UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Progetto Programmazione di Sistemi Embedded: Mobile AI

Gruppo:

Alessandro Girlanda

Andrea Mutti

Umberto Bianchin

ANNO ACCADEMICO 2023-2024

Indice

Elenco delle figure

Capitolo 1

Neural Newtorks

1.1 Storia

Capitolo 2

NNAPI

2.1 Accelerazione Hardware

La caratteristica principale dell'uso di NNAPI[3] è l'accelerazione hardware. In particolare, uno smartphone è sicuramente dotato di una unità di elaborazione centrale (CPU da ora in avanti), che esegue tutte le principali operazioni, sfruttando i registri, la cache, la ram ed eventuali storage. Grazie alla CPU si possono svolgere tutte le operazioni di I/O, gestione memoria, grafica, gestione batteria, errori e quant'altro. La CPU da sola, però, nella maggior parte dei casi non riesce a reggere quantità elevate di calcolo, in quanto si trova già occupata a svolgere operazioni base date dall'OS (Android nel nostro caso, ma anche iOS per esempio) e varie operazioni di I/O e memory. Proprio per questa sua limitazione, nella grande maggioranza degli smartphone moderni si trovano installati nella Motherboard anche altri componenti, in particolare una Scheda Video (GPU) ed eventualmente altri moduli quali Digital Signal Processor (DSP) e Neural Processing Unit (NPU). Una scheda video, come suggerisce il nome, fornisce la potenza di calcolo necessaria a renderizzare ogni aspetto legato alla GUI dell'utente, ma non solo. Grazie alla sua elevata potenza infatti, può essere ampiamente sfruttata per tutte le operazioni che richiedono una grande quantità di dati da elaborare e lo stesso può essere detto per DSP.

Discorso a parte per le NPU: si tratta di un processore "complementare" adatto ad eseguire operazioni di calcolo prettamente incentrate sulle reti neurali. In ambienti mobile la prima comparsa è stata nel 2017, quindi relativamente recente, grazie alla presenza di una NPU sul chip kirin 970, presente nel Huawei Mate 10 pro. La sola integrazione di questo componente ha portato grandi vantaggi in termini di prestazioni e di efficienza energetica (si parla di un 50% di efficienza energetica guadagnata rispetto alla gamma precedente), rendendolo di fatto importante per tutti i prodotti successivi. Grazie a questa novità da parte di Huawei, anche i vari competitors iniziarono a produrre, ricercare e sperimentare il più possibile per stare al passo con il mercato, rendendo di fatto le NPU, e il mondo dell'AI e delle reti neurali in generale, una realtà quotidiana presente su quasi ogni piccolo schermo.

Tornando quindi all'accelerazione hardware, il punto cardine è la possibilità di sfruttare tutti gli altri componenti oltre alla CPU per eseguire complesse operazioni di calcolo e facilitare l'elaborazione di dati. In tema reti neurali, si ottiene un guadagno sotto più punti di vista per quanto riguarda le operazioni di inferenza. Sicuramente si ottiene un incremento nella velocità di risposta, in quanto il carico di lavoro viene efficientemente distribuito tra le varie unità di calcolo disponibili, e una ridotta latenza generale, in quanto non c'è necessità di contattare un server esterno per richiedere dati. Grazie a ciò, non è nemmeno necessaria una connessione esterna, quale wifi o LTE, in quanto è possibile avere i dati salvati in locale, rendendoli di fatto sempre disponibili in qualsiasi situazione, a patto che ci sia batteria residua. La questione energetica è un argomento piuttosto delicato, perché unità di elaborazione come GPU e DSP sono molto dispendiose in termini di batteria e spesso si deve ricorrere a dei compromessi. Altro punto a sfavore, soprattutto per smartphone meno performanti, è la questione memoria: l'APK specifico potrebbe avere un peso di svariati GB e in esecuzione la RAM potrebbe facilmente risultare piena, rischiando così di portare il sistema in stati di rallentamento o crash. Un punto di forza riguarda sicuramente la privacy, in quanto i dati rimangono all'interno del dispositivo e non sono salvati su un cloud.

2.2 API Android Neural Networks

L'API Android Neural Networks è una API pensata per eseguire operazioni pesanti, in termini di calcolo computazionale, per il machine learning su dispositivi Android. Ha una compatibilità SW estremamente elevata, in quanto può essere utilizzata su ogni dispositivo con Android 8.1 (livello API 27) o successivo, coprendo quindi più del 90% dei dispositivi android attualmente in uso.

La NNAPI può essere sfruttata programmando in linguaggio C/C++ e usando librerie di sistema di livello superiore come "Runtime NNAPI". Runtime NNAPI è una libreria condivisa tra un'app e i driver back-end con la particolarità di essere aggiornabile, per ricevere aggiornamenti al di fuori del normale ciclo di rilascio di Android. Gli sviluppatori, quindi, possono correggere più facilmente bug presenti nel runtime e migliorarne le compatibilità con i driver.

La vera potenza di queste API però, risiede nella possibilità di essere sfruttate da framework superiori come TensorFlow Lite, PyTorch Mobile o Caffe2 che creano e addestrano reti neurali. Grazie a NNAPI è dunque possibile sfruttare modelli già definiti e allenati, avendo quindi un'abbondanza di dati a disposizione.

In figura 2.1 si riesce a visualizzare facilmente il flusso di lavoro. L'applicazione utilizza le librerie e i framework di machine learning (come PyTorch o Runtime NNAPI) che a loro volta sfruttano le API di basso livello di NNAPI. Questa comunica con l'astrazione HW del dispositivo per rete neurale, contattando i driver che si interfacciano all'hardware specifico (GPU, DSP, ecc…). Questo meccanismo efficace permette di dividere il lavoro in vari livelli di astrazione, in modo da isolare i compiti e ridurre

le problematiche, aumentando la rapidità di risoluzione dei vari bug presenti. Questo meccanismo di divisione dei compiti è alla base di due importanti design pattern, in particolare “Responsibility”, che permette di dividere le responsabilità ad ogni layer, e “Layered Architecture”, che, come suggerisce il nome, è un pattern architetturale che ha come concetto di base la suddivisione del software in vari strati. La tipologia sopra descritta è alla base della progettazione Android e Java.

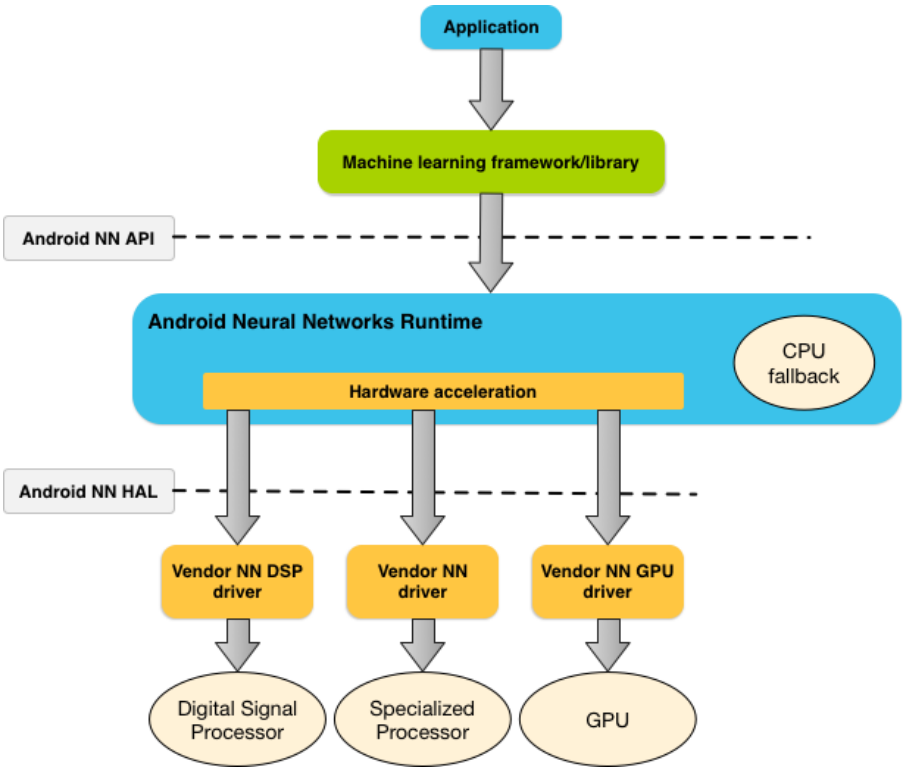

Figura 2.1: Architettura di sistema per l’API Android Neural Networks.

Per sfruttare i calcoli tramite NNAPI è necessario prima creare un grafo diretto che definisca i suddetti calcoli. Questo grafo infatti, combinato con i vari dati di input, forma il modello per la valutazione del runtime NNAPI. Vengono definite 4 astrazioni principali con NNAPI:

- **Modello**: si tratta di un grafo di calcolo delle operazioni matematiche e di tutti i valori costanti appresi durante il processo di addestramento, che sono operazioni specifiche delle reti neurali. Includono convoluzione bidimensionale[1] (2D), attivazione logistica (sigmoid[2]), attivazione lineare rettificata[3] (ReLU) e altro ancora. Una volta creato correttamente, può essere utilizzato

[1] https://en.wikipedia.org/wiki/Convolution

[2] https://en.wikipedia.org/wiki/Sigmoid_function

[3] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

nuovamente in tutti i vari thread e le compilation. In NNAPI, un modello è rappresentato come un'istanza ANeuralNetworksModel[4]. La creazione di un modello è un'operazione sincrona.

- **Compilation**: rappresenta una configurazione per compilare un modello NNAPI in codice di livello inferiore. La creazione di una compilazione è un'operazione sincrona. Una volta creata correttamente, può essere riutilizzata in più thread ed esecuzioni. In NNAPI, ogni compilazione è rappresentata come un'istanza ANeuralNetworksCompilation[5].

- **Memoria**: rappresenta la memoria condivisa, i file mappati di memoria e buffer di memoria simili. L'utilizzo di un buffer di memoria consente al runtime NNAPI di trasferire i dati ai driver in modo più efficiente. In genere, un'app crea un buffer di memoria condiviso contenente tutti i tensori necessari per definire un modello. È possibile utilizzare anche i buffer di memoria per archiviare gli input e gli output per un'istanza di esecuzione. In NNAPI, ogni buffer di memoria è rappresentato come un'istanza ANeuralNetworksMemory[6].

- **Esecuzione**: interfaccia per l'applicazione di un modello NNAPI a un insieme di input e per la raccolta dei risultati. L'esecuzione può essere eseguita in modo sincrono o asincrono. Per l'esecuzione asincrona, più thread possono attendere la stessa esecuzione. Al termine di questa esecuzione, tutti i thread vengono rilasciati. In NNAPI, ogni esecuzione è rappresentata come un'istanza ANeuralNetworksExecution[7].

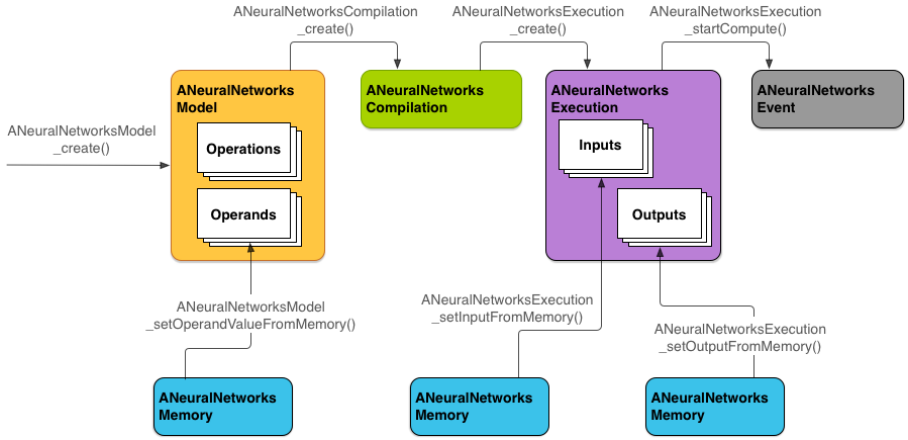

Figura 2.2: Flusso di programmazione per l'API Android Neural Networks.

[4]https://developer.android.com/ndk/reference/group/neural-networks?hl=it#aneuralnetworksmodel

[5]https://developer.android.com/ndk/reference/group/neural-networks?hl=it#aneuralnetworkscompilation

[6]https://developer.android.com/ndk/reference/group/neural-networks?hl=it#aneuralnetworksmemory

[7]https://developer.android.com/ndk/reference/group/neural-networks?hl=it#aneuralnetworksexecution

2.3 Modello

L'unità di calcolo fondamentale in NNAPI è il modello, il quale è definito da uno o più operandi e operazioni. Gli operandi sono data object utilizzati nella definizione del grafo, tra i quali costanti, input, output e nodi intermedi; possono essere di due tipi: scalari e tensori. Uno scalare è un singolo valore in uno dei vari formati booleano, intero o virgola mobile a 16/32 bit. I tensori, invece, sono array n-dimensionali. In NNAPI i tensori possono anche essere quantizzati a 8 o 16 bit.

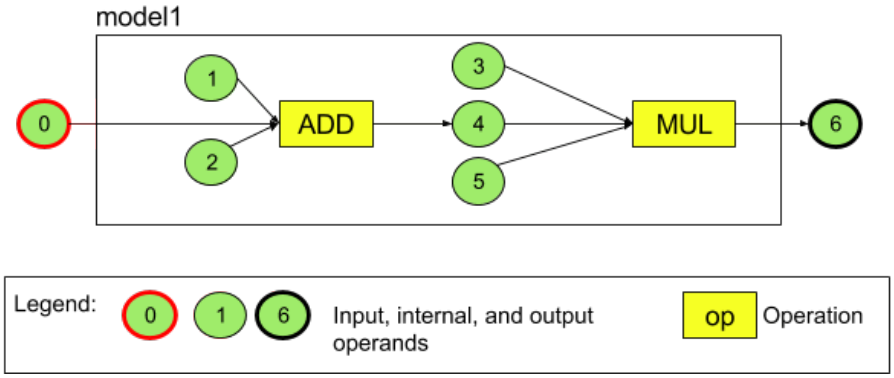

Figura 2.3: Esempio di operandi per un modello NNAPI

2.3.1 Fasi operative

In NNAPI, un'operazione specifica i calcoli da eseguire ed è costituita dai seguenti elementi:

- il tipo di operazione (ad es. addizione, moltiplicazione, convoluzione);
- l'elenco di indici degli operandi utilizzati dall'operazione per l'input;
- l'elenco di indici degli operandi utilizzati dall'operazione per l'output.

Prima di aggiungere l'operazione, bisogna aggiungere al modello gli operandi utilizzati o prodotti da questa operazione.

L'utilizzo delle Neural Networks API avviene a basso livello, grazie a linguaggi come C/C++. In particolare C++ rispecchia la programmazione ad oggetti (come lo sono Java e Kotlin) e permette quindi di definire veri e propri modelli come oggi in maniera intuitiva. Per creare un modello, per esempio, basta definire due righe di codice:

```
ANeuralNetworksModel model = NULL;
ANeuralNetworksModel_create(&model);
```

Dopo la creazione è necessario aggiungere i vari operandi attraverso `ANeuralNetworks_addOperand()` e sfruttando le strutture dati fornite, come *ANeuralNetworksOperandType*.

2.4 Compilation

In questa fase viene determinato su quali processori verrà eseguito il modello e viene chiesto ai driver corrispondenti di preparasi per l'esecuzione di questo, eventualmente generando anche codice macchina specifico per i processori scelti. Da sottolineare la possibilità di scegliere esplicitamente quali dispositivi utilizzare e il tradeoff tra l'utilizzo della batteria e la velocità di esecuzione.

2.4.1 Esecuzione Sincrona VS Asincrona

Alcune operazioni nel runtime NNAPI sono eseguite in maniera sincrona. Ciò significa che ogni operazione di questo tipo può rallentare l'esecuzione di un'altra operazione in attesa del risultato, in quanto è necessario attendere la fine dell'esecuzione per continuare l'esecuzione; in questo modo viene ridotta la potenza innata del multithreading.

Di contro, sfruttare esecuzioni asincrone non sempre è conveniente, in quanto il dispositivo deve poter generare e sincronizzare tutti i vari thread, con latenza variabile. A volte è possibile avere ritardi di più di 500 microsecondi tra quando viene inviata la notifica di un thread al momento in cui viene associato un core.

Da Android 10 in poi, NNAPI supporta le *burst executions* tramite l'uso di un oggetto apposito. Queste esecuzioni altro non sono che sequenze di esecuzioni della stessa compilazione che si verificano in rapida successione, come la campionatura audio o l'acquisizione video. Gli oggetti burst quindi possono portare ad esecuzioni più rapide, dove gli acceleratori rimangono in uno stato di prestazioni elevate per tutta la durata del burst.

2.4.2 Gestione errori

Di norma, se si verifica un errore durante il partizionamento o un errore con i driver scelti, NNAPI ricorre alla implementazione della CPU per una o più operazioni ove possibile. Sfruttando framework come TensorFlow Lite, nella maggior parte dei casi, può essere vantaggioso disabilitare il ricorso all'utilizzo della CPU e gestire gli errori con implementazioni delle operazioni ottimizzate dal client.

Se il partizionamento o la compilazione non vanno a buon fine, l'intero modello sarà provato sulla CPU, stessa cosa vale per un'operazione che continua a fallire. Questo ovviamente riduce notevolmente la velocità di esecuzione ma aumenta le possibilità di successo, riducendo eventuali errori hardware. Da considerare, però, che alcune operazioni non sono nemmeno supportate dalla CPU. È

possibile quindi disabilitare il fallback della CPU e rimuoverla dall'elenco dei processori nel caso in cui non la si voglia utilizzare per le operazioni del modello.

2.4.3 Memoria

Da sottolineare la possibilità, da Android 11 e successive, di sfruttare domini di memoria che forniscono interfacce di allocazione per memorie opache (sono un tipo di memoria che è gestita in modo che i dettagli interni non siano direttamente accessibili o modificabili dall'utente o dalle applicazioni). Ciò permette alle applicazioni di passare dati e ricordi nativi del dispositivo tra le varie esecuzioni, in modo che NNAPI non copi o trasformi i dati inutilmente quando vengono effettuate esecuzioni consecutive sullo stesso driver. Queste funzionalità sono destinate principalmente ai tensori interni ai driver con pochi accessi lato client, come i tensori di stato. Per tensori con frequenti accessi alla CPU lato client, è necessario utilizzare pool di memoria condivisi.

2.5 Tempo di esecuzione

Il test delle prestazioni di un'app può essere effettuato misurando il tempo di esecuzione o tramite profilazione. Per misurare il tempo di esecuzione, si può utilizzare l'API di esecuzione sincrona o API a livello inferiore come *ANeuralNetworksExecution_setMeasureTiming* e *ANeuralNetworksExecution_getDuration*. Queste API consentono di misurare il tempo di esecuzione su un acceleratore o nel driver, escludendo l'overhead dovuto al runtime NNAPI e all'IPC.

Le informazioni di tempistica possono essere utili per ottimizzare le prestazioni dell'applicazione in produzione, anche se la raccolta di tali informazioni potrebbe influenzare le prestazioni stesse (cosa da notare in fase di controllo e revisione dei dati). È importante notare che solo il driver è in grado di calcolare il tempo trascorso su se stesso o sull'acceleratore.

Per profilare l'applicazione, si può utilizzare *Android Systrace*, che genera automaticamente eventi `systrace` per profilare diverse fasi del ciclo di vita del modello e i vari livelli dell'applicazione, come Application, Runtime, IPC e Driver. Per generare i dati di profilazione NNAPI, è possibile utilizzare il comando systrace di Android e l'utility `parse_systrace`. Questi strumenti consentono di analizzare gli eventi systrace generati dall'applicazione e di ottenere una visualizzazione tabellare dei tempi spesi nelle diverse fasi del ciclo di vita del modello e nei diversi livelli dell'applicazione. Inoltre, Android 11 introduce miglioramenti nella qualità del servizio (QoS) per NNAPI, consentendo alle applicazioni di indicare le priorità relative dei propri modelli e di impostare timeout per la compilazione e l'inferenza del modello.

Infine, sono disponibili argomenti avanzati sull'utilizzo degli operandi NNAPI, come tensori quantizzati, operatori facoltativi e tensori di ranking sconosciuto. Il benchmark NNAPI è disponibile su AOSP per valutare latenza e accuratezza dei modelli.

2.5.1 Log

NNAPI offre informazioni diagnostiche utili nei log di sistema, che possono essere analizzati utilizzando l'utilità logcat. È possibile attivare il logging dettagliato per fasi o componenti specifici impostando la proprietà *debug.nn.vlog* tramite adb shell. Questo consente di registrare informazioni dettagliate su fasi come la creazione di modelli, la compilazione, l'esecuzione e altro ancora. Una volta attivato, il logging dettagliato genererà voci di log con un tag impostato sul nome della fase o del componente. Per controllare il livello dei messaggi di log mostrati da logcat si utilizza la variabile di ambiente *ANDROID_LOG_TAGS* (è solo un filtro applicato a logcat, si deve comunque impostare la proprietà debug.nn.vlog su all per generare informazioni di log dettagliate).

Capitolo 3

PyTorch Mobile

3.1 PyTorch & PyTorch Mobile

PyTorch[5] è un framework di deep learning open source, sviluppato inizialmente da *Meta AI* e ora parte della *Linux Foundation*. Progettato con Python, viene usato per creare reti neurali e per progetti di apprendimento automatico, combinando la libreria di machine learning **Torch**[17] con un'API di alto livello basata su Python. Torch è famosa, specialmente nel campo del Deep Learning, per fornire tool semplici e flessibili insieme a performance elevate; uno dei sui punti salienti è il grande supporto per le GPU, che contribuisce ad un allenamento più efficiente dei modelli di deep learning. PyTorch fornisce innanzitutto un pacchetto Python per funzionalità ad alto livello, come l'elaborazione dei **tensori**[1] ed inoltre una conversione in un formato così detto **TorchScript**, che permette di convertire modelli PyTorch in modo da poter essere eseguiti in un ambiente non-Python, rendendoli adatti per l'uso su piattaforme con risorse limitate, come i dispositivi mobili.

PyTorch Mobile[9], introdotto per la prima volta nel 2019 alla *PyTorch Developer Conference*, si riferisce ad un set di librerie e funzionalità fornite da PyTorch che permettono allo sviluppatore di eseguire un modello PyTorch direttamente su dispositivi mobili, come smartphone e tablet.

3.2 Caratteristiche Principali

Le caratteristiche principali di PyTorch Mobile, così come scritto sul sito ufficiale[12], sono:

- Disponibile per iOS, Android e Linux;

[1] Array multidimensionale utilizzato per memorizzare dati. Nel campo del Machine Learning vengono usati per rappresentare e manipolare input, pesi e output.

- Fornisce API per comuni compiti di pre-processing e integrazione necessari ad incorporare Machine Learning nelle applicazioni mobile;
- Supporta la libreria XNNPACK per le CPU Arm e integra QNNPACK per i kernel a 8 bit quantizzati;
- Fornisce un efficiente interprete mobile per Android e iOS;
- Supporterà a breve backend hardware come GPU, DSP e NPU.

XNNPACK[4] è una libreria altamente ottimizzata per accelerare le operazioni di reti neurali convoluzionali (CNN) e altre operazioni di reti neurali su hardware mobile, mentre QNNPACK[7] (Quantized Neural Network PACKage) è progettata per accelerare le reti neurali quantizzate[2] su hardware mobile.

3.3 Da un modello PyTorch a PyTorch Mobile

Il tipico flusso dalla creazione del modello in PyTorch all'implementazione sul dispositivo mobile può essere visionato in figura 3.1; di seguito verranno spiegati i vari step. Il primo step è ovviamente quello di scrivere un modello PyTorch o utilizzarne uno preesistente e convertirlo in un modello per dispositivi mobile; vedremo solamente come compiere questa azione, visto che è ciò che viene richiesto dal progetto.

3.3.1 Quantizzazione

La quantizzazione[14] è una tecnica utilizzata principalmente per accelerare la fase di inferenza nei modelli di machine learning, rendendo più veloce l'elaborazione delle previsioni o delle decisioni basate sui dati. Questo metodo funziona riducendo la precisione dei numeri utilizzati; questo serve per avere una rappresentazione del modello più compatta e per poter utilizzare operazioni vettoriali più efficientemente su molti hardware. Partendo da un modello FP32 (Floating Point 32 bit), PyTorch supporta la quantizzazione INT8 (Integer 8 bit), ottenendo così una riduzione della dimensione del modello e della necessità di memoria di 4 volte. Inoltre il supporto hardware per la computazione INT8 è solitamente 2 o addirittura 4 volte più veloce rispetto a quella FP32. Queste tecniche si utilizzano

[2]La quantizzazione è un processo che riduce la precisione dei numeri usati nei calcoli di una rete neurale, da 32-bit a 8-bit o meno, riducendo così l'uso della memoria e migliorando le prestazioni senza sacrificare significativamente l'accuratezza.

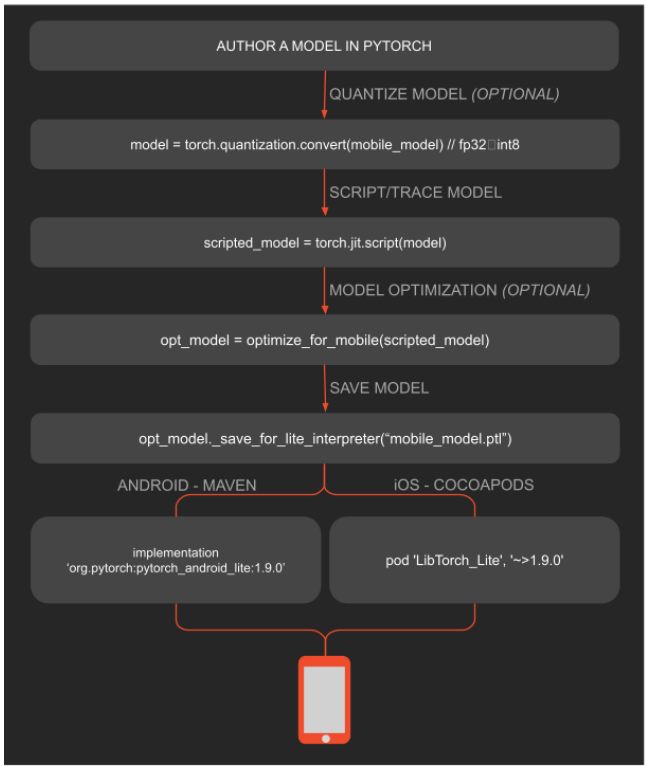

Figura 3.1: Workflow dal training al rilascio di un modello su piattaforma mobile.

principalmente nella fase di inferenza del modello (”only the forward pass is supported for quantized operators”), e non durante la fase di addestramento.

PyTorch fornisce tre differenti modalità per la quantizzazione:

- Eager Mode Quantization (beta)
- FX Graph Mode Quantization (prototipo)
- PyTorch 2 Export Quantization

Eager Mode Quantization

Questa funzionalità, ancora in versione beta, richiede che l’utente gestisca manualmente le fasi di quantizzazione e dequantizzazione. Supporta solamente i moduli[3] e non le funzioni[4].

[3] In PyTorch, un modulo (classe torch.nn.Module) rappresenta un componente di un modello che incapsula uno o più strati, parametri e una funzione di forward che definisce come l’input viene trasformato in output. Possono avere uno stato, ossia dei pesi aggiornati durante il training,

[4] Spesso presenti nel modulo torch.nn.functional, sono operazioni stateless che eseguono calcoli specifici come attivazioni (ReLU, sigmoid), operazioni di pooling, e altre trasformazioni matematiche. Non mantengono uno stato.

FX Graph Mode Quantization

Al contrario del primo, questo modo di effettuare la quantizzazione è automatizzato. Migliora la Eager Mode aggiungendo il supporto per le funzioni, anche se potrebbe essere necessario effettuare un refactor del modello per renderlo compatibile con la modalità FX.

PyTorch 2 Export Quantization

Questa è la nuova modalità di quantizzazione completa del grafo, e può essere utilizzata da una percentuale più elevata di modelli rispetto alla modalità grafica FX, anche se presenta ancora limitazioni riguardo alcuni costrutti Python e richiede l'intervento dell'utente per supportare il dinamismo nel modello esportato. Le caratteristiche principali sono:

1. API programmabili per configurare come un modello viene quantizzato.
2. UX (User Experience) semplificata per gli utenti e per gli sviluppatori backend, poiché è necessario interagire con un singolo oggetto, chiamato *Quantizer*.
3. Rappresentazione (opzionale) del modello quantizzato di riferimento che può rappresentare calcoli quantizzati con operazioni intere più vicine agli attuali calcoli quantizzati che avvengono nell'hardware.

Ci sono poi tre tipi di quantizzazione supportati, è inoltre possibile vedere quali operatori sono compatibili con i tipi di quantizzazione in figura 3.2:

- Quantizzazione dinamica: pesi quantizzati con *activations*[5] lette/salvate in floating point e quantizzate per i calcoli;
- Quantizzazione statica: pesi e activations quantizzati, è necessaria una fase di calibrazione per determinare i migliori parametri di quantizzazione dopo l'addestramento;
- Quantizzazione statica *aware training*: pesi e activations quantizzati, i parametri sono modellati durante l'allenamento.

[5]Le attivazioni, nel contesto del machine learning, si riferiscono ai valori di output prodotti dai neuroni di una rete neurale durante il processo di forward pass, ovvero quando l'input viene elaborato attraverso i vari strati della rete fino a generare un output.

	Static Quantization	Dynamic Quantization
nn.Linear	Y	Y
nn.Conv1d/2d/3d	Y	N
nn.LSTM	Y (through custom modules)	Y
nn.GRU	N	Y
nn.RNNCell	N	Y
nn.GRUCell	N	Y
nn.LSTMCell	N	Y
nn.EmbeddingBag	Y (activations are in fp32)	Y
nn.Embedding	Y	Y
nn.MultiheadAttention	Y (through custom modules)	Not supported
Activations	Broadly supported	Un-changed, computations stay in fp32

Figura 3.2: Operatori compatibili con i due diversi tipi di quantizzazione

3.3.2 Scripting e Tracing del modello

Questi due step[1] servono per convertire un *nn.Module* in un grafo in formato TorchScript.

- **Tracing** usa il comando `torch.jit.trace()`, in cui si passano come argomenti il modello e un input d'esempio. L'input verrà processato dal modello e le operazioni eseguite verranno appunto tracciate e registrate in un grafo.
- **Scripting** usa il comando `torch.jit.script()` che prende in input il solo il modello, in questo caso il modello verrà ispezionato staticamente e da questa analisi verrà generato il codice TorchScript.

Viene usato prevalentemente il metodo Scripting, poiché cattura sia le operazioni che tutta la logica del modello, inoltre se l'esportazione dovesse fallire sarà quasi sicuramente per una ragione ben definita (di conseguenza la modifica da apportare sarà chiara). Il Tracing viene preferito quando non si ha accesso al codice e quindi non è possibile apportare modifiche. È possibile anche utilizzarli insieme.

3.3.3 Ottimizzazione

Grazie alla funzionalità `torch.utils.mobile_optimizer.optimize_for_mobile`[18] si semplifica il processo di ottimizzazione di modelli per garantire che funzionino in modo efficiente su piattaforme con risorse limitate, come gli smartphone e i tablet. Il

comando `torch.utils.mobile_optimizer.optimize_for_mobile(script_module, optimization_blocklist=None, preserved_methods=None, backend='CPU')` esegue diverse operazioni in base ai parametri passati:

1. *script_module*: un'istanza del modello TorchScript;
2. *optimization_blocklist*: ottimizzazioni da escludere tra quelle disponibili;
3. *preserved_methods*: lista di metodi che devono essere mantenuti quando si invoca `freeze_module`;
4. *backend*: tipo di dispositivo usato per eseguire il modello risultante (CPU di default, oppure "Vulkan" o "Metal").

In caso non si passi una lista di ottimizzazioni da non eseguire, vengono eseguite tutte le seguenti:

- Conv2D + BatchNorm fusion: combina operazioni Conv2d[6] e BatchNorm2d[7] in un'unica operazione Conv2d, aggiornando i relativi pesi e bias.
- Insert and Fold prepacked ops: sostituisce le operazioni Conv2D e le operazioni lineari con le loro controparti preconfezionate, ottimizzando l'accesso alla memoria e l'esecuzione del kernel.
- ReLU/Hardtanh fusion: integra operazioni di ReLU[8] o Hardtanh[9] con le operazioni Conv2D o lineari precedenti, sfruttando l'ottimizzazione hardware XNNPACK.
- Dropout removal: elimina i nodi di dropout[10] dal modello quando il training è impostato su false.
- Conv packed params hoisting:sposta i parametri impacchettati delle convoluzioni al modulo radice, riducendo la dimensione del modello senza influire sui risultati numerici.
- Add/ReLU fusion: trova e combina operazioni di addizione seguite da ReLU in un'unica operazione `add_relu`.

[6] Applica una convoluzione 2D su un segnale di ingresso composto da diversi piani di ingresso.

[7] Applica la Batch Normalization ad un input 4D.

[8] Applica la funzione rectified linear unit elemento per elemento

[9] Applica la funzione HardTanh function elemento per elemento

[10] Durante l'addestramento, azzera casualmente alcuni elementi del tensore di input con probabilità p

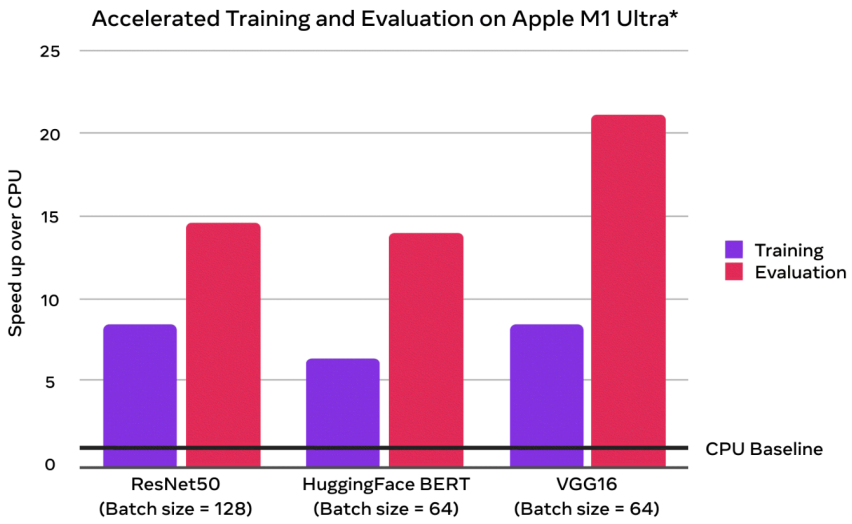

Figura 3.3: Accelerated GPU training and evaluation speedups over CPU-only.

3.4 PyTorch Backend

Come già accennato nella sezione 3.3.3, uno dei punti dell'ottimizzazione di un modello consiste nella scelta del backend. Questa scelta, a pare per la CPU, è basata sul tipo di dispositivo utilizzato: Vulkan per Android e Metal per iOS e macOS.

Vulkan[13] è un'API di basso livello che consente un controllo diretto e efficiente dell'hardware GPU (pensato principalmente per Android ma utilizzabile anche su Linux e Mac), facilitando una gestione delle risorse più efficace e un miglior parallelismo, ottenendo così un netto miglioramento delle prestazioni durante le operazioni di inferenza, soprattutto con modelli graficamente complessi.

Dall'altro lato, Metal[6] offre un'integrazione ottimale per sfruttare al meglio le specificità hardware degli apparecchi Apple che montano processori Apple Silicon. Anche questo può significativamente accelerare le operazioni di inferenza grazie alla sua gestione avanzata della memoria e alle capacità di elaborazione parallela, com'è possibile vedere in figura 3.3

Capitolo 4

TensorFlow

4.1 Introduzione

TensorFlow[15] è una libreria open source per l'apprendimento automatico, il calcolo numerico e altre attività di analisi statistica e predittiva. Questo tipo di tecnologia, sviluppata e rilasciata da Google nel novembre 2015, rende l'implementazione di modelli di machine learning più semplice e veloce per gli sviluppatori, assistendoli nel processo di acquisizione dei dati, nella formulazione di previsioni su larga scala e nel successivo affinamento dei risultati. Lo scopo principale di TensorFlow è la creazione e l'addestramento di reti neurali, che possono essere utilizzate per moltissime applicazioni[16], quali:

- Classificazione delle immagini;
- Elaborazione del linguaggio naturale;
- Analisi delle serie temporali;
- Riconoscimento vocale;
- Sviluppo di soluzioni di visione artificiale;
- Ottimizzazione di chatbot;
- Sistemi di assistenza clienti automatizzati;
- Altri ancora.

La versatilità e il grande range di applicazioni rendono TensorFlow uno strumento veramente potente e, per questo, è il **motore di AI più utilizzato**, come si può osservare in figura 4.1.

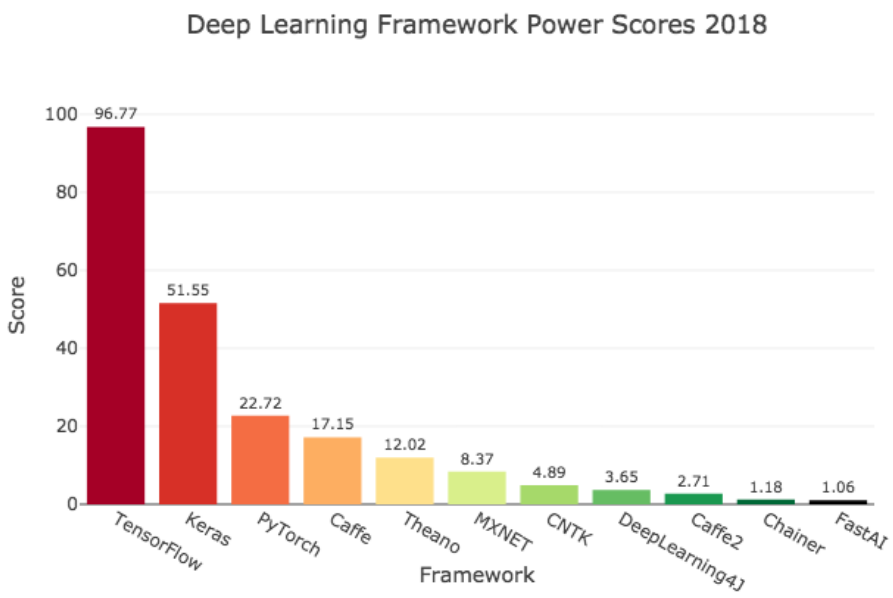

Figura 4.1: Diagramma dei punteggi di utilizzo di vari framework nel 2018

4.2 TensorFlow Lite

TensorFlow si può dire essere il "genitore" di TensorFlow Lite[2], introdotto da Google nel 2017, che non è altro che una versione ottimizzata per l'uso su dispositivi embedded e mobili. TensorFlow Lite è un framework di deep learning open-source, che converte un modello TensorFlow pre-addestrato in un formato specifico, che può essere ottimizzato per la velocità e l'archiviazione. Le sue caratteristiche principali sono:

- Supporto per più piattaforme, coprendo dispositivi Android e iOS, Linux embedded e microcontrollori;
- Supporto per diversi linguaggi di programmazione, come Java, Swift, Objective-C, C++ e Python;
- Alte prestazioni, facendo ricorso all'accelerazione hardware e all'ottimizzazione del modello;
- Ottimizzato per l'apprendimento automatico sul dispositivo, affrontando 5 vincoli chiave:
  - Latenza: TensorFlow Lite riesce ad eliminare il ritardo tra l'invio dei dati al server e il ricevimento della risposta. Infatti, il framework non necessita di inviare dati ad un server esterno (visto che l'apprendimento e l'elaborazione dei dati avvengono direttamente sul dispositivo) e, quindi, il tempo di predizione viene ridotto notevolmente;

- Privacy: è garantita la riservatezza massima nell'elaborazione di dati sensibili o personali che, infatti, non vengono mai condivisi con server remoti. Questo è possibile perché i dati non lasciano mai il dispositivo dell'utente, assicurando la massima privacy;
- Connettività: TensorFlow lite non necessita di una connessione internet permettendo lo svolgimento di attività di apprendimento e inferenza anche in situazioni in cui non è possibile o pratico avere una connessione stabile;
- Dimensioni del modello: i dispositivi mobili o embedded dispongono, generalmente, di risorse minori rispetto ai computer. La riduzione dello spazio di archiviazione e della necessità di risorse computazionali per un modello TensorFlow diventa, dunque, essenziale per l'esecuzione di algoritmi di Machine Learning sul dispositivo;
- Consumo energetico: l'elaborazione dei dati e l'inferenza di un modello di Machine Learning possono incidere pesantemente sulla durata della batteria di un dispositivo mobile. TensorFlow Lite, infatti, si occupa di gestire efficientemente il consumo di energia. Inoltre, l'eliminazione della necessità di trasmettere dati riduce ulteriormente il consumo energetico.

4.3 Workflow di sviluppo

TensorFlow Lite, insieme a TensorFlow, offre tutti gli strumenti per la generazione e l'utilizzo di un modello di Machine Learning. Il workflow di sviluppo è il seguente:

- Generazione di un modello TensorFlow Lite: a questo scopo, si può decidere se utilizzare un modello di TensorFlow Lite esistente, creare un modello TensorFlow Lite da zero o convertire un modello TensorFlow in un modello TensorFlow Lite. In questo report, si analizzerà l'ultima delle 3 opzioni poiché coerente con il progetto;
- Inferenza: il processo di esecuzione di un modello TensorFlow Lite sul dispositivo per effettuare previsioni basate sui dati di input prende nome di inferenza;
- Analisi e miglioramento delle prestazioni: tramite dei delegati, si utilizza l'accelerazione hardware in modo che il modello incontri requisiti di efficienza elevati.

4.3.1 Generazione di un modello TensorFlow Lite

TensorFlow Lite rappresenta i modelli in uno speciale formato portatile efficiente noto come **FlatBuffers** (identificato dall'estensione del file .tflite). Questa scelta offre numerosi vantaggi rispetto al formato del modello di buffer del protocollo di TensorFlow come: dimensioni ridotte (codice meno

ingombrante) e inferenza più rapida (accesso diretto ai dati senza un ulteriore passaggio di analisi/decompressione) che consente a TensorFlow Lite di funzionare in modo efficiente su dispositivi mobili, che hanno disponibilità di calcolo e memoria limitate.

Un modello TensorFlow Lite, inoltre, può contenere i cosiddetti "metadati" che forniscono una descrizione del modello leggibile dall'uomo e dei dati interpretabili dalla macchina per la creazione automatica di pipeline di pre e post-elaborazione durante l'inferenza sul dispositivo. Come è stato anticipato, i principali metodi di generazione di un modello TensorFlow Lite sono 3:

1. Riciclare un modello pre-esistente TensorFlow Lite con o senza metadati;

2. Creare un modello TensorFlow Lite da zero tramite l'ausilio della libreria TensorFlow Lite Model Maker che semplifica il processo di training di un modello TensorFlow Lite utilizzando un set di dati personalizzato;

3. Conversione di modello TensorFlow in un modello TensorFlow Lite utilizzando il convertitore TensorFlow Lite.

Ci si concentrerà su quest'ultima opzione, in quanto è interessante studiare i meccanismi con cui viene trasformato un modello TensorFlow (adatto per dispositivi di vario genere e potenza) in un modello TensorFlow Lite (usato per dispositivi mobili e, quindi, molto più limitati in termini di memoria e risorse di calcolo).

La **conversione dei modelli** TensorFlow nel formato TensorFlow Lite prevede percorsi diversi a seconda del contenuto del modello di Machine Learning. Come primo passaggio di tale processo, infatti, è necessario valutare il modello per determinare se può essere convertito direttamente. Questa valutazione determina se il contenuto del modello è supportato dagli ambienti di runtime TensorFlow Lite standard in base alle operazioni TensorFlow che utilizza. Se il modello utilizza operazioni non presenti nel set supportato, vi è la possibilità di eseguire il refactoring del modello o utilizzare delle tecniche di conversione avanzate.

La **valutazione della conversione** è un passaggio importante prima di provare a convertire il modello. Durante la valutazione, infatti, si determina se il contenuto del modello è compatibile con il formato TensorFlow Lite in termini di dimensione dei dati utilizzati, requisiti di elaborazione hardware e dimensioni e complessità del modello. La maggior parte dei modelli può essere convertita direttamente nel formato TensorFlow Lite, ma alcuni modelli potrebbero necessitare di un refactoring o di una conversione avanzata per renderli compatibili. La conversione vera e propria avviene tramite il **convertitore TensorFlow Lite** che prende un modello TensorFlow e genera un modello TensorFlow Lite in formato FlatBuffer. Il convertitore funziona con i seguenti formati del modello di input: SavedModel (modello TensorFlow salvato su disco come un insieme di file), modello Keras (creato utilizzando l'API di alto livello Keras), formato Keras H5 (una variante del SavedModel supporta-

to dall'API Keras) e modelli creati tramite delle funzioni concrete (ossia tramite API TensorFlow di basso livello).

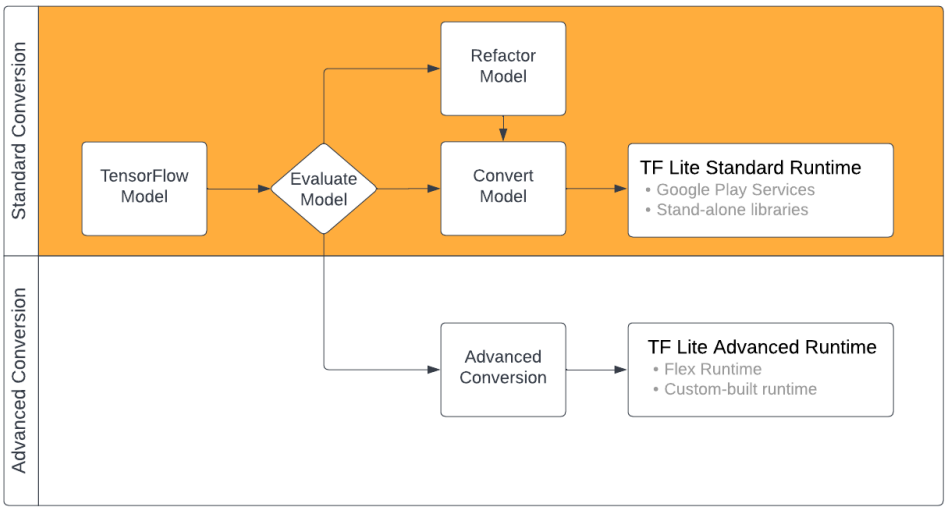

Figura 4.2: Workflow di conversione di TensorFlow Lite

Il modello può essere convertito utilizzando l'**API Python** o anche lo strumento della riga di comando. E' consigliato l'utilizzo dell'API Python perché consente di integrare la conversione nella pipeline di sviluppo, applicare ottimizzazioni, aggiungere metadati e altre ulteriori attività che semplificano il processo di conversione. La riga di comando, invece, supporta solo la conversione del modello di base. L'API Python usa, in particolare, la classe **TF.lite.TFLiteConverter** e i sui metodi `from_saved_model()`, `from_keras_model()` e `from_concrete_functions()` per convertire modelli rispettivamente di tipo SavedModel, Keras e da funzioni concrete.

Il convertitore accetta, inoltre, 3 opzioni (o flag) che personalizzano la conversione del modello:

- I flag di compatibilità che specificano se la conversione deve consentire operatori personalizzati, nel caso in cui nel modello TensorFlow vi siano delle operazioni non supportate da TensorFlow Lite;
- I flag di ottimizzazione che specificano il tipo di ottimizzazione da applicare durante la conversione. Tendenzialmente la quantizzazione è la tecnica maggiormente usata;
- I flag di metadati che permettono l'aggiunta di metadati al modello convertito.

Nel caso in cui vi siano problemi di compatibilità con gli operatori, si può proporre la **conversione avanzata**, che prevede il refactoring del modello e ulteriori opzioni alternative.

4.3.2 Ottimizzazione del modello

Per andare incontro alla limitatezza della memoria e della potenza di dispositivi mobili ed Edge, TensorFlow Lite fornisce delle **tecniche di ottimizzazione** per far rientrare i modelli in questi vincoli. I modi principali in cui l'ottimizzazione del modello aiuta lo sviluppo dell'applicazione sono:

- Riduzione delle dimensioni: i modelli più piccoli dispongono di dimensioni di archiviazione ridotte sul dispositivo mobile dell'utente, dimensioni di download inferiori in termini di tempo e larghezza di banda e meno utilizzo della memoria RAM durante l'esecuzione, garantendo prestazioni e stabilità migliori.

- Riduzione della latenza: la diminuzione della quantità di tempo necessaria per eseguire una singola inferenza con un determinato modello (latenza) è sintomo di buone prestazioni. La latenza, inoltre, può avere impatto sul consumo energetico ed è importante, dunque, tenere questa caratteristica in considerazione;

- Compatibilità con l'acceleratore: l'ottimizzazione di un modello permette, in alcuni casi, di utilizzare acceleratori hardware estremamente efficienti e veloci.

L'ottimizzazione può, però, comportare modifiche nell'accuratezza del modello, che devono essere tenute in considerazione durante il processo di sviluppo di un'applicazione. Queste variazioni di precisione dipendono molto dall'ottimizzazione del singolo modello e, per questo, non sono facili da prevedere. In genere, però, i modelli ottimizzati su dimensioni o latenza perdono sempre una quantità variabile di precisione (tendenzialmente piccola). Ci sono diversi tipi di ottimizzazione, ma quelli più usati sono:

- Quantizzazione: tecnica di ottimizzazione che funziona riducendo la precisione dei valori usati per rappresentare i parametri di un modello, che di default sono numeri in virgola mobile a 32 bit. In base ai requisiti dei dati, la richiesta di dimensione, la precisione e l'hardware supportato vi sono 4 tipi di quantizzazione: float16 post-training, gamma dinamica post-training, intera post-training e training consapevole della quantizzazione. Ogni tipologia è specifica per determinate casistiche ma, in generale, tutti e 4 i tipi di quantizzazione portano ad una riduzione della latenza e delle dimensioni a discapito della precisione del modello;

- Pruning: metodologia di ottimizzazione che funziona rimuovendo i parametri all'interno di un modello che hanno solo un impatto minimo sulle sue previsioni. In questo modo il modello avrà le stesse dimensioni e la stessa latenza di runtime ma potrà essere compresso in maniera più efficace e, quindi, sarà più facile ridurre le dimensioni di download;

- Clustering: strategia di ottimizzazione che prevede il raggruppamento dei pesi di ciascun livello di un modello in un numero predefinito di cluster e, per ogni gruppo, il calcolo del valore del

centroide. In questo modo, si riduce il numero dei pesi e quindi si diminuisce la complessità. I modelli a cui è stata applicata questa tecnica possono essere compressi più efficacemente.

4.3.3 Eseguire l'inferenza

Una volta ottenuto il modello addestrato è possibile testarlo con operazioni di inferenza, ossia il processo di generazione di stime del modello per nuovi non usati per la fase di training. Nel caso di TensorFlow Lite l'inferenza può essere eseguita in due modi diversi in base al tipo di modello:

- Nel caso di **modelli senza metadati** si può utilizzare l'API dell'interprete TensorFlow Lite;
- Nel caso di **modelli con metadati** è possibile far ricorso a API predefinite utilizzando le Task Library di TensorFlow Lite o costruire delle pipeline di inferenza personalizzate con le librerie di supporto di TensorFlow Lite.

Per eseguire un'inferenza in un modello TensorFlow Lite è necessario un **interprete**, il quale deve essere snello e veloce garantendo minima latenza di carico, di inizializzazione ed esecuzione. L'inferenza in TensorFlow Lite segue i seguenti passaggi:

1. Caricamento del modello che contiene il grafo di esecuzione;
2. Costruzione di un interprete e trasformazione del formato dei dati di input grezzi in un formato supportato dal modello;
3. Esecuzione dell'inferenza, utilizzando apposite API per l'allocazione dei tensori;
4. Interpretare l'output in un modo significativo che sia utile nell'applicazione.

Le API di inferenza di TensorFlow sono supportate da Android, iOS e Linux in più linguaggi di programmazione. L'esecuzione di un'inferenza (ma anche la costruzione di modelli) può far uso di specifiche librerie che aiutano lo sviluppatore a creare esperienze Machine Learning migliori. Queste librerie si suddividono in: **task libraries** e **support libraries**.

4.3.4 Task libraries di TensorFlow Lite

Le task libraries di TFLite forniscono interfacce ottimizzate per modelli per attività frequenti di Machine Learning, come classificazione di immagini, domande e risposte, ecc. Le interfacce sono progettate specificamente per ciascuna attività per ottenere le migliori prestazioni e usabilità. La libreria attività funziona su più piattaforme ed è supportata su Java, C++ e Swift. Quali sono le caratteristiche di una task library?

- API ben definite utilizzabili anche da non esperti di machine learning: È possibile eseguire l'inferenza in sole 5 righe di codice. Le API fornite sono potenti e facili da usare, e permettono il facile sviluppo di modelli di machine learning su dispositivi mobili;
- Elaborazione dei dati complessa ma efficace: supporta la logica di elaborazione del linguaggio naturale per la conversione dei dati nel formato richiesto dal modello. Questa logica è usabile anche nell'addestramento e nell'inferenza;
- Aumento della performance: l'elaborazione dei dati viene eseguita in pochi millisecondi, assicurando inferenze rapide e poco costose;
- Estendibilità e personalizzazione: è possibile sfruttare tutte i vantaggi forniti dalle task libraries e creare facilmente API personalizzate di inferenza Android/iOS.

Grazie a queste efficienti librerie, sono supportate diverse attività tra cui: API di visione (come un classificatore di immagini o un rilevatore di oggetti), di linguaggio naturale, di audio e personalizzate.

4.3.5 Support libraries di TensorFlow Lite

Gli sviluppatori di applicazioni per dispositivi mobili interagiscono con oggetti tipizzati come bitmap o con primitive come gli interi. TensorFlow Lite, però, usa tensori nella forma di ByteBuffer che sono difficili da debuggare e manipolare. Le support libraries Android di TensorFlow Lite nascono proprio da questa necessità di supportare l'elaborazione di input e di output di modelli TFLite, rendendo l'interprete più facile da usare. Le librerie di supporto TensorFlow Lite forniscono, per esempio, una serie di metodi di manipolazione delle immagini base (ritagli/ridimensionamenti) e di elaborazione di dati audio di base.

4.4 Miglioramento delle prestazioni: i delegati

TensorFlow Lite mette a disposizione diverse strategie per l'ottimizzazione e la massimizzazione delle prestazioni di un modello di machine learning, uno di questi è il delegato. Un delegato consente di eseguire i modelli (in parte o interamente) su un altro esecutore più efficiente specificatamente per il tipo di modello e la piattaforma su cui si esegue. I delegati abilitano l'accelerazione hardware dei modelli TensorFlow Lite sfruttando gli acceleratori sul dispositivo come la GPU e il processore di segnale digitale (DSP).

Come impostazione di default TensorFlow Lite utilizza kernel CPU per l'ottimizzazione del set di istruzioni ARM Neon. Tuttavia, la CPU non è necessariamente ottima per l'aritmetica complessa tipica dei modelli di machine learning. La maggior parte dei telefoni cellulari attuali contiene, però, chip che sono in grado di gestire meglio queste operazioni pesanti. Utilizzarli per le operazioni di

rete neurale offre enormi vantaggi in termini di latenza ed efficienza energetica. Per esempio, le GPU riescono a velocizzare fino a 5 volte la latenza, mentre il processore DSP è in grado di ridurre del 75% il consumo di energia. A ciascuno di questi acceleratori sono associate API che consentono la computazione personalizzata, come OpenCL o OpenGL ES per GPU e Hexagon SDK per DSP. Dunque, per il corretto funzionamento degli acceleratori è necessario scrivere parecchio codice. TensorFlow Lite risolve il problema fornendo delle API che fungono come ponte tra il runtime TFLite e queste API di basso livello.

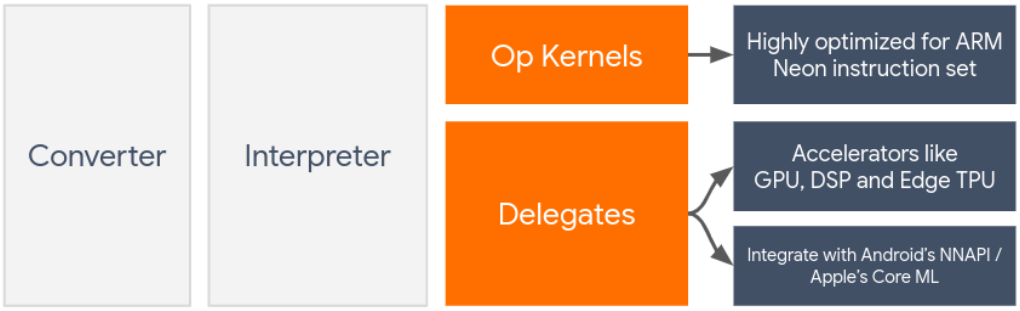

Figura 4.3: Distinzione tra kernel CPU e i delegati.

4.4.1 Scelta di un delegato

TensorFlow Lite supporta più tipi di delegati, ognuno dei quali è ottimizzato per determinate piattaforme e particolari modelli. Nello specifico, la scelta di un delegato si basa su due criteri principali: la piattaforma (Android o iOS) e il tipo di modello (a virgola mobile o quantizzato) da accelerare. Per quanto riguarda la piattaforma:

- Multipiattaforma (Android e iOS): GPU è l'unico tra i vari delegati che permette l'utilizzo sia su android che su iOS;
- Android: ci sono due delegati che supportano l'utilizzo su android (e NON su iOS), ossia il delegato NNAPI (disponibile in android 8.1 e versioni successive) e il delegato Hexagon (disponibile in versioni android precedenti che non supportano NNAPI);
- iOS: l'unico delegato ottimizzato per iOS è Core ML (disponibile su dispositivi mobili apple con SoC A12 o superiore).

Per quanto riguarda il tipo di modello:

Tipo di modello	GPU	NNAPI	Hexagon	CoreML
Virgola mobile (32 bit)	✓	✓	X	✓
Quantizzazione float16 post-training	✓	X	X	✓
Quantizzazione della gamma dinamica post-training	✓	✓	X	X
Quantizzazione intera post-training	✓	✓	✓	X
Training consapevole della quantizzazione	✓	✓	✓	X

Ogni acceleratore è progettato per una certa larghezza di bit dei dati. Per esempio, se viene fornito un modello in virgola mobile ad un delegato che supporta solo operazioni quantizzate a 8 bit (come il delegato Hexagon), allora il delegato non accetterà il modello il quale verrà eseguito interamente sulla CPU.

Scegliere la configurazione di accelerazione hardware ottimale per il dispositivo di ciascun utente può essere difficile. Inoltre, abilitare la configurazione errata su un dispositivo può causare elevata latenza, errori di runtime o problemi di precisione causati da incompatibilità hardware sfociando, quindi, in un servizio scadente e insoddisfacente per l'utente. In questo contesto, TensorFlow Lite mette a disposizione un **servizio di accelerazione per Android**: un'API che aiuta nella scelta della configurazione di accelerazione hardware ottimale per un determinato dispositivo utente e per uno specifico modello, riducendo al minimo il rischio di errori di runtime o problemi di precisione. Il servizio di accelerazione valuta diverse configurazioni di accelerazione sui dispositivi target eseguendo benchmark di inferenza con il modello TensorFlow Lite creato. I risultati dell'esecuzione dei benchmark possono essere salvati su cache e utilizzati durante l'inferenza. Inoltre, questi benchmark sono fuori processo, riducendo al minimo il rischio di arresti anomali dell'applicazione Android. Fornendo il modello, i campioni di dati e i risultati attesi il servizio di accelerazione eseguirà un benchmark di inferenza TFLite per fornire consigli hardware allo sviluppatore.

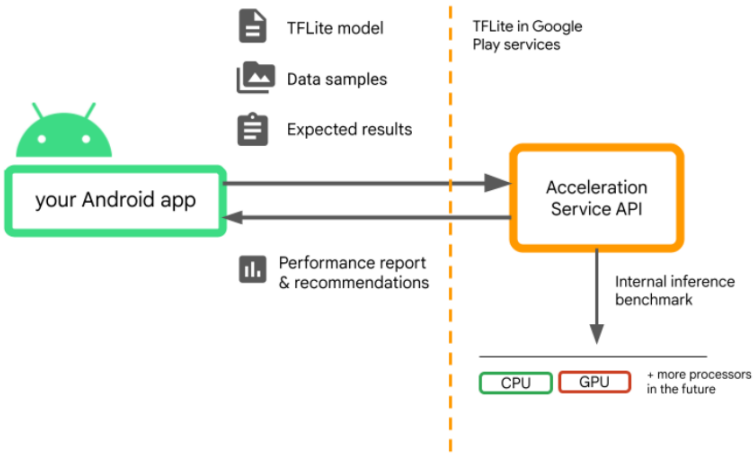

Figura 4.4: Schema riassuntivo del funzionamento del servizio di accelerazione

4.4.2 Tools per la valutazione

TensorFlow Lite fornisce strumenti di valutazione delle prestazioni e dell'accuratezza che consentono agli sviluppatori di analizzare e verificare l'efficienza dell'utilizzo dei delegati nell'applicazione creata. I due principali tools utilizzati sono:

- Tools per la latenza e l'utilizzo di memoria: stimano le prestazioni del modello considerando la latenza media di inferenza, l'overhead di inizializzazione, l'ingombro di memoria e altri;
- Tools per l'accuratezza e la correttezza: tendenzialmente, i delegati eseguono calcoli con una precisione differente rispetto ai kernel CPU. Di conseguenza, quando uso un delegato per l'accelerazione hardware la precisione tende a diminuire. Questo non succede sempre: per esempio la GPU, che esegue operazioni in virgola mobile, potrebbe migliorare leggermente la precisione. I tools che misurano questa metrica possono essere basati o meno sulla task specifica da valutare.

4.5 TensorFlow Lite per Android

TensorFlow Lite dispone di ambienti di esecuzione predefiniti e personalizzabili per l'esecuzione rapida e efficiente di modelli Android, incluse le opzioni per l'accelerazione hardware. Come può essere **eseguito un modello su Android**? Un modello TensorFlow Lite, all'interno di un'app Android, acquisisce ed elabora dati, generando una previsione basata sulla logica dello stesso modello. Per essere eseguito, il modello ha bisogno di uno specifico ambiente di runtime e i dati di ingresso devono essere nel formato speciale tensor. Quando il modello esegue un'inferenza, produce risultati di previsione nel formato tensor e li consegna all'app Android che, a seconda degli output ricevuti, intraprende una o più azioni (come mostrare a schermo il risultato o eseguire specifiche operazioni di calcolo).

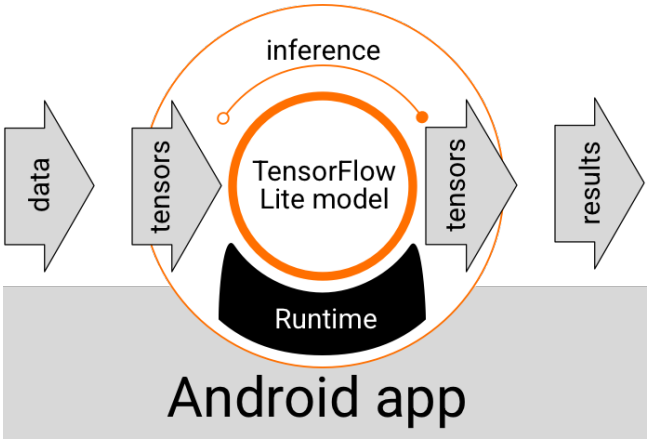

Figura 4.5: Schema di esecuzione di un modello in un app Android.

L'app, quindi, ha bisogno dei seguenti elementi per utilizzare un modello TF Lite:

- Ambiente di runtime: per l'esecuzione del modello;
- Modellatore di input: per convertire i dati di ingresso nel formato textittensor;
- Modellatore di output: per ricevere i tensor di output e interpretarli come risultati di previsione.

4.5.1 Ambiente di runtime

Per l'esecuzione di un modello TensorFlow Lite su un app android c'è bisogno di un ambiente di runtime il quale può essere abilitato nei seguenti modi:

- Google play services.
- Ambiente di runtime TensorFlow Lite Stand-alone.

Utilizzare i servizi Google Play è l'approccio preferito e consigliato. Infatti, questo ambiente di runtime è più efficiente in termini di spazio poiché si carica in modo dinamico, riducendo le dimensioni dell'app. I servizi Google Play, inoltre, utilizzano automaticamente la versione più recente e aggiornata del runtime di TensorFlow Lite, offrendo funzionalità aggiuntive e prestazioni in continuo miglioramento. Adottare un ambiente di runtime fornito da Google Play Services ha anche degli svantaggi: il set di delegati per l'accelerazione hardware da cui si può scegliere si riduce e non sono supportate API TensorFLow Lite sperimentali (come le operazioni personalizzate). Nel caso in cui i servizi Google Play non siano inclusi nell'app o nel caso in cui lo sviluppatore debba gestire il proprio modello machine learning "da vicino", allora è conveniente usare l'ambiente runtime TensorFlow Lite standalone che aggiunge codice nell'app, garantendo allo sviluppatore un maggiore controllo del runtime di machine learning al costo di un incremento della dimensione dell'app. Entrambi gli ambienti di runtime possono essere acceduti utilizzando specifiche API e librerie di sviluppo all'app.

4.5.2 API e librerie di sviluppo

Per integrare i modelli TensorFlow Lite in un applicazione Android, è possibile utilizzare due API molto utili fornite da TensorFlow Lite stesso:

- L'API TensorFlow Lite Task;
- L'API TensorFLow LIte Interpreter.

Quali sono le differenze? L'API Interpreter fornisce classi e metodi per eseguire inferenze su modelli TensorFlow Lite già esistenti, mentre l'API Task racchiude l'API Interpreter in un wrap che dispone di un'interfaccia di programmazione ad alto livello per la gestione di attività di machine learning riguardo

a dati visivi, audio e testuali. L'API Task viene sempre utilizzata nello sviluppo di applicazioni AI mobile, salvo casi specifici in cui il dispositivo non è in grado di supportare tale API. Entrambe le API possono essere accedute tramite i servizi di Google Play.

4.5.3 I tensori

Al fine di comprendere maggiormente la prossima sezione, è giusto approfondire il concetto di tensore[10]. Un tensore è un array multidimensionale con un tipo uniforme (chiamato dtype, o data type) di cui vettori, matrici, funzioni lineari ed endomorfismi costituiscono casi particolari. Tutti i tensori sono immutabili: non è mai possibile modificare il contenuto di un tensore, si può solamente crearne uno nuovo (ad eccezione del caso specifico dei tensori di tipo tf.Variable). Oltre al data type, i tensori possiedono due caratteristiche principali: la forma (o shape), ossia la lunghezza di ogni dimensione, e il grado (o rank) che consiste nel numero di dimensioni del tensore e può essere considerato il suo ordine di grandezza. Per esempio, uno scalare è un tensore di rank 0 e di shape nulla, un vettore è un tensore di rank 1 e di shape che corrisponde al numero di elementi del vettore, una matrice è un tensore di rank 2 (2 dimensioni) e di shape costituita da due numeri: il numero di righe e il numero di colonne.

Al fine di creare un rete neurale, si utilizzano delle rappresentazioni grafiche per mostrare il flusso di calcolo dei dati e le operazioni avvenute. Per fare ciò, si utilizza un grafo: la rappresentazione, per mezzo di nodi, di operazioni eseguite sui tensori. Il grafo è costituito da nodi che rappresentano i tensori e i dati memorizzati all'interno di essi e da nodi che rappresentano le operazioni su tali dati. I tensori si uniscono tra di loro per formare unità neurali più complesse, come il percettrone. Una rete neurale è costituita da innumerevoli percettroni, caratterizzando la complessità del mondo AI. Dopo questa brevissima introduzione dei tensori, torniamo a TensorFlow Lite che, come suggerisce il nome, sfrutta a pieno la struttura dati appena introdotta.

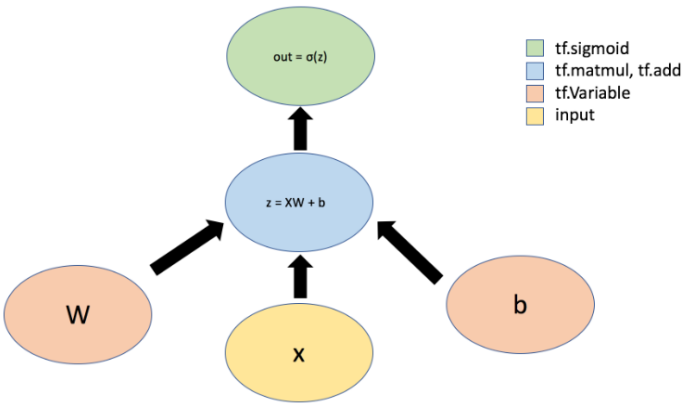

Figura 4.6: Percettrone, esempio di un semplice grafo con tensori e operazioni

4.5.4 Gestione dei dati in input

Come già anticipato, per elaborare i dati di input è necessario che essi siano nel formato di tensore. Il tensore, inoltre, deve essere in una specifica forma. Ogni modello TensorFlow Lite di machine learning, quindi, deve trasformare un dato in input dal suo formato nativo (immagine, testo, audio…) in un tensore con la specifica forma richiesta dal modello. Molto spesso il formato richiesto viene specificato nei metadati del modello. Come eseguire tale trasformazione? TensorFlow Lite fornisce una libreria apposita chiamata TensorFlow Lite Task che dispone di una logica di gestione dati per la conversione dei dati da un qualsiasi formato a un tensore con la forma giusta. Una volta ottenuto un ambiente di runtime ottimale, un modello e dei dati di input nel formato richiesto è possibile eseguire inferenze come descritto precedentemente.

4.5.5 Gestione dei risultati in output

Eseguita l'inferenza, il modello produce in output dei risultati nella forma di tensori che devono essere gestiti ed interpretati dall'applicazione Android agendo o mostrando a schermo un risultato per l'utente. I tensori di output possono essere dei semplici numeri che corrispondono ad un singolo risultato (per esempio, 0 = gatto, 1 = cane, 2 = pesce) per una classificazione di immagini, ma possono essere anche risultati estremamente complessi come diversi riquadri di delimitazione nel caso in cui vi siano più oggetti classificati in un'immagine, insieme a delle valutazioni di affidabilità delle previsioni nell'intervallo [0, 1]. Molto spesso la descrizione dei risultati di output e di come interpretarli viene fornita dai metadati incorporati nel modello.

4.5.6 Approcci di sviluppo avanzati

Nel caso in cui vengano utilizzati modelli TensorFlow Lite più sofisticati, potrebbe essere necessario utilizzare percorsi di sviluppo avanzati e più complicati rispetto a quanto descritto sopra. Quindi, si utilizzano delle tecniche avanzate per sviluppare ed eseguire modelli TensorFlow LIte in un'app Android:

- Ambiente di runtime avanzato: quando si dispone di un modello di machine learning che utilizza operazioni non supportate dall'ambiente di runtime standard di TensorFlow Lite o da quello fornito dai servizi di Google Play, è possibile utilizzare degli ambienti runtime avanzati: l'ambiente runtime TensorFlow Lite Flex, che permette di implementare operatori specifici per il proprio modello, e l' ambiente runtime TensorFlow Lite personalizzato. Come opzione ancora più avanzata, è possibile importare la libreria TensorFlow Lite su Android per l'inclusione di funzionalità e operatori richiesti per eseguire il proprio modello.

- APIs di C e C++: TensorFlow Lite fornisce delle API per eseguire modelli usando C e C++. L'utilizzo di queste API è consigliato quando l'applicazione usa Android NDK o se si vuole condividere codice tra piattaforme.
- Esecuzione del modello su un server: l'esecuzione dei modelli nella propria app Android permette di abbassare la latenza e migliorare la privacy per gli utenti. In alcuni casi, però, conviene eseguire il proprio modello su un server cloud: per esempio, questo approccio è consigliato se il modello è troppo grande e non è facilmente comprimibile in una dimensione gestibile da dispositivi Android degli utenti. Inoltre, si utilizza questa tecnica quando è prioritario garantire una performance consistente del modello su un vasto range di dispositivi.
- Ottimizzazione di modelli personalizzati: è importante considerare di ottimizzare il proprio modello personalizzato per ridurre costi di elaborazione e di memoria. L'approccio consigliato è sempre quello della quantizzazione.

Capitolo 5

PyTorch Mobile vs TensorFlow Lite

Sono stati singolarmente analizzati entrambi i framework, ma quale bisogna scegliere per l'implementazione di modelli di machine learning su dispositivi mobili? Ci si limiterà a confrontare i due framework sulle caratteristiche più importanti per lo sviluppo di un modello:

- Facilità di implementazione: un punto critico per ogni sviluppatore è quello di poter implementare facilmente un progetto e iniziare ad usarlo fin da subito senza troppe configurazioni e setup iniziali. In questo contesto, TensorFlow Lite offre un set vastissimo di APIs[11] che permettono di semplificare il processo di implementazione in modo significativo. Tendenzialmente, TensorFlow Lite richiede solo 2 ore per implementare un semplice prototipo. È possibile implementare le stesse funzionalità anche con PyTorch Mobile ma non si dispone di tutte queste API che facilitano e velocizzano il processo.

- Manipolazione low-level del modello: al contrario di TensorFlow Lite, PyTorch è rinomato per la sua flessibilità nella manipolazione del modello[8], offrendo la possibilità di personalizzare il modello fino ai minimi dettagli.

- Velocità di inferenza: una piacevole esperienza di utilizzo e creazione di un modello machine learning dipende molto anche dalla velocità con cui si riesce a testarlo e praticarlo. In questo contesto, TensorFlow Lite riesce a garantire un tempo di inferenza 3 volte più piccolo di quello di PyTorch Mobile grazie al supporto della GPU e dei suoi delegati (NNAPI).

- Dimensione: un'importante metrica da considerare nell'ambito della progettazione di modelli su dispositivi mobili è la dimensione del modello stesso. La capacità di memoria su questi dispositivi è minima e, per questo, c'è bisogno di un modello il più leggero possibile. Anche su questo fronte, TensorFlow Lite riesce a comprimere maggiormente la dimensione del modello garantendo un occupazione minima di memoria.

- Community e supporto: al fine di un uso più pacifico e sereno di un framework, avere una community fidata e del supporto ufficiale diventa essenziale. Sapere che c'è un team dedicato per segnalazioni di bug, richieste di nuove features e per un contributo attivo alla libreria è un fattore chiave. In questo contesto, TensorFlow Lite e PyTorch Mobile presentano entrambi una community attiva e una documentazione egualmente buona. Inoltre, sono entrambi open source.

In conclusione, TensorFlow Lite è la scelta migliore: maturità, supporto della GPU, dimensione delle librerie e del modello e larga gamma di API sono degli ottimi motivi per questa decisione. PyTorch Mobile resta comunque un'opzione plausibile, soprattutto nei casi in cui è richiesta una personalizzazione profonda del modello e una flessibilità di controllo notevole.

Bibliografia

[1] Paul Bridger. *Mastering TorchScript: Tracing vs Scripting, Device Pinning, Direct Graph Modification*. URL: https://paulbridger.com/posts/mastering-torchscript/.

[2] *Deploy machine learning models on mobile and edge devices*. URL: https://www.tensorflow.org/lite.

[3] Android Developers. *API Neural Networks*. URL: https://developer.android.com/ndk/guides/neuralnetworks?hl=it.

[4] Marat Dukhan. *Accelerating TensorFlow Lite with XNNPACK Integration*. 2020. URL: https://blog.tensorflow.org/2020/07/accelerating-tensorflow-lite-xnnpack-integration.html.

[5] IBM. *Cos'è PyTorch?* 2024. URL: https://www.ibm.com/it-it/topics/pytorch.

[6] *Introducing Accelerated PyTorch Training on Mac*. URL: https://pytorch.org/blog/introducing-accelerated-pytorch-training-on-mac/.

[7] Hao Lu Marat Dukhan Yiming Wu. *QNNPACK: Open source library for optimized mobile deep learning*. 2018. URL: https://engineering.fb.com/2018/10/29/ml-applications/qnnpack/.

[8] Mustafa Mohammadi. *PyTorch vs. TensorFlow for Mobile Machine Learning Applications*. 2023. URL: https://www.linkedin.com/pulse/pytorch-vs-tensorflow-mobile-machine-learning-mustafa-mohammadi-b8a1c.

[9] Sujatha Mudadla. *PyTorch Mobile*. 2023. URL: https://medium.com/@sujathamudadla1213/pytorch-mobile-a5dc9cabe511.

[10] Andrea Provino. *Tensorflow Guida Italiano: primi passi con Tensorflow*. 2020. URL: https://www.andreaprovino.it/tensorflow-guida-italiano-primi-passi-con-tensorflow.

[11] Federico Puy. *TensorFlow Lite vs PyTorch Mobile for On-Device Machine Learning*. 2024. URL: https://proandroiddev.com/tensorflow-lite-vs-pytorch-mobile-for-on-device-machine-learning-1b214d13635f.

[12] *PyTorch Mobile: End-to-end workflow from Training to Deployment for iOS and Android mobile devices*. URL: https://pytorch.org/mobile/home/.

[13] *PyTorch Vulkan Backend User Workflow*. URL: https://pytorch.org/tutorials/prototype/vulkan_workflow.html.

[14] *Quantization*. URL: https://pytorch.org/docs/stable/quantization.html.

[15] *TensorFlow*. URL: https://www.databricks.com/it/glossary/tensorflow-guide.

[16] *TensorFlow: cos'è e a cosa serve*. URL: https://www.egovaleo.it/geek-lab/tensorflow-cose-e-a-cosa-serve/.

[17] *Torch (machine learning)*. 2024. URL: https://en.wikipedia.org/wiki/Torch_(machine_learning).

[18] *torch.utils.mobile_optimizer*. URL: https://pytorch.org/docs/stable/mobile%5C_optimizer.html.