UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Progetto Programmazione di Sistemi Embedded: Mobile AI

Autori
Alessandro Girlanda, Andrea Mutti, Umberto Bianchin

ANNO ACCADEMICO 2023-2024

Indice

Elenco delle figure

Capitolo 1

NNAPI

Capitolo 2

PyTorch Mobile

2.1 PyTorch & PyTorch Mobile

PyTorch[3] 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**[8] 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 un così detto **TorchScript**, che permette di creare modelli da PyTorch che possono poi venire salvati e caricati in un processo dove non c'è alcuna dipendenza di Python.

PyTorch Mobile[5], 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.

2.2 Caratteristiche Principali

Le caratteristiche principali di PyTorch Mobile, così come scritto sul sito ufficiale[6], 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[2] 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[4] (Quantized Neural Network PACKage) è progettata per accelerare le reti neurali quantizzate[2] su hardware mobile.

2.3 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 2.1; 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.

2.3.1 Quantizzazione

La quantizzazione[7] 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 principalmente nella fase di inferenza del

[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.

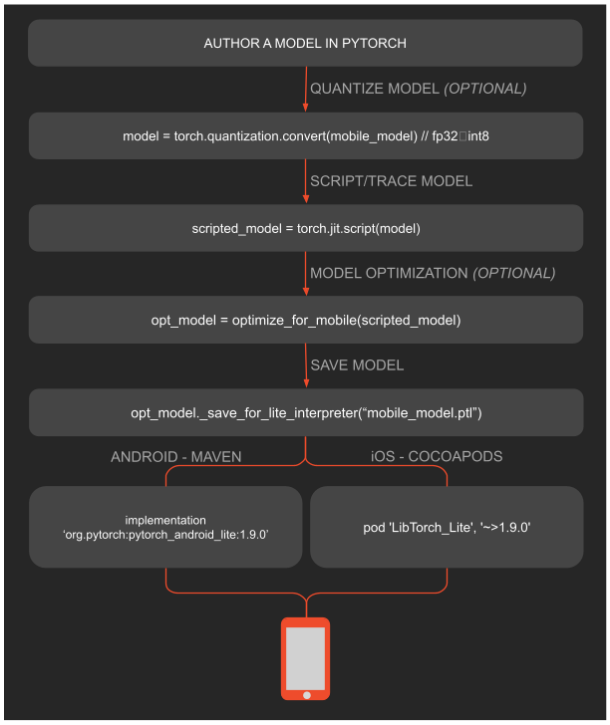

Figura 2.1: Workflow dal training al rilascio di un modello su piattaforma mobile. Fonte [6]

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 2.2:

- 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 2.2: Operatori compatibili con i due diversi tipi di quantizzazione

2.3.2 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.

2.3.3 Ottimizzazione

Grazie alla funzionalità `torch.utils.mobile_optimizer.optimize_for_mobile`[9] 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

Capitolo 3

TensorFlow

TensorFlow 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, assistendo 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, quali:

- Classificazione delle immagini;
- Elaborazione del linguaggio naturale;

Figura 3.1: Diagramma dei punteggi di utilizzo di vari framework nel 2018

Capitolo 4

PyTorch Mobile vs TensorFlow Lite

Bibliografia

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

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

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

[4] 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/.

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

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

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

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

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