Xarxes neuronals sobre plataformes encastades per l'espai

Alejandro Labajos Martin de Villodres

Resum
Dins del marc de treball i de les necessitats del projecte Víbria de l'Institut d'Estudis Espacials de Catalunya, aquest treball busca implementar un sistema electrònic encastat per al reconeixement i classificació d'imatges sobre microcontroladors tipus ARM-Cortex i RISC-V fent ús de les instruccions vectorials d'aquesta darrera arquitectura emergent. Sota aquesta premissa, s'aprofundeix en l'ús del framework TensorFlow Lite per xarxes neuronals i la seva implementació per realitzar inferències de diferents models tant en llenguatge Python com C; degut a la naturalesa dels sistemes encastats al sector espacial, s'han el·liminat les dependències de llibreries externes típiques d'aquests projectes com OpenCV o Pillow.
Paraules clau
Xarxes Neuronals/Deep learning, processat d'imatges, reconeixement d'objectes, arquitectures RISC, espai

Abstract
Within the framework and project requirements of Víbria at the Institute of Space Studies of Catalonia, this study aims to implement an imatge recognition and classification embedded system on ARM-Cortex and RISC-V microcontrollers, levereaging the vector instructions of this latest emerging architecture. Under this premise, it deepens in using TensorFlow Lite framework for neural networks and its implementation in order to perform inferences of different models in both Python and C languages; due to space's embedded systems nature, typical dependences on big external libraries such as OpenCV or Pillow have been eliminated.
IndexTerms
Neural nets/Deep learning, image processing, object recognition, RISC architectures, space

1 Introducció - Context del treball

AVUI dia, el sector espacial està en consant expansió, tant per l'observació de l'univers com del nostre planeta, incorporant diverses tecnologies o creant-ne de noves amb l'objectiu de millorar el rendiment i la capacitat d'observació; això ha produït multitud de dispositius d'adquisició de dades creant la necessitat d'implementar sistemes de processament que facin comprensible la informació d'interès. En el projecte Víbria de l'Institut d'Estudis Espacials de Catalunya (IEEC) s'investiga per crear un nucli d'arquitectura RISC-V incorporant instruccions de l'extensió vectorial [1], amb l'objectiu (entre d'altres casos d'ús) d'executar-hi xarxes neuronals per al reconeixement d'objectes en imatges procedents d'alguns satèl·lits com pot ser el MENUT.

Dins d'aquest context, és present la necessitat de realitzar un estudi preliminar sobre les diferents xarxes neuronals i escollir un framework per a l'execució d'inferències, sobre un dels models pre-entrenats de codi obert disponibles; l'objectiu és val·lidar la seva funcionalitat per a l'arquitectura RISC-V, ja sigui en un emulador com QEMU o nativament en una placa amb un nucli d'aquesta arquitectura.

En aquest Treball de Fi de Grau es descriuen les etapes i els processos que s'han dut a terme, tenint present aquest objectiu i les necessitats de l'IEEC, per aconseguir l'execució de dos models basats en la xarxa neuronal *MobileNet*, pre-entrenats sobre un conjunt d'imatges genèriques anomenat *ImageNet* [2]. Les implementacions de les inferències s'han executat fent ús del framework *TensorFlow*, concretament, una versió reduïda, dissenyada i optimitzada específicament per a sistemes encastats de baix consum i memòria limitada; han sigut validades tant en llenguatge Python, típicament utilitzat en l'àmbit de les xarxes neuronals, com en llenguatge C, més propi dels sistemes encastats.

Cal destacar que aquest marc tecnol·lògic s'ha testejat en tres dispositius físics:

1. RaspberryPi 1 B+ (arquitectura ARMV6).
2. RaspberryPi 3 B+ (arquitectura ARMV8).
3. Visionfive V1 (arquitectura RISC-V64GC).

A més, fet que encara no hi ha cap *hardware* RISC-V amb instruccions vectorials disponible al mercat, s'ha realitzat el mateix procediment virtualment en QEMU, simulant una arquitectura RISC-V64GCV.

L'IEEC farà ús d'aquest treball per implementar un sistema encastat per la detecció de núvols i incendis *on the fly* al seu processador Víbria, és a dir, la informació serà processada directament a l'espai.

- *E-mail de contacte: 1455391@uab.cat*
- *Menció realitzada: Enginyeria de Computadors*
- *Treball tutoritzat per: Dr. Màrius Montón Macian (Departament de Microelectrònica i Sistemes Electrònics)*
- *Curs 2022/23*

2 OBJECTIUS

2.1 Generals

Com s'ha introduït prèviament, l'objectiu principal és aconseguir realitzar inferències sobre una xarxa de classificació d'imatges en una arquitectura RISC-V que suporti instruccions de l'extensió vectorial i comparar els resultats amb altres arquitectures.

2.2 Específics

Degut a les necessitats dels sistemes encastats, tant el framework com els models han de ser de mida reduïda, a més, les necessitats de l'IEEC requereixen que la implementació final faci ús de les instruccions vectorials de RISC-V per paral·lelitzar les operacions. També s'ha de tractar de reduir les dependències amb altres llibreries i minimitzar l'espai a disc que ocupa el codi; aquesta reducció millora el rendiment al fer un menor ús de memòria caché i millora el control del procés amb independència tecnològica, a més de rediïr el nombre de possibles inconvenients.

3 ENTORN DE DESENVOLUPAMENT

3.1 Models neurals

Les xarxes neuronals convolucionals o CNN son àmpliament utilitzades en àmbits com la classificació o compressió d'imatges i són el principal motiu d'interès d'aquest projecte.

D'entre la diversitat de models per a la classificació d'imatges s'ha de destacar aquells que han sigut entrenats amb grans conjunts d'imatges de propòsit general com el dataset anomenat ILSVRC 2012 [3], que conté més d'un milió d'imatges organitzades en mil categories; d'altra banda, han de ser fàcilment accessibles ja sigui a partir de webs d'aquest propòsit o proveïdes pel framework intèrpret escollit, *Tensorflow Lite* en el nostre cas. També cal considerar el pés dels models i el seu format ja que no poden ser excessivament grans pel sistema que estem tractant, com a màxim d'uns 20MBytes.

Per aquests motius, els models escollits són *MobileNet* V1 i V2: aquests models de deep learning van ser produïts per Google amb l'objectiu d'accelerar el temps de resposta sense detriment de la precisió, disminuint el nombre de

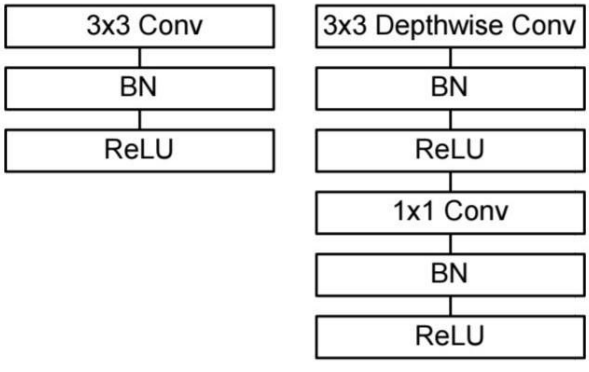
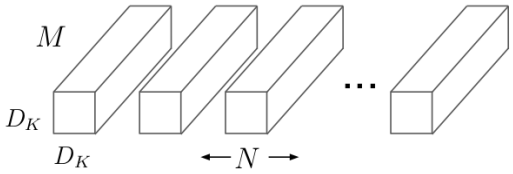
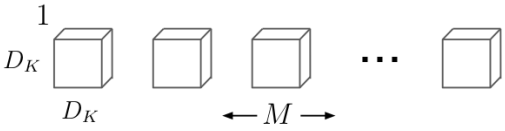


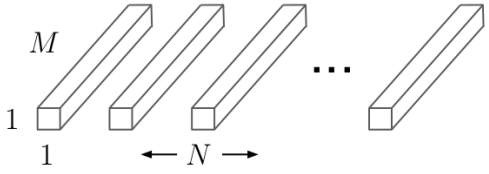
Fig. 1. Esquerra: capa convolucional estàndard, batchnorm i ReLu. Dreta: capa convolucional depthwise i 1x1 conv (pointwise convolution) amb els respectius batchnorm i ReLu. [4]



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Fig. 2. La suma de les capes convolucionals deepwise (b) i pointwise (c) s'anomena depthwise separable convolution. [5]

paràmentres. *MobileNet* modifica l'estructura de les capes en CNN tradicionals separant la convol·lució en dues capes [5].

On M és el nombre de canals d'entrada, N el nombre de canals de sortida i D_K la dimensió espcial del kernel.

Normalment, l'estructura de les xarxes és complexa i s'han de contemplar un gran nombre de paràmetres a les operacions, perjudicant el temps d'execució i dificultant la rapidesa que requereix un sistema amb les necessitats del projecte. La separació de capes que s'aplica en *MobileNet* redueix la complexitat de les operacions millorant el temps d'execució (Fig. 2).

La principal diferència entre les dues versions de la xarxa és l'adició d'una capa pointwise 1x1 conv (Fig. 2 apartat c) al principi de l'algorisme en la segona versió, mentre que la primera versió segueix l'esquema tradicional de depthwise convolution (Fig 1. Dreta). Ambdós models, tot i estar entrenats amb imatges de 224x224 píxels, accepten imatges de qualsevol mida superior a 32x32. L'obtenció de les dues versions s'ha obtingut descarregant la versió estàndard dels models oficials publicats per *TensorFlow* [6].

3.2 TensorFlow Lite 2.4

TensorFlow és un framework que proporciona multitud de funcions i classes amb l'objectiu de crear, entrenar i utilitzar models de xarxes neuronals de qualsevol àmbit. Un cop tenim un model entrenat i només volem executar-hi inferències, *TensorFlow* incorpora una biblioteca anomenada *Lite* dissenyada per a dispositius mòvils, *edge* o microcontroladors; els beneficis d'util·litzar *TensorFlow Lite* en comptes de la versió original és la reducció d'espai

dels models a memòria, degut a la compressió *FlatBuffer* que crea un arxiu del model .tf en format .tflite.

En sistemes *edge*, no cal la capacitat d'entrenar models i n'hi ha prou en realitzar la fase d'inferència de la forma més òptima possible; això s'aconsegueix tant amb l'esmentada compressió dels models com amb la reducció de la precisió dels paràmetres d'aquests (per exemple de float a int8); tot i així el canvi a format .tflite no és definitiu si sorgeix la necessitat de tornar a entrenar el model neural.

D'altra banda, *TF Lite* proporciona diferents *API*s segons el llenguatge de programació que volguem utilitzar, amb l'objectiu d'estalviar espai instal·lant només la part d'interès; per aquest projecte s'han utilitzat les llibreries per Python i C.

Com s'ha esmentat prèviament, els models poden acceptar imatges de qualsevol mida superior a la mínima, tot i estar entrenats amb una mida concreta; d'altre banda, a les CNN són habituals les reduccions de dimensions a les imatges, i degut al funcionament de *TF* a l'ajustar el tamany d'entrada (*padding rules*) és necessari determinar aquest tamany a l'executar inferències ja que dependrà de la dimensió. Per tal de simplificar el codi i les crides a funcions de *TF*, el redimensionat d'imatges s'ha fet amb una llibreria molt més simple i s'han mantingut les imatges d'entrada en 224x224 píxels.

El funcionament general de les inferències és sempre el mateix:

1. Redimensionat de la imatge a classificar.
2. Crida a *TF* per crear el model a partir del fitxer en format .tflite.
3. Crida a l'intèrpret de *TF Lite* amb la imatge de la dimensió esperada pel model (batch size x width x height x rgb channels).
4. Obtenció de resultats (*accuracy*, categoria predita, etc).

Aquest framework de desenvol·lupament s'ha obtingut del repositori oficial de *Tensorflow* [7].

3.3 Hardware

Per a l'implementació del sistema de classificació d'imatges s'han utilitzat tres plaques de desenvol·lupament anomenades *Single Board Computer* (SBC): les Raspberry Pi 1 i 3 models B+ i la placa VisionFive V1 que incorpora un multicore basat en RISC-V.

Una SBC consisteix en una única placa de circuit imprés que incorpora la memòria, l'entrada/sortida, el microprocessador i altres característiques que conformen un ordinador functional.

Avui dia amb l'augment de la potència dels microprocessadors, la majoría de SBC incorporen *heatsinks* o dissipadors de calor d'un tamany molt més reduït que els d'un ordinador personal.

L'alt nivell d'integració de les SBC degut a la implementació nativa dels sistemes HW en una sola placa redueix el temps i cost de fabricació d'aquests sistemes, a més de fer-les més compactes, confiables i eficients en consum d'energía que els computadors tradicionals. Degut a les seves característiques són utilitzades principalment per aplicacions integrades o sistemes encastats.

TABLE 1
CARACTERÍSTIQUES DISPOSITIUS HW

	Raspberry Pi1 B+	Raspberry Pi3 B+	VisionFive V1
System on Chip	Broadcom BCM2835	Broadcom BCM2837B0	StarFive JH7100
CPU	ARM1176JZF-S	ARM Cortex A53	2x ARM Cortex-A55
Arquitectura	ARMv6 de 32 bits	ARMv8 de 64 bits	RISC-V64GC
Freqüència CPU (GHz)	0.7	1.4	1.5
RAM (GB)	0.512	1	8
Sistema Operatiu	Raspbian	Arch Linux ARM	Arch Linux RISC-V

GHz = gigahertz, GB = gigabyte.

A la taula s'introdueixen les especificacions d'interès de les SBC utilitzades per implementar el sistema de classificació d'imatges (Table 1).

La VisionFive V1, és una placa experimental de l'empresa SiFive enviada a l'IEEC de forma gratuïta per a realitzar proves de cara a la sortida de la segona versió, ja disponible al mercat. Al ser un prototip no s'espera que tingui totes les funcionalitats que tindría una placa comercial d'aquestes característiques.

Degut a la sortida de la VisionFive V2 al mercat i ser una de les primeres plaques RISC-V totalment funcional (a diferència de la primera versió), es poden trobar gran quantitat de benchmarks d'aquesta i comparacions amb les diferents Raspberry que actualment acaparen gran part d'aquest mercat. Les estadístiques d'aquests benchmarks es discutiràn a la secció de Resultats; tot i així, com a introducció, podem afirmar que encara queda un llarg camí per recórrer en termes d'eficiència i aprofitament de recursos per l'arquitectura RISC-V en comparació amb el seu principal competidor.

3.4 Imatges

Com s'ha introduït prèviament, els models han sigut entrenats amb imatges del dataset ILSVRC 2012, compost per 1.3 milions d'imatges de mil categoríes diferents [8].

Per executar les inferències, s'ha optat per realitzar el test amb imatges reals i no imatges ideals pre-escollides, amb les dificultats que introdueix aquesta implicació: problemes d'il·luminació, borrositat, etc; obtenint però un sistema amb uns resultats més propers a una implementació real. Per aquest motiu, el dataset utilitzat s'ha extret de Kaggle; aquesta pàgina web es dedica únicament a l'àmbit de l'aprenentatge computacional i l'anàlisi de grans volums de dades. Principalment és utilitzada per publicar reptes d'aprenentatge computacional i propostes de datasets o models neurals per l'investigació

El conjunt de test escollit consisteix en dos milers d'imatges d'animals de diferents qualitats, molt utilitzada per *MobileNet* en la introducció a *TensorFlow* i xarxes neuronals per *embedded systems*, degut a estar redimensionades a 224x224 píxels i consistir en categoríes presents al dataset ImageNet amb que han sigut entrenats els models.

3.5 Altres eines

1. Gestió d'imatges:

   1.1) Pillow és una llibreria de codi obert que suporta diferents formats d'imatge per Python. Permet llegir imatges de múltiples formats i redimensio-

narles.

1.2) *Ujpeg* és una petita llibreria per redimensionat d'imatges en C que evita utilitzar grans llibreries com *OpenCV*, possiblement massa pesades per només fer-les servir en un *resize*. *Ujpeg* ocupant només 37,4KBytes, ideal per aquest projecte. El gran inconvenient d'aquest framework és la restricció a imatges de format .jpeg però no és un inconvenient per al projecte al ser un format tan habitual.

2. Framework d'emulació: QEMU és un emulador de codi obert que actualment suporta l'arquitectura RISC-V amb la versió 1.0 de l'extensió vectorial [9]; tot i que les seves mètriques temporals poden ser descartades, ha servit per val·lidar la funcionalitat i l'us de les instruccions vectorials d'aquesta arquitectura fins que es trobi un nucli d'aquestes característiques al mercat. En concret s'ha utilitzat l'arquitectura RISC-V de 64 bits amb les extensions d'instruccions GCV.
3. Compilador: per a totes les arquitectures s'ha utilitzat el compilador g++. La diferència de les comanda per compilar el codi principal entre versions o arquitectures és mínima, variant en la incorporació del flag d'operacions atòmiques per a RISC-V. Tots els resultats de les execucions d'inferència s'han obtingut indicant l'opció d'optimització -O3 per a que el compilador tracti d'optimitzar el codi principal al màxim, sense arriscar en afectar a la funcionalitat fent les aproximacions que suposa el nivel -Ofast. D'altra banda, amb l'objectiu d'evitar contemplar el soroll de l'execució, es compila i executa el mateix codi múltiples vegades, s'escull el temps menor resultant suposant que les variacions són degudes a aquest factor.

3.6 Codi de validació

Per verificar el funcionament de la llibrería de *TF Lite* per C s'utilitza un codi proveït pels creadors d'aquest framework que crida a les funcions bàsiques necessàries per executar una inferència. Primer es crea un model a partir d'un fitxer .tflite i l'intèrpret d'aquest model. A continuació, es copia el model i una variable amb un valor concret en un bufer on seran executats per l'intèrpret; el resultat d'aquesta predicció es guarda en una variable sempre amb el valor 0 (si la llibrería funciona correctament). Les funcions de *TF* tenen un mecanisme concret per la detecció i local·lització d'errors: al cridar funcions es va actualitzant l'estat esperat en una variable anomenada *kTfLiteOk,* si el resultat de la nostra crida a una funció de *TF Lite* no és el mateix que el d'aquesta variable la funció no s'ha executat correctament per algún motiu. Aquesta funcionalitat ens permet tractar els errors de forma individual i localitzat on s'han produït de forma inequívoca.

3.6 Codi principal

Aquest codi es basa en el codi de val·lidació provist per *TF*, afegint funcional·litats per executar les inferències amb el dataset escollit i fer llegible la informació resultant de la interpretació. Primer es defineixen la funció per borrar el model i l'intèrpret en cas de detectar un error, cridant a les funcions Delete de TF. També es defineix la funció que llegeix línies d'un fitxer de texte on tenim guardades les categoríes que utilitza el model per classificar, és a dir, es llegeixen les mil categoríes (labels) d'ImageNet.

Tot seguit inicia el codi d'execució principal. Per tal d'agilitzar les proves amb diferents imatges i models, al principi del main es declaren tots els paràmetres modificables que seràn utilitzats per l'algorisme, com el path dels arxius o el nombre de categoríes. Seguidament, es crida a la funció per llegir categoríes del fitxer i comença el bucle principal d'execució del programa, que llegeix una imatge per iteració i realitza la inferència. Aquesta implementació utilitza punters de la llibrería dirent per obtenir el nom de cada arxiu .jpeg dins el directori indicat, evitant instanciar explícitament o renombrar el nom de cada imatge amb el que predir la categoría; el sacrifici d'aquesta implementació recau en l'augment del cost computacional. Un cop s'ha obtingut l'arxiu, es llegeix amb la llibrería *ujpeg* i es disposa en format llegible per l'intèrpret indicant el nombre de canals de color (3 en el cas d'RGB). En aquest punt es segueix el mateix procediment descrit per l'apartat anterior, obtenint el resultat de la predicció. Finalment s'extreu la probabilitat d'encert del resultat indicant la categoría predita i la imatge utilitzada, calculant la mitjana del conjunt d'inferències i s'allibera la memòria dinàmica.

4 ESTAT DE L'ART

4.1 Les xarxes neuronals

A l'any 1958, Rosenblatt proposa el model del perceptró: aquest model es basa en un nombre d'entrades, uns pesos indicant la seva importància i l'ús d'una funció d'activació per calcular la sortida. Posteriorment, es va utilitzar per a crear xarxes on la sortida d'un o més perceptrons actuen com entrada del següent, simulant capes de neurones, és a dir, una xarxa neuronal.

Les xarxes neuronals en el camp de la computació són un paradigma de computació, inspirat en la xarxa cerebral de neurones, que permet que un sistema informàtic aprengui a prendre decisions d'acord amb les observacions i a l'experiència previa; per tal d'aconseguir aquest comportament d'aprenentatge s'utilitzen tècniques de Deep Learning i Machine learning, obtenint un sistema amb *Intel·ligència Artificial* (IA). Aquests sistemes poden d'executar tasques que requereixin d'intel·ligència humana, com per exemple la capacitat de percebre, reconèixer l'entorn i actuar en conseqüència o la capacitat d'aprendre constantment i adaptar-se als canvis.

Avui dia, trobem la IA en la major part dels àmbits de l'ésser humà i en les seves interaccions amb els sistemes informàtics; alguns exemples poden ser: el reconeixement de patrons o tendències comercials, la compressió d'imatges, eines de diagnòstic al sector sanitari i processament d'aliments, d'entre moltes de les seves aplicacions.

Els projectes i propostes tecnològiques més punteres s'exposen en congressos (com el Congrés Mundial d'Intel·ligència Computacional o el Congrés Mundial de

Neuro-Informàtica), papers i altres documents tècnics orientats al sector. Actualment, hi ha infinitat de models neurals, combinant diferents tipus de capes i funcions d'activació, amb l'objectiu de millorar la precisió i velocitat de resposta, així com la incorporació d'aquesta tecnología en nous sectors i l'automatització de diverses tasques.

4.2 Sector espacial

L'estudi i l'observació de l'espai sempre ha estat present dins dels àmbits d'interès de la humanitat: iniciant la seva observació a l'època prehistòrica deixant monuments megalítics com l'imponent Stonehenge (aprox 3000 aC.), passant pels egipcis i la construcció de les piràmides, i arribant als descobriments d'estudiosos grecs com *Tales de Mileto* (630 aC.), considerat el primer astrònom, que vàren establir les bases per posteriors recerques.

Durant l'època de la història moderna, l'evolució de la tecnología i la digitalització va suposar una nova instrumentació que permetía un rang d'observació més ampli i precís, superant grans barreres com la percepció de l'ésser humà, impulsant un creixement del sector. Posteriorment, la carrera especial del segle XX entre Estats Units i la Unió Soviètica va fomentar l'ús de noves tecnol·logíes per crear el primer satèl·lit artificial, la primera sonda especial orbitant en un altre planeta o la primera estació especial.

A l'actualitat es llençen centenars de satèl·lits anualment, innovant constantment amb l'objectiu de reduir el consum i l'impacte al medi ambient, allargant la vida útil dels dispositius, millorant la precissió o incorporant sistemes amb noves funcionalitats. S'ha de tenir en compte que els dispositius necessiten proteccions molt específiques i sistemes amb recuperació d'errors degut a la radiació si aquests es troben orbitant a l'espai [10].

4.3 Sistemes Encastats

Els sistemes encastats són dispositius computacionals dissenyats per realitzar tasques específiques i poden ser integrats en sistemes d'altres característiques.

Durant la dècada de 1960 van sorgir els primers circuits simples utilitzats per aplicacions específiques, en àmbits com l'aeroespacial o el militar. Iniciant un període de creixement exponencial, en la dècada dels 70 es van començar a crear els primers microprocessadors i el seu ús va generalitzar-se, cobrint les necessitats de cada sector: des de rellotges digitals o consoles de videojocs, fins a dispositius mèdics. Degut a l'avanç de la tecnol·logía dels semiconductors, la miniaturització dels components electrònics i les possibilitats que oferíen, es va extendre la integració dels sistemes encastats a la majoría dels aspectes de la nostra societat on no s'havien incorporat.

D'entre els diferents conjunts d'instruccions dels microprocessadors destaca l'arquitectura RISC-V, d'arquitectura oberta, amb el seu conjunt d'instruccions (*ISA*) dissenyat col·laborativament.

Actualment, el sector dels encastats es veu influenciat per les demandes i tendències de la indústria actual, innovant per crear sistemes més segurs, eficients energèticament, més potents, etc [11]. D'altra banda, continuament sorgeixen nous reptes degut a l'augment de la interconnexió entre dispositius i del volum de dades a l'actualitat.

4.4 Instruccions vectorials

Sempre s'ha tractat d'optimitzar temporalment un procés, i un bon exemple son les instruccions vectorials. Es tracta d'un conjunt de funcionalitats clau que permeten realitzar operacions de forma paral·lela, particularment útils en aplicacions de còmput intensiu de dades com el processament d'imatges; principalment, destaquen dues tecnologies amb característiques similars: Single Instruction-Multiple Data (SIMD) i processadors vectorials, com els RISC-V Vector Extension (RVV).

SIMD és una tecnologia que permet realitzar operacions en paral·lel amb diferents conjunts de dades utilitzant una única instrucció. Primer les dades són agrupades en paquets iguals i es guarden en registres vectorials, després s'aplica la mateixa operació a tots els elements del vector. Aquest enfocament és molt eficient en tasques repetitives com els càlculs matricials (Fig 3).

D'altra banda, l'extensió RVV proporciona funcionalitats vectorials al conjunt d'instruccions base. Aquestes instruccions permeten la personalització dels formats del vector, ampliant la seva capacitat d'aplicabilitat. El funcionament és similar a l'anterior implementació, permetent operacions aritmètiques, lògiques i de càrrega en paral·lel. Actualment l'extensió RVV es troba en la versió 1.0 pendent de revisió per a la seva publicació.

Les diferències principals entre les dues aproximacions, rauen en l'arquitectura i les possibilitats

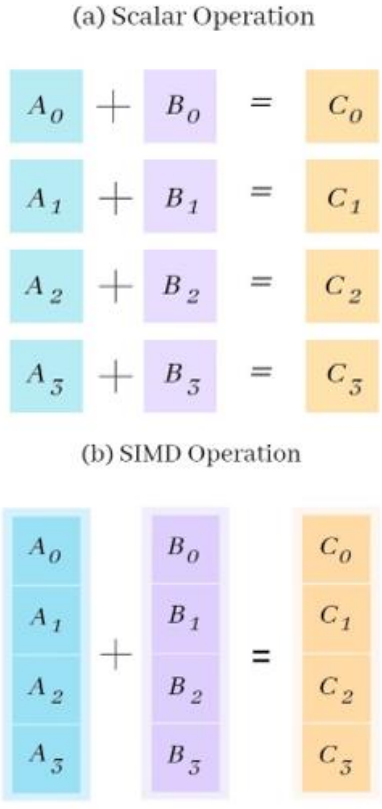


Fig. 3. Diferència entre operacions tradicionals executades independentment (a) i utilitzant agrupacions de dades amb operacions SIMD (b).

d'adaptació. SIMD és una tecnología més estàndard utilitzada en diverses arquitectures de processadors, mentre que RVV és únicament implementable amb l'arquitectura RISC-V tot i que ofereix un grau més de personalització i flexibilitat.

5 METODOLOGÍA

5.1 Planificació i procediment

S'ha escollit una metodología de complexitat incremental, amb una planificació inicial de tasques revisades periòdicament durant reunions setmanals del projecte Víbria; aquesta aproximació, sense arribar a seguir cap metodología àgil, ajuda a mantenir un seguiment constant i detectar la necessitat d'incorporar noves tasques molt ràpidament.

En primera instància, s'ha elaborat un anàlisi teòric sobre les xarxes neuronals més utilitzades, seleccionant les més adients per al camp d'interès del projecte Víbria de l'IEEC, és a dir, aquelles amb un pes mínim orientades al processament d'imatges amb un temps d'inferència acceptable. També s'han hagut d'analitzar els frameworks i entorns de desenvolupament on implementar els models de les xarxes neuronals escollides tenint en compte la compatibilitat en els microcontroladors a l'abast del projecte, en concret les arquitectures ARM i RISC-V.

Havent escollit l'entorn de desenvolupament descrit anteriorment, s'inicia amb la placa Raspberry1, d'implementació més simple que la VisionFive V1 degut al temps al mercat i comptar amb molt més suport; primer s'instal·la el framework *TensorFlow* 2.4 per realitzar les primeres proves de funcionament. Aquesta primera aproximació a les inferències és implementat en Python degut a ser el llenguatge més estés en l'àmbit de les neural networks. Per evaluar el correcte funcionament del codi d'inferència i la seva predicció, es comparen els resultats d'*accuracy* amb altres projectes utilitzant aquest toolchain en arquitectures similars.

A continuació degut als requeriments del projecte, es fa la primera exploració de l'*API* per llenguatge C de *TensorFlow*.

Les limitacions d'una placa tan antiga com la RPi1 (principalment la falta de RAM per instal·lar TF) es fan patents ràpidament, pel que el procés d'implementació passa a tenir la RPi3 com a tàrget, iniciant la instal·lació de l'aplicatiu de Python com a la SBC anterior i després per C. Cal destacar que al ser una llibrería que vol abarcar tantes arquitectures, acaba tenint molts problemes d'instal·lació a la majoría de SBC d'aquest projecte i és necessària la modificació de flags o crides al compilador del makefile original; aquests errors i la seva sol·lució ja han sigut reportats a l'IEEC per tal de facilitar la reutilització d'aquest projecte.

Per verificar que les modificacions de la instal·lació no han afectat la funcionalitat, es val·lida el model neural i les funcions del framework d'inferència amb el codi de prova que incorpora detecció d'errors i el punt on s'han produït, explicat prèviament; Finalment, s'implementa la inferència a la placa de forma nativa amb el conjunt de dos mil imatges d'inferència que l'algoritme principal anirà llegint. El procediment per la librería de C és repetit en la placa de RISC-V i a QEMU, incloent una comparativa de rendiment amb les implementacions prèvies per tal d'extreure conclusions, el procediment amb Python és descartat degut als resultats obtinguts. Per l'anàlisi d'instruccions vectorials es guarda el resultat d'objdump en un fitxer fe texte.

5.2 Dificultats per l'acompliment

Els aspectes continguts en aquest apartat han suportat un canvi dràstic en el desenvol·lupament del projecte o un replantejament de les tasques planificades.

El primer plantejament per la compilació del toolchain va ser la compilació creuada (*cross-compiling*) de les llibreríes per a les diferents arquitectures objectiu. Aquesta és una aproximació molt més habitual al sector dels sistemes encastats per tal d'evitar problemes de recursos a l'instalar grans frameworks com *TF Lite*; tot i així, va ser impossible aconseguir una llibrería funcional amb compilació creuada de *TF Lite* per C, degut, en concret, a dos fitxers (d'entre els centenars creats) que es basen en l'arquitectura host i no en l'especificada als paràmetres.

En qualsevol cas, tan per a la compilació creuada com per a la compilació nativa, el makefile i altres fitxers de *TF* han de ser modificats per a la seva implementació en ARM i RISC-V.

6 RESULTATS

6.1 Inferència de test en Python

Com s'ha descrit en l'apartat de metodología, les primeres proves van ser en llengüatge Python degut a la facilitat d'instal·lació de la llibrería de *TF Lite* amb l'instal·lador pip. Al ser un llengüatge interpretat, el temps d'execució és molt més elevat que en llengüatges compilats com C; en els resultats obtinguts anal·litzant una inferència individualment es pot observar la mateixa funcional·litat en els dos dispositius. De les dades obtingudes destaca una millora significativa en el temps d'execució del 191.52% en la tercera versió de Raspberry respecte la primera, probablement degut a la diferència de freqüència del processador (Table 2).

Cal destacar que en aquest cas concret la versió dos de

TABLE 2
INFERÈNCIA DE TEST EN PYTHON

MobileNet		Raspberry Pi 1 B+	Raspberry Pi 3 B+
V1	accuracy test	49.2%	49.2%
	categoria	Briard	Briard
	temps	17.739 seg	9.262 seg
V2	accuracy test	65.1%	65.1%
	categoría	Briard	Briard
	temps	15.547 seg	7.839 seg

Accuracy test = probabilitat d'encert obtingut utilitzant la mateixa imatge, categoría = categoría predita amb l'accuracy anterior, temps = el temps d'execució.

TABLE 3
INFERÈNCIA DE TEST EN C

MobileNet		Raspberry Pi 3 B+	VisionFive V1
V1	accuracy test	49.2%	49.2%
	categoria	Briard	Briard
	temps	0.437 seg	2.972 seg
V2	accuracy test	65.1%	45.9%
	categoria	Briard	Briard
	temps	0.404 seg	2.307 seg

Accuracy test = probabilitat d'encert obtingut utilitzant la mateixa imatge, categoría = categoría predita amb l'accuracy anterior, temps = el temps d'execució.

TABLE 4
INFERÈNCIES DEL DATASET EN C

MobileNet		Raspberry Pi 3 B+	VisionFive V1	Qemu
V1	accuracy total	46.8%	46.8%	46.8%
	temps total	15.015 min	95.322 min	-
	temps unitari	0.450 seg	2.859 seg	-
V2	accuracy total	41.6%	39.3%	39.3%
	temps total	11.141 min	62.404 min	-
	temps unitari	0.334 seg	1.872 seg	-

Accuracy total = mittjana de les probabilitats del conjunt d'inferències, temps total = el temps d'execució total en minuts, temps unitari=mitjana de temps per inferència de les dos mil imatges a partir del temps total.

MobileNet obté una categorització més precisa amb aquesta imatge; tot i així, és un cas excepcional com veurem posteriorment amb els resultats generals. Aquesta versió aconsegueix una reducció en el temps d'execució amb un speed-up de 1.14x respecte la primera per la RPi 1, molt similar al 1.18x de la RPi3.

6.2 Inferència de test en C

El primer gran resultat d'aquest apartat i d'aquest projecte és la compilació de la llibrería estàtica *de TensorFlow Lite* per llengüatge C executant correctament el codi de validació, constituint una seccions que més temps de recerca en fòrums ha suposat.

L'execució d'inferències amb la imatge de test complementa el codi de validació de la llibrería i verifica el mateix funcionament obtingut anteriorment amb l'intèrpret de Python. La principal diferència a destacar és la reducció en el temps d'execució del codi principal en la SBC RPi3, aconseguint una temps de predicció de 0.473 segons i un speed-up de 21.19x a la primera versió de *MobileNet* de la versió de C respecte la de Python, i 0.404 segons amb una millora de 19.40x a la segona. Per al cas de la placa VisionFive V1 s'observa un temps d'execució 6.8 vegades més lent que la placa de l'arquitectura ARM pel primer model i 5.7 per al segon. Tot i no ser prioritari per al projecte de l'IEEC, els resultats temporals a les dues SBC són molt més propers als que s'espera d'un sistema encastat real de classificació d'imatges; obtenint en la RPi3 una taxa d'execució del codi en C de 2.29 *frames per second* contra els 0.11 *fps* de l'intèrpret de Python pel primer model, i 2.47 *fps* del llengüatge compilat contra els 0.13 *fps* de l'interpretat. Si analitzem la diferencia entre models en aquest cas, segueix destacant una reducció en el temps d'inferència de la segona versió respecte la primera amb una petita millora des *fps* processats de 1,08x en la placa d'ARM contra la millora de 1.29x a l'arquitectura RISC-V (Table 3). També cal destacar la diferència en la predicció d'aquesta imatge de test del model V2 a les dues arquitectures, aquesta variació en la precisió és molt accentuada per aquesta imatge concreta, la diferència real entre les dues arquitectures s'analitzarà a continuació amb tot el dataset.

6.3 Inferències del dataset en C

En aquest punt, tenint l'execució del codi principal en C funcionant correctament pels diferents sets d'instruccions, es realitza el conjunt d'inferències sobre les dos mil imatges del dataset escollit, tant en les SBC de l'apartat anterior com en QEMU emulant l'arquitectura de la placa VisionFive amb l'inclusió de l'extensió vectorial RVV.

Si bé el temps de l'emulador no és significatiu a l'hora d'implementar el sistema físicament, analitzant aquests darrers resultats podem verificar que la classificació produeix exactament els mateixos resultats en les arquitectures RISC-V utilitzant totes les imatges, independentment de l'ús d'instruccions vectorials.

El resultats del primer model amb el dataset indiquen un temps d'inferència per imatge molt proper a l'apartat anterior; en canvi, el temps unitari del segon model es redueix aconseguint un speed-up sobre el temps d'execució total de 1.21x per la RPi 3 i de 1.23x per la placa experimental VF (Table 4). Aquesta millora de rendiment és possible degut a l'alt nivell de paral·lelització de la capa convolucional d'una dimensió que afegeix *MobileNet* V2, aprofitat pel compilador amb el flag d'optimització -O3. D'altra banda el segon model tot i ser més ràpid, aconseguint 2.99 *fps* en la RPi i 0.53 en la VF, té petites variacions en les prediccions a les dues arquitectures principals. Destaca un augment dels temps d'execució del programa per ambdós models, de mitjana, 5.975 vegades més lent a l'arquitectura RISC-V.

6.4 Ús d'instruccions vectorials de RISC-V

Per tal d'analitzar si s'han utilitzat les instruccions de l'extensió RVV en les execucions d'arquitectures RISC-V de l'apartat anterior, s'extreu la traducció en llengüatge ensamblador. Aquest codi assemblador està compost per més de 25 milions de línies d'instruccions per a l'arquitectura RV64GC de la VisionFive i 6 milions de línies en l'emulador amb RV64GCV; per comparar l'equivalència d'instruccions, es busquen les instruccions vectorials a l'assembler de l'emulador i s'identifica la funció on se'n fa ús. Entre les funcions del codi que utilitzen instruccions vectorials podem trobar la creació i decodificació d'imatges de la llibrería *ujpeg*, constructors i

(a) Assembler VisionFive

```
0000000000018448 <TfLiteDelegateCreate>:
   18448: 23 34 05 02   sd  zero, 40(a0)
   1844c: 23 30 05 02   sd  zero, 32(a0)
   18450: 23 3c 05 00   sd  zero, 24(a0)
   18454: 23 38 05 00   sd  zero, 16(a0)
   18458: 23 34 05 00   sd  zero, 8(a0)
   1845c: 23 30 05 00   sd  zero, 0(a0)
   18460: 82 80         ret
```

(b) Assembler Qemu

```
000000000001845a <TfLiteDelegateCreate>:
   1845a: 23 34 05 02   sd  zero, 40(a0)
   1845e: 23 30 05 02   sd  zero, 32(a0)
   18462: 57 70 82 cd   vsetivli  zero, 4, e64, m1, ta, ma
   18466: 57 34 00 5e   vmv.v.i v8, 0
   1846a: 27 74 05 02   vse64.v v8, (a0)
   1846e: 82 80         ret
```

Fig. 4. Diferència entre l'execució amb extensions tradicionals (a) i utilitzant l'extensió RVV (b).

destructors de *TensorFlow Lite*, i el propi main. Fet que la majoría d'aquestes funcions tenen centenars o milers d'instruccions (desenes de milers en el cas del main), s'ha escollit com a mostra la funció de menor extensió.

En aquesta exemple s'observa una funció amb les instruccions store *sd* de l'extensió integer RV64I que guarden el valor de *zero* en una posició de memòria amb un offset (Fig 4).

En el cas d'execució tradicional, aquesta funcionalitat s'implementa repetint la mateixa instrucció per a tots els elements; en canvi, utilitzant les instruccions vectorials s'aprofita aquesta repetició per optimitzar l'execució: primer es crea un vector amb *vsetivli* al registre v8 i s'utilitza la intrucció vectorial move *vmv* per omplir el vector amb els zeros, finalment el resultat es guarda a memòria amb la instrucció d'store vectorial *vse.*

7 Conclusions

L'API de TensorFlow Lite pel llengüatge de programació C malauradament no és fàcil d'implementar a les arquitectures ARM o RISC-V en comparació amb el framework per Python; no obstant, a l'aconseguir la llibrería funcional pel llengüatge compilat el temps d'execució de les inferències es redueix significativament. Els mecanismes de detecció d'errors i el codi de validació de TF Lite faciliten molt la seva localització i tractament. D'altra banda és possible utilitzar el mateix codi d'inferències per diferents models ajustant la imatge d'entrada a l'esperada per l'intèrpret, permetent un gran rang de resol·lucions.
La implementació d'aquest classificador d'imatges d'un miler de categoríes minimitza l'espai ocupat (tant a memòria com a disc) i els temps de gestió de la imatge utilitzant funcions paral·lelitzables de la llibrería ujpeg; també evita el renombrat del fitxers llegint qualsevol imatge en format .jpeg del directori, obtenint una classificació de 3 frames per second en la RPi3 i 0.53 fps en la placa de SiFive.

Actualment, l'arquitectura RISC-V està obtenint un creixement exponencial en quant a investigació i comunitat; amb la sortida del SBC VisionFive V2 al mercat, els benchmarks disponibles actualment d'aquesta segona versió contra la Raspberry Pi 3 model B+ indiquen un rendiment superior de la placa d'ARM. Aquestes proves ja pronosticaven els resultats d'una comparativa entre de la placa experimental predecessora, la VisionFive V1 disponible a l'IEEC, i la RPi3 B+, amb un temps d'execució 6 vegades més ràpid en ARM.

Com a conclusió final del present Treball de Final de Grau, s'ha verificat l'ús d'instruccions vectorials de l'extensió RVV en l'arquirectura emulada RV64GCV; tot i no obtenir uns resultats temporals realistes en QEMU, els resultats funcionals s'han aconseguit assolir, cobrint els objectius i necessitats del projecte satisfactòriament. S'espera que el temps d'execució es redueixi amb l'ús de les instruccions vectorials quan es tingui accés a un SBC funcional amb aquesta arquitectura.

Aquest projecte podría ser continuat en les següents línies de desenvolupament, cobrint alguns dels casos d'ús del nucli Víbria de l'IEEC utilitzant models neurals: detecció i classificació de núvols o incendis, compressió de dades o detecció i prevenció de col·lisions d'embarcacions.

Agraïments

Primer agraïr a la meva família i amics, al meu tutor de projecte Màrius per acceptar-me a l'IEEC i a l'incansable suport del company de projecte Eugeni; a l'ICE-IEEC i els seus membres per fer-me sentir com a casa. Finalment, als professors de la d'Enginyería de la UAB, gràcies als quals estic aquí.

Bibliografia

[1] RISC-V Foundation. "RISC-V Vector Specification". Disponible en: https://github.com/riscv/riscv-v-spec 2021. [Accedit: 11 de juny del 2023].

[2] Wang, W., Li, Y., Zou, T., Wang, X., You, J., & Luo, Y. (2020). A novel image classification approach via dense-MobileNet models. *Mobile Information Systems*, *2020*.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. "Imagenet large scale visual recognition challenge". International Journal of Computer Vision, 115(3):211–252, 2015.

[4] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". *arXiv preprint arXiv:1704.04861*. (Unpublished).

[5] F. Chollet. "Xception: Deep learning with depthwise separable convolutions." *arXiv preprint arXiv:1610.02357* (2016).

[6] Google Inc. "TensorFlow Hub". Disponible en: https://tfhub.dev/s?subtype=module,placeholder

[7] Google Inc. "TensorFlow Library". [En línea]. Disponible en: https://github.com/tensorflow/tensorflow 2021. [Accedit: 13 de juny del 2023].

[8] A. Nayak. "Cats vs dogs-2000 images". Disponible en: https://www.kaggle.com/datasets/abhinavnayak/catsvdogs-transformed 2021. [Accedit: 23 d'abril del 2023].

[9] P. Adelt, B. Koppelmann, W. Müller, and C. Scheytt, "QEMU Support for RISC-V: Current State and Future Releases," *2nd International Workshop on RISC-V Research Activities*, vol. (Presentation), 2019.

[10] N.J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. Abella."De-RISC: the First RISC-V Space-

Grade Platform for Safety-Critical Systems," 2021 IEEE Space Computing Conference (SCC), Laurel, MD, USA, 2021. (pp. 17-26). doi: 10.1109/SCC49971.2021.00010.

[11] Hamdioui, S., Gaydadjiev, G., & Van de Goor, A. J. "The state-of-art and future trends in testing embedded memories. In *Records of the 2004 International Workshop on Memory Technology, Design and Testing, 2004.* (pp. 54-59). IEEE. 2019.

[12] Ju, R. Y., Lin, T. Y., Jian, J. H., & Chiang, J. S. (2023). "Efficient convolutional neural networks on Raspberry Pi for image classification." *Journal of Real-Time Image Processing*, (pp. 20(2)- 21).

[13] Curtin, B. H., & Matthews, S. J. (2019, October). "Deep learning for inexpensive image classification of wildlife on the Raspberry Pi." In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (pp. 0082-0087). IEEE.

[14] François Chollet. 2016. "Xception: Deep learning with depthwise separable convolutions." arXiv preprint arXiv:1610.02357 (2016).

[15] Wang, W., Li, Y., Zou, T., Wang, X., You, J., & Luo, Y. (2020). A novel image classification approach via dense-MobileNet models. *Mobile Information Systems*, *2020*.

[16] Chiu, Y. C., Tsai, C. Y., Ruan, M. D., Shen, G. Y., & Lee, T. T. (2020, August). Mobilenet-SSDv2: An improved object detection model for embedded systems. In *2020 International conference on system science and engineering (ICSSE)* (pp. 1-5). IEEE.

APÈNDIX

A1. ARQUITECTURES CISC I RISC

L'arquitectura d'un processador és un punt vital en el disseny dels sistemes informàtics. Les diferents arquitectures incorporen els seus propis sets d'instruccions pel processament de dades i l'execució de processos. En aquest annex s'aprofundeix i es comparen les dues principals arquitectures de processadors: CISC (Complex Instruction Set Computer i RISC (Reduced Instruction Set Computer).

L'arquitectura CISC es va desenvolupar a mitjans del segle XX amb la característica principal de tenir un conjunt d'instruccions complexes, que es poden dividir en múltiples operacions; per exemple, els accessos a memòria es poden indicar en la mateixa instrucció junt amb altres operacions aritmètiques. Les instruccions CISC poden tenir longitud variable, és a dir, algunes instruccions poden requerir diferents cicles de rellotge per executar-se incloent diferents longituds i formats, dificultant la gestió per part de la Unitat de Control. D'altra banda aquesta arquitectura ofereix una varietat de mètodes d'adreçament a memòria flexibilitzant-ne l'accés. Aquesta flexibitat facilita la programació en llenguatges d'alt nivell i permet realitzar tasques complexes amb un menor nombre d'instruccions, optimitzant l'ús de memòria. Els desavantatges d'aquesta arquitectura sorgeixen de la complexitat de les instruccions, allargant el temps de desenvolupament del set d'instruccions; així mateix, degut a la intensitat de còmput del processador consumeix més energia i augmenta la seva temperatura.

D'altra banda, l'arquitectura RISC va sorgir per oferir una alternativa a CISC i es caracteritza per tenir l'enfocament contrari. El set d'instruccions està compost per un conjunt d'instruccions que executen operacions bàsiques de càlcul i transferència de dades, totes de longitud fixa i usualment d'operacions indivisibles; aquesta característica, sumada al nombre limitat de modes d'adreçament, simplifica la Unitat de Control i la gestió d'instruccions. En contraposició a CISC, les instruccions d'aquest conjunt estàn dissenyades per executar-se en un sol cicle de rellotge augmentant el rendiment i l'eficiència del processador i reduïnt el consum. També es fa un ús extensiu dels registres, els quals són accessibles únicament amb instruccions de càrrega i emmagatzematge, minimitzant els accessos a memòria. La seva cinquena versió, anomenada RISCV, es caracteritza per estar dividit en extensions segons el tipus d'operacions a executar, permetent una implementació personalitzada minimitzant l'espai ocupat a disc. Aquestes característiques disminueixen la complexitat i faciliten la seva implementació en microarquitectures, el consum d'energía i generació de calor es veu reduït. Els desavantatges d'aquesta arquitectura s'adverteixen en aquesta simplicitat, requerint moltes més instruccions per l'execució de tasques.

A l'actualitat, tot i que la implementació de sistemes informàtics amb processadors de l'arquitectura CISC està més estesa, aquesta elecció depèn de les necessitats de diseny específiques de cada sistema. L'arquitectura CISC ofereix un conjunt d'instuccions complex que pot ser més útil el tasques específiques, mentre que l'arquitectura RISC prioritza sempre una execució eficient i ràpida. De fet, l'empresa Apple ha anat canviant espontàniament l'arquitectura dels processadors que incorporava als seus dispositius.

Cal destacar que amb l'avanç de la tecnologia i l'enfocament de codi obert pel que aboga RISC, la diferència entre aquestes dues arquitectures s'ha anat reduint i algunes funcionalitatss, com les instruccions SIMD, són implementables en ambdues aproximacions.

En l'àmbit dels sistemes encastats, l'aproximació de l'arquitectura RISC cobreix millor les necessitats d'eficiència, baix consum i simplicitat que requereix aquest sector; per exemple, l'arquitectura ARM (Addvanced RISC Machine) integrada en multitud de microcontroladors i microprocessadors actuals, com indica el seu nom, es basa en l'arquitectura RISC.

A2. INSTAL·LACIÓ TENSORFLOW LITE

Totes les implementacions d'aqest framework s'han realitzat en la família de sistemes operatius GNU/Linux.

La instal·lació de l'intèrpret per Python és molt simple i està molt ben optimitzada, obtenint una llibreria funcional en uns deu minuts. Requereix l'ús de la versió Python 3.7 o superior, i la instal·lació de la llibreria Atlas (paquet libatlas-base-dev). A continuació, simplement s'utilitza l'administrador de paquets *pip* per instal·lar *TensorFlow*. Els errors del framework per aquest llengüatge són mínims; només s'ha observat un error generat per la versió dels buffers de protocol, molt fàcil de solventar fent un downgrade de protobuf a la versió 3.20.x o menor (solució indicada pel compilador).

En canvi, la instal·lació de l'*API* de *TF Lite* per C ha dificultat l'avanç del projecte degut a errors en el *Makefile* del repositori of icial i la falta d'informació al fòrum de *TensorFlow*, tant tractant de cross-compilar la llibreria estàtica com compilant de forma nativa. En aquest cas, es parteix del repositori i guia d'instal·lació oficial de *TF*, descarregant la versió 2.4.0 i les dependències necessàries; es recomana tenir una versió dels compiladors gcc i g++ igual o superior a 10 per evitar més errors. A continuació, s'ha d'incloure la llibreria *limits* en el fitxer block_map.cc ubicat en el directori tensorflow-2.4.0/tensorflow/lite/tools/make/downloads/ruy/ruy. També serà necessari instal·lar el paquet libz-dev. En les SBC d'arquitectura ARM hem de modificar el fitxer rpi_makefile.inc i incloure els flags *-marm* als paràmetres de CFLAGS i CXXFLAGS i *-latomic* a LDFLAGS. Per la placa de l'arquitectura RV64 només cal incloure el darrer flag al fitxer riscv_makefile.inc. Finalment, generem la llibreria estàtica tensorflow-lite.a i tots els fitxers de l'*API* per C a partir del Makefile.

A3. PLANIFICACIÓ D'ENTREGUES RELLEVANTS

Tasca	Darrera data d'entrega	Data màxima d'entrega
Reunió inicial	14/02/2023	19/02/2023
Informe inicial	12/03/2023	12/03/2023
Informe de progrés I	23/04/2023	23/04/2023
Informe de progrés II	28/05/2023	28/05/2023
Informe Final Provisional	17/06/2023	18/06/2023
Presentació Provisional	30/06/2023	30/06/2023
Informe Final Definitiu	02/07/2023	02/07/2023
Dossier del TFG	02/07/2023	02/07/2023
Pòster	06/07/2023	06/07/2023
Presentació Final	09/07/2023	09/07/2023
Defensa Tribunal	10/07/2023	10/07/2023