro lado, para redes más grandes, se prefiere la activación ReLU [Nair & Hinton, 2010] y sus variantes, como LeakyReLU [Xu et al., 2015].

Siguiendo esta idea, se llevaron a cabo experimentos para intentar mejorar estas redes únicamente modificando la función de activación. Se implementó esta mejora, optimizando las áreas donde se incluyó la nueva función de activación, limitándola solo a

las etapas de baja dimensionalidad (etapa 3 y 4) en estos tres experimentos (R5, R11 y R12). Los resultados obtenidos en términos de precisión y latencia se presentan en la Tabla 42.

Tabla 42: Resultados de experimentos R5, R11 y R12 y sus variaciones en la función de activación en Raspberry Pi v4.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
R5_V2	92,65	90,43	79,45	87,51	265ms
R5_V2_HS	92,81	90,54	79,21	87,52	233ms
R5_V3	92,65	90,24	78,87	87,25	226ms
R5_V3_HS	92,49	90,31	78,71	87,17	230ms
R11_V2	92,97	90,7	78,73	87,47	232ms
R11_V2_HS	93,02	90,70	79,17	**87,63**	235ms
R11_V3	92,74	90,43	78,43	87,20	223ms
R11_V3_HS	92,90	90,48	78,22	87,20	221ms
R12_V2	91,56	89	76,32	85,63	130ms
R12_V2_HS	91,76	89,24	76,78	85,93	136ms
R12_V3	91,55	88,78	75,95	85,43	126ms
R12_V3_HS	91,68	89,17	76,48	85,78	**125ms**

Luego de todas las iteraciones y variaciones realizadas en el extractor de características NAS, se ha determinado que el experimento R11_V2_HS cuenta con la mayor precisión y una latencia total similar a la de los otros experimentos. Por lo tanto, se ha seleccionado como la red final obtenida a través del método NAS. Además, debido a su alta precisión y muy baja latencia, se incluirá en la tabla final el experimento R12_V3_HS, ya que también supera la precisión de la RetinaFace mobile con una disminución significativa en la latencia. Estos resultados finales se compararán con la versión de la red existente de RetinaFace mobile en este hardware, lo que demuestra en la Tabla 43 la capacidad de adaptación que el método NAS ha logrado al utilizar únicamente los atributos de latencia de los bloques básicos para optimizar su bajo costo computacional y lograr una alta precisión.

Tabla 43: Resultados finales y comparación con RetinaFace base y mobile para Raspberry Pi v4.

Nombre Modelo	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
RetinaFace Mobile (Mobile0.25)	89,71	86,83	71,47	82,67	204ms

RetinaFace Base (ResNet50)	95,27	93,83	84,33	**91,14**	1521ms
R11_V2_HS	93,02	90,70	79,17	87,63	235ms
R12_V3_HS	91,68	89,17	76,48	85,78	**125ms**

Se ha logrado una mejora de la precisión del 4,96% con solo un aumento del 15% en la latencia en el caso del experimento R11_V2_HS en comparación con RetinaFace mobile. Por otro lado, en comparación con RetinaFace base, se ha obtenido una disminución de aproximadamente 3,51% en la precisión con una mejora en la latencia del 84,5%. La R12_V3_HS es 1,63 veces más rápida que la versión de RetinaFace mobile y logra una precisión un 3,1% más alta. Comparándola con la versión RetinaFace base, a pesar de tener una disminución del 5,36% en la precisión, esta nueva red es hasta 12,16 veces más rápida que la original. En resumen, se han obtenido dos arquitecturas finales que ofrecen un rendimiento competitivo tanto en precisión como en inferencia. Las arquitecturas finales, R11_V2_HS y R12_V3_HS, tienen su composición descrita en las Tablas 44 y 45 respectivamente.

Tabla 44: Arquitectura de R11_V2_HS.

Modelo R11_V2_HS	Bloques Básicos	Bloques Residuales	Función de activación
First Conv	3x3_Conv_8	-	LeakyReLU
Pre-Stage	3x3_KKConv_Mid_16 3x3_MAXPool	- -	LeakyReLU -
Stage 1	3x3_DKConv_32 [3x3_DKConv_32] x 3	- Identity	LeakyReLU [LeakyReLU] x 3
Stage 2	S_3x3_DKConv_64 [3x3_DKConv_64] x 2	- [Identity] x 2	LeakyReLU [LeakyReLU] x 2
Stage 3	S_3x3_ DKConv _128 [3x3_ DKConv _128] x 8	- [Identity] x 8	HardSwish [HardSwish] x 8
Stage 4	S_3x3_DKConv_256 [3x3_ DKConv _256] x 5	- [Identity] x 5	HardSwish [HardSwish] x 5
Last Conv	1x1_Conv_256	-	HardSwish

Tabla 45: Arquitectura de R12_V3_HS.

Modelo R12_V3_HS	Bloques Básicos	Bloques Residuales	Función de activación
First Conv	3x3_Conv_8	-	LeakyReLU
Pre-Stage	3x3_DKConv_16 3x3_MaxPool	- -	LeakyReLU -
Stage 1	3x3_KKConv_32	-	LeakyReLU
Stage 2	S_3x3_KKConv_64	-	LeakyReLU
Stage 3	S_3x3_KKConv_Mid_128 3x3_KKConv_Mid_128	- Identity	HardSwish HardSwish
Stage 4	S_3x3_ KKConv_Mid_256 3x3_ KKConv_Mid _256	- Identity	HardSwish HardSwish
Last Conv	1x1_Conv_256	-	HardSwish

4.3 Jetson Nano

La Jetson Nano es un hardware embebido y con recursos limitados como la Raspberry Pi. Su diferenciación es que la Jetson Nano contiene una GPU interna que permite trabajar con Deep Learning y lograr paralelizar operaciones como en este caso las operaciones convolucionales. De esta manera puede hacer inferencia en tiempo real, al menos para trabajos de batch 1. Al ser hardware embebido, la red a utilizar como base es la RetinaFace mobile obteniendo los resultados mostrados en la Tabla 46.

Tabla 46: Precisión y latencia de RetinaFace mobile en Jetson Nano.

Nombre Modelo	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,640,640]
RetinaFace Mobile (Mobile0.25)	89,71	86,83	71,47	82,67	37,22ms

Con esta información de referencia, se procede a obtener gráficos que analicen el comportamiento de los bloques básicos. Para esto, se utiliza una configuración de batch 1 y resolución 640x640, lo que permite comprender cómo se comportan los bloques básicos antes de diseñar los experimentos a llevar a cabo.

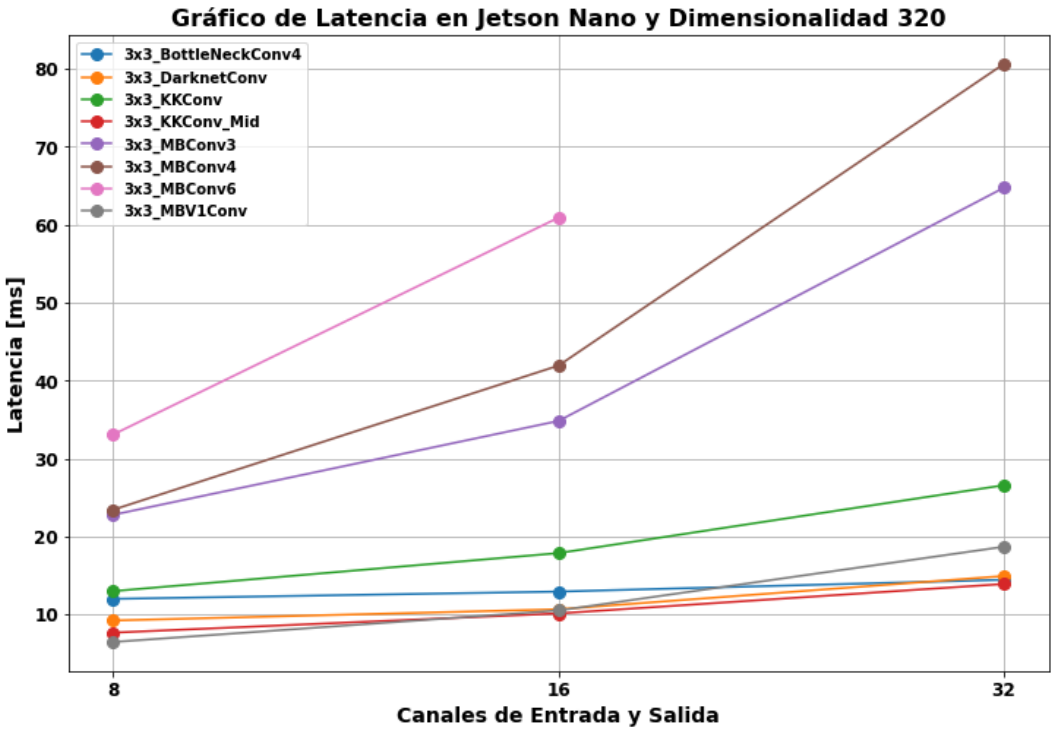

Figura 55: Gráfico de Latencia en Jetson Nano, Dimensionalidad 320.

En la Figura 55, se muestra un gráfico de latencia en la Jetson Nano con una dimensionalidad de 320 en la etapa previa. En este gráfico, se observa que la latencia de todos los bloques básicos es similar, con la excepción de los tipos MBV2. Dado que los bloques MBV2 tienen latencias significativamente más altas, se descarta su uso en esta sección de la etapa previa. Su uso no permitiría alcanzar la latencia de referencia buscada (45ms) debido a su alto costo en latencia, considerando que se trata solo del uso de un bloque.

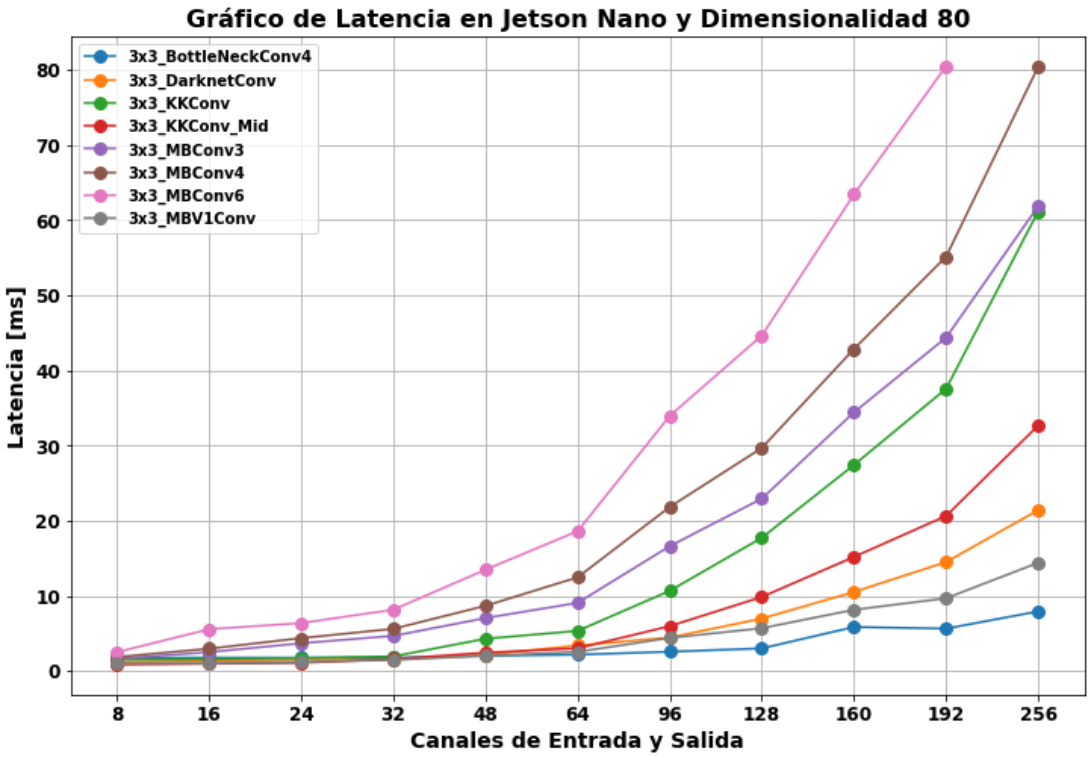

Figura 56: Gráfico de Latencia en Jetson Nano, Dimensionalidad 80.

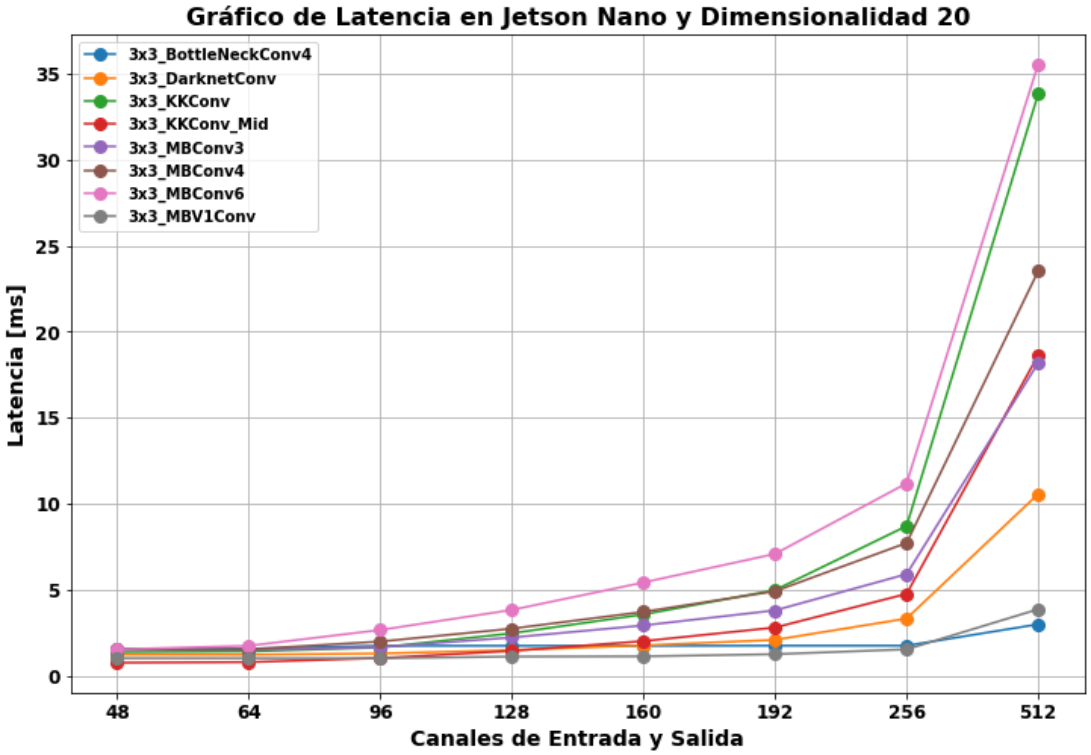

Figura 57: Gráfico de Latencia en Jetson Nano, Dimensionalidad 20.

En las Figuras 56 y 57, se presentan gráficos de latencia en la Jetson Nano con dimensionalidades de 80 y 20, respectivamente. Estos gráficos revelan que el bloque básico KK tiene un alto costo en términos de latencia, incluso mayor que algunos de los bloques básicos MBV2, que solían ser los más lentos. A partir de esta información, se buscará limitar la cantidad de canales en los experimentos que utilicen bloques básicos KK. Además, se considera que los bloques básicos BL, MBV1 y DK tienen un menor costo en términos de latencia o carga computacional, según se observa en los gráficos.

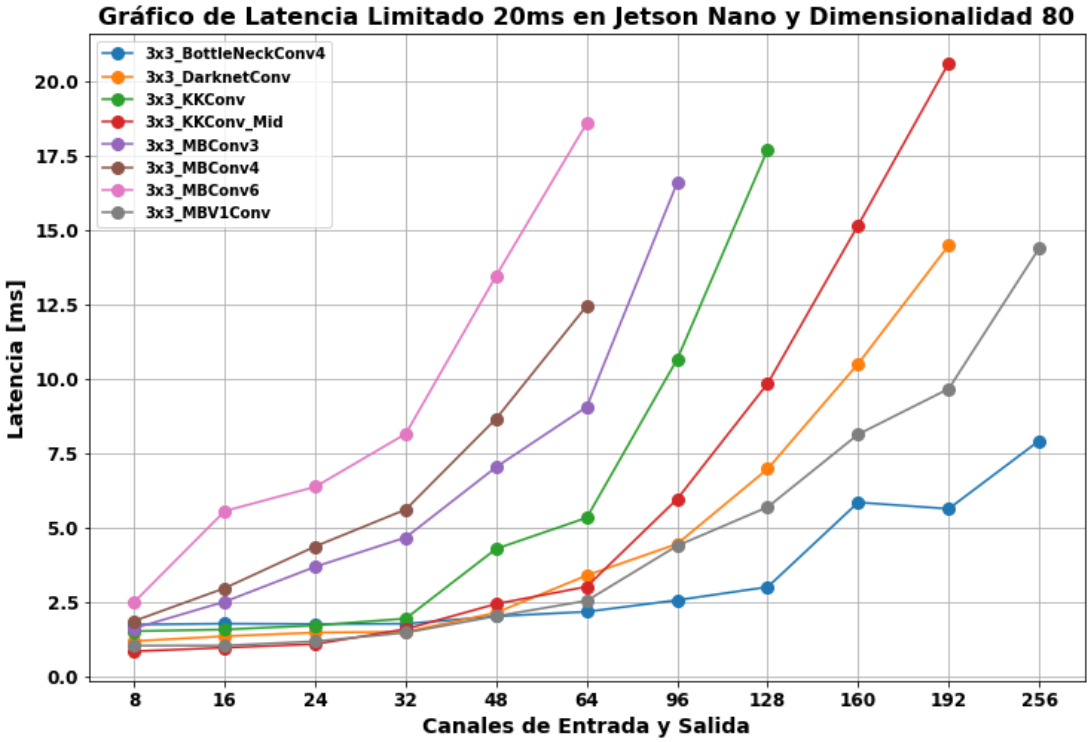

Figura 58: Gráfico de Latencia en Jetson Nano, Dimensionalidad 80, Acotado a 20ms.

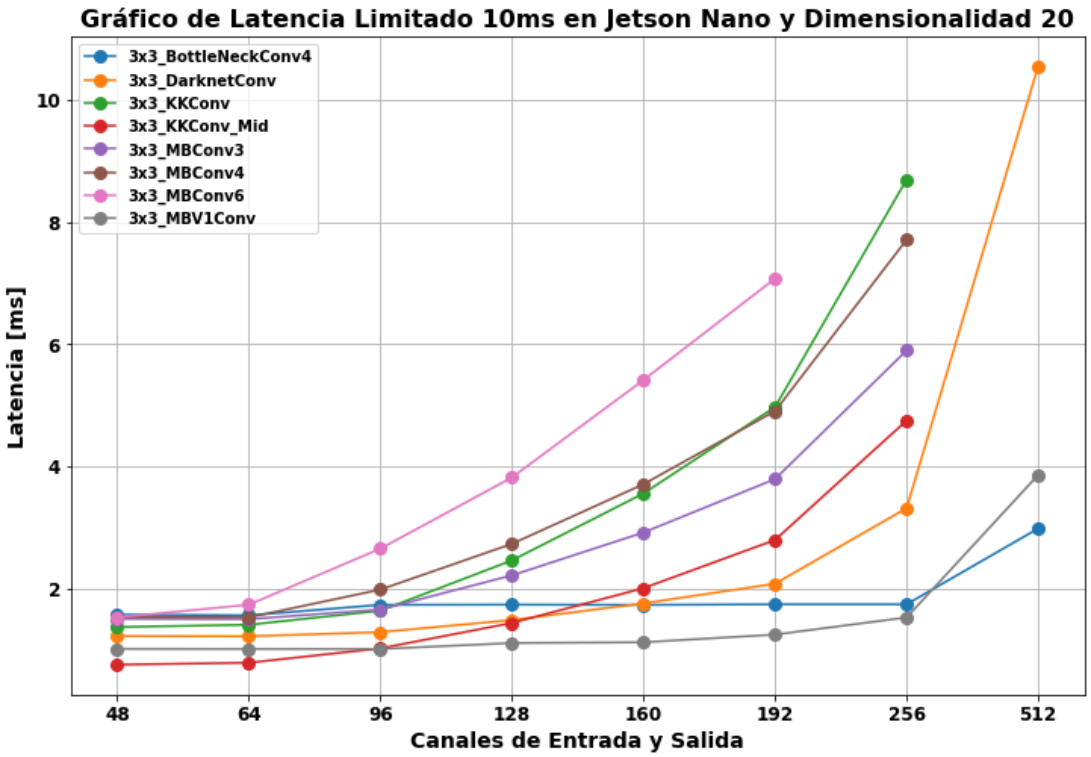

Figura 59: Gráfico de Latencia en Jetson Nano, Dimensionalidad 20, Acotado a 10ms.

Las Figuras 58 y 59 muestran gráficos de latencia en la Jetson Nano con limitaciones de latencia de 20 ms y 10 ms respectivamente, y con dimensionalidades de 80 y 20. Estos gráficos sugieren una similitud o una pequeña diferencia en las latencias de los bloques básicos para ciertos canales, y generalmente se observa un punto de inflexión en esta similitud alrededor del canal 64 (al menos para los bloques BL, MBV1, DK y KK_Mid).

Dado esto, se buscará un diseño que aproveche el campo receptivo más grande que se obtiene al utilizar bloques del tipo DK y KK en las primeras etapas (1 y 2). Por lo tanto, se dará énfasis a estos tipos de experimentos, sin descartar otras variaciones basadas en esta idea.

En cuanto a la comparativa entre el uso de kernel de 3x3 y 5x5 para los bloques tipo MBV1 y MBV2 en la Jetson Nano, los resultados obtenidos se presentan a continuación.

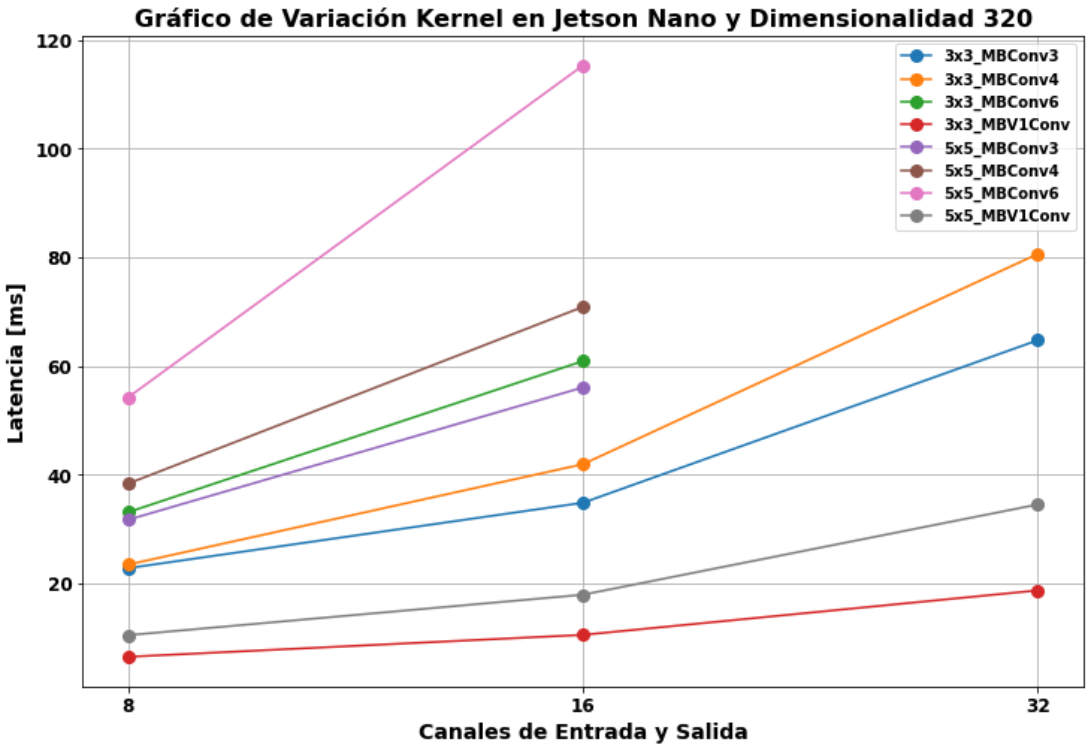

Figura 60: Gráfico de Latencia en Jetson Nano, Dimensionalidad 320, Variación de kernel 3x3 y 5x5.

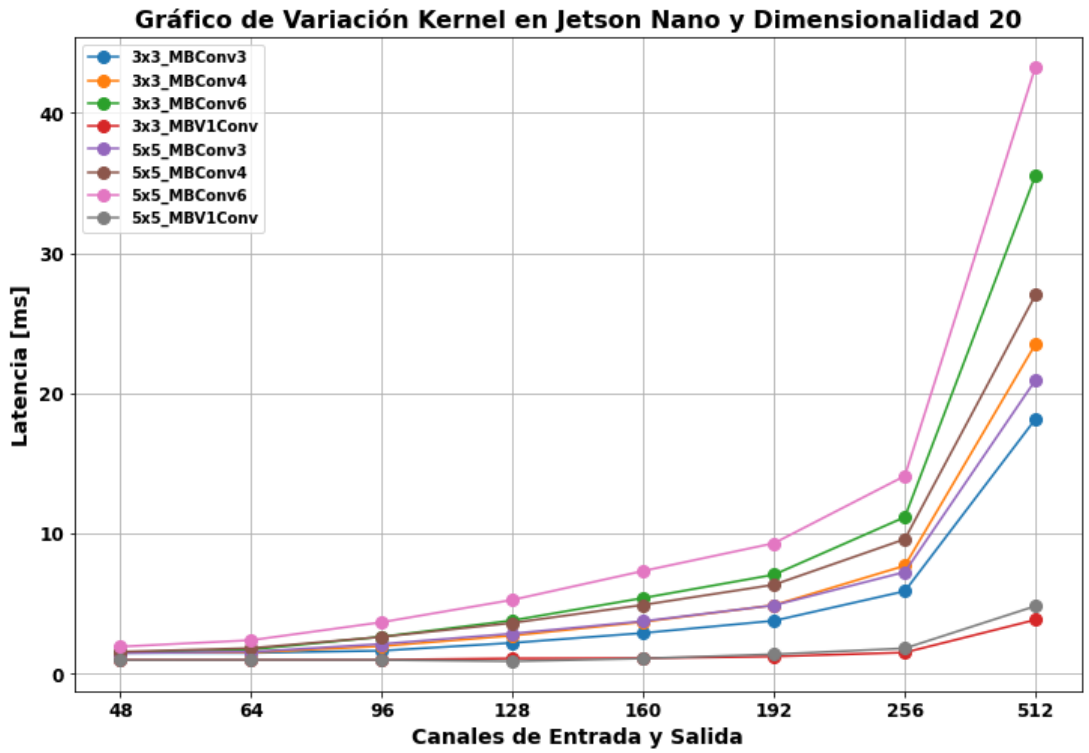

Figura 61: Gráfico de Latencia en Jetson Nano, Dimensionalidad 20, Variación de kernel 3x3 y 5x5.

Los gráficos presentados en las Figuras 60 y 61 muestran las latencias para las variaciones de kernel 3x3 y 5x5 en la Jetson Nano, con dimensionalidades de 320 y 20. Estos gráficos revelan que no existe una diferencia significativa entre las variaciones de kernel 3x3 y 5x5 en términos de latencia. Por lo tanto, no se restringirá el uso de kernel 5x5 en el diseño de los experimentos.

Dado estos análisis, los hiperparámetros a utilizar son los siguientes:

- **Latencia esperada de la red neuronal NAS:** 45 ms.
- **Uso de SE/eSE:** No se aplica ninguna variación debido a sus costes computacionales, en especial para dispositivos embebidos.
- **Resolución de entrenamiento y búsqueda:** 640x640 píxeles.
- **Valor de lambda:** 7.
- **Alpha:** Se fija en 2.0 para mantener la consistencia con el entrenamiento diseñado para RetinaFace en Pytorch, permitiendo así comparaciones precisas.
- **Batch de Entrenamiento y Búsqueda:** 32.
- **Bloque Básico NAS Inicial (primer MixedEdge en la etapa previa):** KK_Mid. Se utiliza este tipo de bloque básico para aprovechar su precisión y en variación Mid para no aumentar en gran manera la latencia.

Mientras que los diseños de los experimentos a realizar son los mostrados en la Tabla 47.

Tabla 47: Diseño de experimentos para primera iteración de Método FaceNAS en Jetson Nano.

	Tipo de Bloque y Cantidad de Bloques				Canales			
Nombre Test	Stage1	Stage2	Stage3	Stage4	S1	S2	S3	S4
J1	KK+DK|2	KK+DK|3	KK+DK+BL|12	KK+DK+BL|6	32	64	96	128
J2	KK+DK|2	KK+DK|3	MBV1|12	MBV1|6	32	64	128	256
J3	MBV2|2	MBV2|3	MBV2|12	MBV2|6	16	32	64	96
J4	MBV1|2	MBV1|3	MBV1|12	MBV1|6	32	64	96	128
J5	KK+DK|2	KK+DK|3	MBV1|12	MBV1|6	32	64	128	256
J6	MBV1|2	MBV1|3	BL|12	BL|6	32	64	128	256
J7	BL|2	BL|3	MBV1|12	MBV1|6	64	128	192	256
J8	KK+DK|2	KK+DK|3	MBV2|12	MBV2|6	32	64	64	96
J9	KK+DK|2	KK+DK|3	MBV1+MBV2|12	MBV1+MBV2|6	32	64	96	192
J10	KK+DK|2	KK+DK|3	KK+DK|12	KK+DK|6	24	32	64	128
J11	BL|2	BL|3	BL|12	BL|6	32	64	128	256
J12	KK+DK+BL|2	KK+DK+BL|3	KK+DK+BL|12	KK+DK+BL|6	24	32	128	192
J13	KK+DK|2	KK+DK|3	BL|12	BL|6	24	32	128	192
J14	DK|2	DK|3	DK|12	DK|6	24	32	64	128
J15	KK|2	KK|3	KK|12	KK|6	24	32	64	128

Luego de la búsqueda de los extractores de características por el método NAS propuesto y su posterior entrenamiento, se obtienen los resultados de la Tabla 48.

Tabla 48: Resultados de experimentos para primera iteración de Método FaceNAS en Jetson Nano.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,640,640]
J1	92,00	89,37	77,11	86,16	37,20ms
J2	91,55	89,37	76,98	85,96	38,28ms
J3	90,79	88,18	74,10	84,36	38,35ms
J4	89,67	86,94	72,62	83,08	37,27ms
J5	90,83	88,34	75,81	84,99	35,77ms
J6	91,19	88,73	75,92	85,28	37,29ms
J7	88,67	86,29	72,37	82,44	36,67ms
J8	91,29	88,74	75,48	85,17	38,62ms
J9	90,51	87,84	75,12	84,49	39,3ms
J10	91,97	89,47	77,63	**86,36**	41,95ms
J11	89,98	87,04	72,92	83,31	33,1ms
J12	91,25	88,52	75,42	85,06	34,07ms
J13	89,3	86,79	73,05	83,05	**28,17ms**
J14	90,95	88,27	74,85	84,69	32,6ms
J15	92,06	89,56	77,02	86,21	37,51ms

En la Tabla 48 se observa que el experimento de mejor desempeño es el J10, aunque con una diferencia mínima respecto a los experimentos J1 y J15. Sin embargo, es importante destacar que el experimento J10 presenta la latencia más alta entre todos los experimentos. Por lo tanto, para la próxima iteración, se ha decidido continuar con los experimentos J1 y J15, ya que ofrecen un equilibrio favorable entre precisión y latencia. Los diseños y componentes de estos dos experimentos se describen en las Tablas 49 y 50, respectivamente.

Tabla 49: Arquitectura interna del experimento J1.

Arquitectura Final, Experimento J1	
Bloques Básicos	Bloques Residuales
3x3_KKConv_Mid_16	None
3x3_MAXPool	None
3x3_KKConv_Mid_32	None
S_3X3_DKConv_64	1x1_Conv_Pool_64

S_3X3_DKConv_96	1x1_Conv_Pool_96
3X3_DKConv_96	Identity
3x3_DKConv_96	Identity
3x3_DKConv_96	Identity
S_3x3_DKConv_128	1x1_Conv_Pool_128
3x3_KKConv_Mid_128	Identity
3x3_KKConv_Mid_128	Identity
3x3_KKConv_Mid_128	Identity
3x3_KKConv_Mid_128	Identity
3x3_KKConv_Mid_128	Identity
1x1_Conv_256	-

Tabla 50: Arquitectura interna del experimento J15.

Arquitectura Final, Experimento J15	
Bloques Básicos	Bloques Residuales
3x3_KKConv_Mid_16	None
3x3_MAXPool	None
3x3_KKConv_24	None
S_3X3_KKConv_32	1x1_Conv_Pool_32
S_3X3_KKConv_64	1x1_Conv_Pool_64
3X3_KKConv_64	Identity
3x3_KKConv_64	Identity
3x3_KKConv_64	Identity
S_3x3_KKConv_128	1x1_Conv_Pool_128
3x3_KKConv_128	Identity
3x3_KKConv_128	Identity
1x1_Conv_256	-

Al igual que para el hardware objetivo Raspberry Pi v4, debido a que en ambos casos se busca una red que funcione para un dispositivo embebido, se implementa la modificación de las funciones de activación de LeakyReLU [Xu et al., 2015] a HardSwish [Howard et al., 2019] para las últimas dos etapas de la arquitectura de los experimentos J1 y J15. Los resultados para estas dos variaciones son los descritos en la Tabla 51.

Tabla 51: Resultados de experimentos J1, J15 y sus variaciones en la función de activación en Jetson Nano.

Nombre Test	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,320,320]
J1	92,00	89,37	77,11	86,16	**37,20ms**
J1_HS	91,88	89,45	77,17	86,17	37,4ms
J15	92,06	89,56	77,02	86,21	37,51ms
J15_HS	92,1	89,56	77,38	**86,35**	37,53ms

Con estos resultados obtenidos en la tabla anterior, se ve una variación muy baja en aumento de precisión para el experimento J1. Para el experimento J15 se obtiene un mayor aumento de precisión con una subida de latencia menor.

Teniendo esto en consideración, se elige el experimento J15_HS como el modelo final obtenido por el método NAS y se realiza una comparación entre el experimento J15_HS, RetinaFace mobile y RetinaFace base mostrado en la Tabla 52.

Tabla 52: Resultados finales y comparación con RetinaFace base y mobile para Jetson Nano.

Nombre Modelo	Easy (%)	Medium (%)	Hard (%)	Precisión Promedio (%)	Latencia de Extractor de características|[1,640,640]
RetinaFace Mobile (Mobile0.25)	89,71	86,83	71,47	82,67	**37,22ms**
RetinaFace Base (ResNet50)	95,27	93,83	84,33	**91,14**	613,64ms
J15_HS	92,1	89,56	77,38	86,35	37,53ms

El experimento J15_HS muestra una mejora de precisión de un 3,68% con respecto al modelo RetinaFace mobile, mientras que su latencia es similar por lo que cumple lo esperado por el método NAS desarrollado. En este caso de Jetson Nano si bien en los gráficos no mostraba que el uso de bloques básicos KK/KK_Mid fuera una buena decisión debido a su alto costo en latencia, el uso de estos bloques básicos con canales más bajos fue determinante para mostrar su eficiencia aún con menor cantidad de canales logrando mantener una latencia similar incluso a otros tipos de bloques básicos utilizados como diseño. Esto demuestra también la mejora que conlleva el uso de bloques del tipo KK aun cuando su coste computacional sea elevado, el disminuir este coste utilizado menor cantidad de canales logra ser óptimo igualmente. Para la comparación entre J15_HS y RetinaFace base, se tiene una diferencia de 4,79% en precisión, mientras que la latencia del J15_HS es 16,35 veces más rápida.

El experimento J15_HS muestra una mejora de precisión del 3,68% en comparación con el modelo RetinaFace mobile, manteniendo una latencia similar, lo cual cumple con las expectativas del método NAS desarrollado. En este caso de la Jetson Nano, aunque los gráficos inicialmente no sugerían que el uso de bloques básicos KK/KK_Mid fuera una buena elección debido a su alto costo en latencia, el uso de estos bloques con un menor número de canales resultó ser eficiente y permitió mantener una latencia similar a otros tipos de bloques básicos utilizados en el diseño. Esto demuestra la mejora que conlleva el uso de bloques del tipo KK, incluso cuando su costo computacional es alto. Reducir este costo mediante la utilización de menos canales resulta ser una estrategia óptima. En cuanto a la comparación entre J15_HS y RetinaFace base, se observa una diferencia del 4,79% en precisión, mientras que la latencia de J15_HS es 16,35 veces más rápida.

Finalmente, la arquitectura encontrada en el experimento J15_HS y su arquitectura interna se encuentra detallado en la Tabla 53.

Tabla 53: Arquitectura de J15_HS.

Modelo J15_HS	Bloques Básicos	Bloques Residuales	Función de activación
First Conv	3x3_Conv_8	-	LeakyReLU
Pre-Stage	3x3_KKConv_Mid_16 3x3_MaxPool	- -	LeakyReLU -
Stage 1	3x3_KKConv_24	-	LeakyReLU
Stage 2	S_3x3_KKConv_32	1x1_Conv_Pool_32	LeakyReLU
Stage 3	S_3x3_KKConv_64 [3x3_KKConv_64] x 3	1x1_Conv_Pool_64 [Identity] x 3	HardSwish [HardSwish] x 3
Stage 4	S_3x3_ KKConv_128 [3x3_ KKConv _128] x 2	1x1_Conv_Pool_128 [Identity] x 2	HardSwish [HardSwish] x 2
Last Conv	1x1_Conv_256	-	HardSwish

Capítulo 5

Conclusiones y trabajo futuro

5.1 Conclusiones

El método propuesto en esta tesis ha cumplido con éxito los objetivos y las hipótesis planteadas. Con respecto a la hipótesis inicial, el método desarrollado ha demostrado su capacidad para diseñar arquitecturas neuronales eficientes en términos de precisión y latencia. Además, ha demostrado su capacidad para adaptarse a diferentes hardware objetivo, incluso cuando la búsqueda y entrenamiento no se realizan en el mismo hardware. Esto se logra mediante la obtención precisa de las latencias y la eficiencia por bloque, basándose en un diccionario JSON previamente obtenido en el hardware objetivo y transferido al hardware donde se utiliza el método NAS propuesto.

Este método demuestra su capacidad para operar eficientemente con una latencia de referencia, lo que le permite, en línea con uno de los principales objetivos de esta tesis, buscar redes más precisas manteniendo un costo de latencia cercano al establecido por los requisitos del usuario o humano experto.

Los bloques KK-Block y DK-Block, a pesar de su alto costo computacional, han demostrado ser altamente eficientes en varios experimentos, siendo muy similar a lo encontrado en investigaciones actuales como [Chen et al., 2023], especialmente cuando se utilizan con un número reducido de canales. Estos bloques han destacado en términos de precisión-latencia, lo que resalta su utilidad en la optimización de arquitecturas. Por otro lado, los bloques MBV2-Block han mostrado el peor rendimiento frente a los otros bloques básicos, al menos en los hardware objetivo analizados, no son bloques eficientes para utilizar.

Es importante destacar que cada hardware tiene restricciones computacionales distintas, lo que afecta la eficiencia de los diversos bloques básicos en cada dispositivo. Esto se refleja en los diferentes factores de curva presentados en los gráficos de la sección 4, para cada bloque y dispositivo. Por lo tanto, el método crea arquitecturas adaptadas específicamente para cada hardware, aprovechando al máximo sus recursos computacionales disponibles.

Los experimentos realizados en la DGX-1, que cuenta con amplios recursos computacionales, han revelado que la arquitectura óptima es similar a la propuesta en el trabajo TResNet [Ridnik et al., 2020] detallada en la sección 2.1.3. Esta arquitectura utiliza bloques KK-Block en las primeras etapas con mayor dimensionalidad y bloques BL-Block en las últimas etapas con dimensionalidades más bajas. Esto se debe a la elección de bloques con un campo receptivo más amplio y, por ende, mayor costo computacional en

las primeras etapas para capturar características fundamentales antes de utilizar bloques más ligeros como BottleNeck en etapas posteriores.

En el caso de la Raspberry Pi, los resultados indican que el método propone arquitecturas con una mayor profundidad de capas, es decir, con más bloques secuenciales. Esto sugiere que, dado su menor poder de paralelización al trabajar en CPU, son óptimas las redes que requieren menos paralelización y dependen en mayor medida de utilizar más bloques secuenciales.

A pesar de que la Jetson Nano está equipada con una GPU, su capacidad de paralelización es limitada en comparación con las GPU de servidores como las de la DGX-1. Por lo tanto, debe trabajar con un límite más acotado de canales, que varía según el bloque básico utilizado, como se refleja en los gráficos de los experimentos realizados en la sección 4.3.

Finalmente, es crucial destacar que el método ha demostrado ser capaz de comprender los atributos de latencia de cada bloque básico y sus variaciones. Esto le permite generar arquitecturas optimizadas para restricciones específicas, en combinación con los hiperparámetros proporcionados por el humano experto. Por lo mismo, es esencial que el humano experto tenga un conocimiento preciso de los aspectos que aún no están automatizados por el método NAS para maximizar los resultados de la arquitectura encontrada.

5.2 Trabajo Futuro

Como trabajo a futuro, se contempla la posibilidad de adaptar la base de datos de detección a utilizar. Dentro de las posibles bases de dato, se encuentra la detección de personas, animales u objetos en general, como la base de datos COCO [Lin et al., 2014]. De igual manera, se puede explorar la posibilidad de ampliar y otorgar mayor flexibilidad al método NAS propuesto, permitiendo que el mismo método defina internamente valores relacionados con los canales de entrada y salida, así como la elección de bloques básicos para cada etapa, entre otros aspectos. Adicionalmente, la posibilidad de modificar la FPN por la utilizada en el trabajo de [Zhang et al., 2022].

El trabajo realizado en esta tesis se presenta como una base sólida para futuras investigaciones que busquen incorporar nuevos tipos de bloques básicos en el espacio de búsqueda. Esto debido a la facilitad de implementar nuevos bloques básicos en el método, considerando especialmente los avances recientes y en constante mejora como son los Transformers [Dosovitskiy et al., 2020, Vaswani et al., 2017] en el campo de visión computacional, donde ya existen trabajos que buscan llevar los Transformers a dispositivos móviles o embebidos como [Mehta & Rastegari, 2021, Li et al., 2022, Wadekar & Chaurasia, 2022], posicionándose como una potencial incorporación al método desarrollado para un futuro trabajo. La incorporación de estos bloques puede mejorar de manera significativa la capacidad de las redes de detección, superando los resultados alcanzados hasta la fecha. Por lo tanto, el potencial en el área de visión

computacional en el futuro es inmenso, debido principalmente al continuo progreso en investigación y publicaciones científicas que pueden ir complementándose para construir una base aún más sólida para futuras investigaciones.

Glosario

NAS (Neural Architecture Search): Es un proceso automatizado para diseñar y encontrar redes neuronales, en este caso acotado a detección de rostros. NAS busca optimizar la arquitectura de la red para maximizar su precisión en complemento a restricciones específicas.

FLOPs: En redes neuronales, representa el número total de operaciones de punto flotante requeridas para procesar una entrada o imagen a través de la red. Esta métrica se utiliza para evaluar el costo computacional que tiene una arquitectura de red neuronal.

Latencia: Se refiere al tiempo que toma procesar una entrada o imagen a través de la red neuronal hasta que se obtiene un resultado. Esta métrica es importante ya que permite evaluar el rendimiento de la red neuronal en un contexto real y no solo teórico.

Benchmark: En esta tesis, los benchmarks se componen de base de datos de imágenes y son utilizados para evaluar y comparar el rendimiento de diferentes redes neuronales, tanto existentes como las obtenidas mediante esta tesis usando NAS.

Hardware: Dispositivos físicos o servidores que serán utilizados para realizar cada una de las pruebas y experimentos de esta tesis.

GPUs: Unidades de procesamiento gráfico esenciales para el entrenamiento e inferencia de redes neuronales, gracias a su capacidad para realizar eficientemente procesos y cálculos de manera paralelizada.

Fully-connected: Este tipo de capas en una red neuronal conectan cada neurona con todas las neuronas de la capa anterior, comúnmente utilizadas para la clasificación o la extracción de características al final de la red.

Batch: Se refiere a un conjunto de datos de entrada, en este caso se habla de batch para un conjunto de entrada de imágenes para realizar entrenamiento o inferencia a través de las redes neuronales.

Bibliografía

[Bochkovskiy et al., 2020] Bochkovskiy, A., Wang, C., & Liao, H. M. (2020). *YOLOV4: Optimal speed and accuracy of object detection*. arXiv (Cornell University). https://arxiv.org/pdf/2004.10934v1

[Cai et al., 2018] Cai, H., Zhu, L., & Han, S. (2018). *ProxylessNAS: Direct neural architecture search on target task and hardware*. arXiv (Cornell University). https://arxiv.org/pdf/1812.00332.pdf

[Chen et al., 2023] Chen, H., Wang, Y., Guo, J., & Tao, D. (2023). *VanillaNet: The Power of Minimalism in Deep Learning*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2305.12972

[Cui et al., 2021] Cui, C., Gao, T., Wei, S., Du, Y., Guo, R., Dong, S., Lu, B., Zhou, Y., Lv, X., Liu, Q., Hu, X., Yu, D., & Ma, Y. (2021). *PP-LCNET: a lightweight CPU convolutional neural network*. arXiv (Cornell University). https://arxiv.org/pdf/2109.15099.pdf

[Dai et al., 2016] Dai, J., Li, Y., He, K., & Sun, J. (2016). *R-FCN: object detection via region-based fully convolutional networks*. arXiv (Cornell University), 29, 379-387. https://arxiv.org/pdf/1605.06409

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Li, F. (2009). *ImageNet: a large-scale hierarchical image database*. 2009 IEEE Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/cvpr.2009.5206848

[Deng et al., 2019] Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I., & Zafeiriou, S. (2019). *RetinaFace: Single-stage dense face localisation in the wild*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1905.00641

[Ding et al., 2021] Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., & Sun, J. (2021). *RepVGG: Making VGG-style ConvNets Great again*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2101.03697

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). *An image is worth 16x16 words: Transformers for image recognition at scale*. arXiv (Cornell University). https://arxiv.org/pdf/2010.11929

[Feng et al., 2022] Feng, Y., Yu, S., Peng, H., Li, Y., & Zhang, J. (2022). *Detect faces Efficiently: a survey and evaluations*. IEEE transactions on biometrics, behavior, and identity science, 4(1), 1-18. https://doi.org/10.1109/tbiom.2021.3120412

[Ge et al., 2021] Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). *YOLOX: Exceeding YOLO Series in 2021*. arXiv (Cornell University). http://export.arxiv.org/pdf/2107.08430

[Han et al., 2020] Han, D., Yun, S., Heo, B., & Yoo, Y. (2020). *Rethinking channel dimensions for efficient model design*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2007.00992

[He et al., 2016] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.90

[He et al., 2016] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Identity mappings in deep residual networks*. En Lecture Notes in Computer Science (pp. 630-645). https://doi.org/10.1007/978-3-319-46493-0_38

[He et al., 2018] He, K., Girshick, R., & Dollár, P. (2018). *Rethinking ImageNet Pre-training*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1811.08883

[He et al., 2018] He, T., Zhi, Z., Zhang, H., Zhang, Z., Xie, J., & Li, M. (2018). *Bag of Tricks for Image Classification with Convolutional Neural Networks*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1812.01187

[Howard et al., 2017] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: efficient convolutional neural networks for mobile vision applications*. arXiv (Cornell University). http://export.arxiv.org/pdf/1704.04861

[Howard et al., 2019] Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). *Searching for MobileNetV3*. arXiv (Cornell University). https://arxiv.org/pdf/1905.02244.pdf

[Hu et al., 2018] Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2018). *Squeeze-and-Excitation Networks*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2018.00745

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). *Densely Connected Convolutional Networks*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2017.243

[Jeong et al., 2022] Jeong, J., Kim, B., Yu, J., & Yoo, Y. (2022). *ERESFD: Rediscovery of the effectiveness of Standard convolution for Lightweight Face Detection*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2204.01209

[Lee et al., 2019] Lee, Y., Hwang, J. H., Lee, S., Bae, Y., & Park, J. (2019). *An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvprw.2019.00103

[Lee & Park, 2020] Lee, Y., & Park, J. (2020). *CenterMask: Real-Time Anchor-Free Instance Segmentation*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr42600.2020.01392

[Li et al., 2022] Li, Y., Wu, C., Fan, H., Mangalam, K., Xiong, B., Malik, J., & Feichtenhofer, C. (2022). *MVITV2: Improved Multiscale Vision Transformers for classification and detection*.

2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr52688.2022.00476

[Lin et al., 2014] Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). *Microsoft COCO: Common Objects in context*. En Lecture Notes in Computer Science (pp. 740-755). https://doi.org/10.1007/978-3-319-10602-1_48

[Lin et al., 2017] Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). *Feature Pyramid Networks for Object Detection*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2017.106

[Lin et al., 2020] Lin, T., Goyal, P., Girshick, R., He, K., & Dollár, P. (2020). *Focal loss for dense object detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2), 318-327. https://doi.org/10.1109/tpami.2018.2858826

[Lin et al., 2020] Lin, M., Chen, H., Sun, X., Qian, Q., Li, H., & Jin, R. (2020). *Neural architecture design for GPU-Efficient networks*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2006.14090

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (2016). *SSD: Single Shot MultiBox Detector*. En Lecture Notes in Computer Science (pp. 21-37). https://doi.org/10.1007/978-3-319-46448-0_2

[Liu et al., 2018] Liu, H., Simonyan, K., & Yang, Y. (2018). *DARTS: Differentiable Architecture Search*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1806.09055

[Luo et al., 2022] Luo, S., Li, X., & Zhang, X. (2022). *Bounding-box deep calibration for high performance face detection*. Iet Computer Vision, 16(8), 747-758. https://doi.org/10.1049/cvi2.12122

[Mehta & Rastegari, 2021] Mehta, S., & Rastegari, M. (2021). *MobileVIT: light-weight, general-purpose, and mobile-friendly vision transformer*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2110.02178

[Nair & Hinton, 2010] Nair, V. S., & Hinton, G. E. (2010). *Rectified linear units improve restricted Boltzmann machines*. International Conference on Machine Learning, 807-814. https://icml.cc/Conferences/2010/papers/432.pdf

[Nayman et al., 2021] Nayman, N., Aflalo, Y., Noy, A., & Zelnik-Manor, L. (2021). *HardCoRe-NAS: Hard Constrained DiffeRentiable Neural Architecture Search*. arXiv (Cornell University). https://arxiv.org/pdf/2102.11646.pdf

[Qi et al., 2023] Qi, D., Tan, W., Yao, Q., & Liu, J. (2023). *YOLO5Face: Why reinventing a face detector*. En Lecture Notes in Computer Science (pp. 228-244). https://doi.org/10.1007/978-3-031-25072-9_15

[Radosavovic et al., 2020] Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., & Dollár, P. (2020). *Designing network design spaces*. arXiv (Cornell University). http://export.arxiv.org/pdf/2003.13678

[Ramachandran et al., 2017] Ramachandran, P., Zoph, B., & Le, Q. V. (2017). *Searching for activation functions*. arXiv (Cornell University). https://arxiv.org/pdf/1710.05941.pdf

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.91

[Redmon & Farhadi, 2018] Redmon, J., & Farhadi, A. (2018). *YOLOV3: an incremental improvement*. arXiv (Cornell University). https://arxiv.org/pdf/1804.02767

[Ridnik et al., 2020] Ridnik, T., Lawen, H., Noy, A., Baruch, E. B., Sharir, G., & Friedman, I. (2020). *TResNet: High Performance GPU-Dedicated Architecture*. arXiv (Cornell University). https://arxiv.org/pdf/2003.13630.pdf

[Sandler et al., 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1801.04381

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). *Going Deeper with Convolutions*. arXiv (Cornell University). http://arxiv.org/pdf/1409.4842.pdf

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). *Rethinking the Inception Architecture for Computer Vision*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.308

[Tan et al., 2019] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2019.00293

[Tan & Le, 2019] Tan, M., & Le, Q. V. (2019). *EfficientNet: Rethinking model scaling for convolutional neural networks*. arXiv (Cornell University). https://arxiv.org/pdf/1905.11946v5

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. arXiv (Cornell University), 30, 5998-6008. https://arxiv.org/pdf/1706.03762v5

[Wadekar & Chaurasia, 2022] Wadekar, S. N., & Chaurasia, A. (2022). *MobileVITV3: Mobile-Friendly vision transformer with simple and effective fusion of local, global and input features*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2209.15159

[Wang et al., 2019] Wang, C., Liao, H. M., Yeh, I., Wu, Y., Chen, P., & Hsieh, J. (2019). *CSPNet: A new backbone that can enhance learning capability of CNN*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1911.11929

[Wightman et al., 2021] Wightman, R., Touvron, H., & Jeǵou, H. (2021). *ResNet strikes back: An improved training procedure in Timm*. arXiv (Cornell University). https://arxiv.org/pdf/2110.00476.pdf

[Wu et al., 2021] Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Liu, Y., & Zhang, L. (2021). *CVT: Introducing Convolutions to Vision Transformers*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2103.15808

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., & Li, M. (2015). *Empirical evaluation of rectified activations in convolutional network*. arXiv (Cornell University). https://arxiv.org/pdf/1505.00853.pdf

[Yang et al., 2016] Yang, S., Luo, P., Loy, C. C., & Tang, X. (2016). *WIDER FACE: A Face Detection Benchmark*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.596

[Zhang et al., 2022] Zhang, B., Li, J., Wang, Y., Tai, Y., Wang, C., Li, J., Huang, F., Xia, Y., Pei, W., & Ji, R. (2022). *ASFD: Automatic and Scalable Face Detector*. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2201.10781

[Zoph et al., 2018] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). *Learning Transferable Architectures for Scalable Image Recognition*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). *https://doi.org/10.1109/cvpr.2018.00907*