Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Ingegneria Informatica, delle Comunicazioni ed Elettronica

Elaborato finale

Titolo

Sottotitolo (alcune volte lungo - opzionale)

Supervisore
Elisa Ricci

Laureando
Matteo Beltrami

Anno accademico 2022/2023

Ringraziamenti

...thanks to...

Indice

Sommario

Negli ultimi anni, il Machine Learning è diventato una delle tecnologie più utilizzate in molti campi, dalla computer vision per il settore automobilistico al riconoscimento del linguaggio naturale per gli assistenti vocali. L'implementazione di modelli di Machine Learning su dispositivi con risorse limitate, come microcontrollori e sensori a bassa potenza, è stata tuttavia a lungo considerata una sfida tecnologica a causa delle limitazioni di memoria e potenza di elaborazione.

In questo contesto, è emersa la tecnologia del Tiny Machine Learning (TinyML) come soluzione per l'elaborazione di dati di sensori su dispositivi a capacità limitate. Il TinyML è l'apprendimento automatico implementato su dispositivi embedded a basso consumo energetico in grado di processare dati in tempo reale senza la necessità di trasferirli a un server remoto.

Il Tiny Machine Learning offre numerosi vantaggi nell'elaborazione dei dati di sensori su dispositivi integrati a bassa potenza; dato che non è neceassaria alcun tipo di connettività per l'inferenza, si possono elaborare i dati in tempo reale direttamnte sul dispositivo e si può evitare di trasferire dati grezzi a terze parti per l'elaborazione, ciò evita i ritardi dovuti all'attesa di inutili tempi di trasmissione delle informazioni e riduce notevolmetne la latenza. Per comprendere l'impatto della bassa latenza di risposta, basti pensare al monitoraggio continuo delle macchine in ambito industriale per prevederne i problemi ed i possibili guasti. Questo tipo di applicazione può dare una risposta tempestiva ai danni, riducendo i costi di manutenzione, i rischi di guasto e i tempi di inattività, oltre a migliorare le prestazioni e l'efficienza.

I dispositivi integrati che supportano gli algoritmi di TinyML hanno bisogno di una quantità molto ridotta di potenza (nell'ordine dei mW), che consente loro di operare per lunghi periodi senza bisogno di essere caricati se provvisti di batteria, o con un consumo esiguo di energia se alimentati. Pensando invece a sistemi di video sorveglianza nelle smart cities, la possibilità di eseguire l'inferenza sul dispositivo senza la necessità di trasferire i dati per l'elaborazione a server esterni, oltre a diminuire i consumi di potenza dei dispositivi, aumenta il livello di privacy dei dati.

0.1 Classificazione di immagini

L'elaborazione e la classificazione di immagini sono diventate aree cruciali nel campo del machine learning. La capacità di comprendere e interpretare le informazioni visive è di fondamentale importanza in molti settori, quali la medicina, l'automazione industriale, la sorveglianza e molto altro. La classificazione di immagini con tecniche di machine learning consente di addestrare algoritmi complessi per riconoscere e assegnare etichette a diverse categorie di immagini in modo automatico. L'obiettivo è quello di creare modelli che siano in grado di apprendere i pattern e le caratteristiche distintive presenti nelle immagini per effettuare previsioni accurate sulla classe di appartenenza di nuove immagini mai viste prima.

Le Convolutional Neural Networks sono tipicamente composte da strati di convoluzione, pooling e strati di classificazione e sono progettate appositamente per estrarre le caratteristiche salienti dalle immagini. Attraverso la convoluzione e l'applicazione di filtri, le CNN catturano dettagli significativi, riducendo la complessità dei dati, tramite gli strati di pooling, si riducono le dimensioni delle rappresentazioni intermedie per concentrarsi sulle informazioni essenziali, infine gli strati di classificazione combinano le caratteristiche estratte per effettuare la predizione della classe di appartenenza dell'immagine. Grazie al processo di apprendimento e all'ottimizzazione dei pesi, queste reti sono in grado di affinare le loro prestazioni nel riconoscimento e nella classificazione di immagini. L'intero processo richiede un'adeguata preparazione dei dati di addestramento (dataset), che devono essere etichettati correttamente; successivamente le immagini vengono suddivise in un set di addestramento, un set di validazione e un set di test. Durante la fase di addestramento, le CNN sono sottoposte a un processo iterativo in cui i pesi dei vari strati vengono regolati per ridurre l'errore tra le etichette reali

e le previsioni della rete; ciò avviene mediante l'ottimizzazione di una funzione di perdita, come ad esempio la *cross-entropy loss*. Una volta che la rete è stata addestrata, viene valutata utilizzando il set di validazione per misurare la sua capacità di generalizzazione; infine il modello viene testato sul set di test per valutare le sue prestazioni finali, ovvero l'accuratezza nella classificazione di nuove immagini. La classificazione di immagini con tecniche di machine learning offre numerosi vantaggi, come l'automazione dei compiti di classificazione che richiederebbero molto tempo se eseguiti manualmente; inoltre permette di ottenere previsioni precise e affidabili, contribuendo a migliorare l'efficienza e la qualità delle attività che coinvolgono l'elaborazione di grandi quantità di immagini.

Tuttavia, è importante sottolineare che i modelli di classificazione di immagini più performanti spesso richiedono una grande quantità di risorse computazionali, tra cui potenza di calcolo e memoria. Questo aspetto rappresenta una sfida significativa nell'implementazione pratica di tali modelli, specialmente su dispositivi con risorse limitate come i microcontrollori. Qui sorge l'importanza del Tiny Machine Learning, che si propone di implementare modelli di machine learning anche su dispositivi a capacità computazionali limitate.

0.2 Tecniche di implementazione di TinyML

Una delle sfide principali che i ricercatori e gli ingegneri affrontano nell'implementazione del Tiny Machine Learning è l'eterogeneità delle piattaforme. A causa della grande varietà di microcontrollori, sensori e dispositivi integrati disponibili sul mercato, c'è una notevole diversità di architetture hardware e software che può rendere difficile lo sviluppo e la distribuzione di soluzioni TinyML su una vasta gamma di dispositivi.

L'eterogeneità delle piattaforme può causare problemi di compatibilità tra hardware e software e può rendere difficile la creazione di modelli che funzionino in modo affidabile su più dispositivi, ciò può comportare l'implementazione di soluzioni specifiche per singoli dispositivi, riducendo l'efficienza del processo di sviluppo ed aumentandone i costi.

Inoltre, i microcontrollori spesso hanno limitazioni in termini di memoria (Flash e RAM) e capacità di elaborazione. Queste limitazioni possono essere diverse tra differenti dispositivi, il che può rendere difficile la creazione di modelli di TinyML che funzionino indistiantamente per diverse piattaforme con risore limitate senza incorrere ad eccedere vincoli architetturali o senza compromettere l'accuratezza dei risultati. Per evitare questi problemi, è opportuno per gli sviluppatori di TinyML implementare soluzioni che siano compatibili con il maggior numero possibile di dispositivi, il che può includere l'adozione di standard aperti e l'uso di architetture software flessibili e scalabili che consentano di adattarsi alle esigenze di un'ampia gamma di dispositivi.

Alcune tecniche che mirano alla creazione di modelli efficienti, pur mantenendo la flessibilità di adattamento a piattaforme con risorse computazionali stringenti, sono la compressione del modello tramite *pruning* e *quantizzazione*, lo *scaling* (ad oggi lo stato dell'arte è il *compound scaling*) e il *neural architecture serach*. La prima pratica prevede di comprimere il modello già allenato in modo tale da ridurre le connessioni neurali eliminando (*pruning*) quelle i cui pesi sono minori di una certa soglia e che quindi hanno un impatto irrilevante sul risultato finale di predizione; la *quantizzazione* invece mira a ridurre lo spazio in memoria occupato dai pesi della rete riducendo la loro precisione in termini di bit (passando ad esempio da una codifica a 32bit ad una a 8bit). Lo *scaling* si basa sul modificare alcune caratteristiche di una rete (numero di layer/risoluzione/canali) in modo da adattarla ai vincoli computazionali imposti cercando di preservare l'accuratezza di predizione; il *neural architecture serach* invece è un nuovo ambito di ricerca molto complesso con l'intento di trovare la miglior architettura per una rete neurale per un compito specifico, automatizzando la ricerca e selezionando le caratteristiche della rete che sembrano portare a risultati migliori.

0.3 Scaling di Convolutional Neural Networks

Lo *scaling* (ed il *compound scaling*) è sicuramente la tecnica più utilizzata per implementare algoritmi efficienti di machine learning con poche esigenze di risorse. Le caratteristiche principali di una CNN che possono essere "scalate" sono la profondità, ovvero il numero di strati convolutivi presenti, la larghezza, ovvero il numero di canali delle rappresentazioni intermedie, e la risoluzione, ovvero le dimensioni in pixel delle immagini di input; queste proprietà sono in genere dette *fattori di scala*.

Per lo sviluppo di algoritmi di machine learning in grado di scalare è buona prassi implementare intrinsicamente nell'architettura della rete le tecniche di *scaling*, singolarmente o combinate, in maniera personalizzabile, in modo tale da permettere la modifica delle caratteristiche della rete (tra cui memoria e potenza computazionale) tramite dei fattori di scala e generare quindi un modello ottimizzato per determinati dispositivi con risorse limitate.

Abbiamo quindi appurato che tramite l'aggiustamento dei fattori di scala è possibile inizializzare una rete da allenare che soddisfi determinati vincoli di occupazione di memoria e di potenza computazionale; tuttavia ci sono diverse configurazioni dei fattori che ci portano ad avere un'architettura di rete differente, con le stesse esigenze di risorse ma con possibili risultati di accuratezza discrepanti, da qui nasce quindi il mio obiettivo di ricerca.

Dato un preciso budget di risorse computazionali, quale è la miglior configurazione dei fattori di scala che soddisfa i vincoli imposti e massimizza la performance di classificazione?

Per riuscire a rispondere a questa domanda bisogna innanzitutto selezionare un modello di rete di riferimento su cui condurre questa ricerca, ho scelto quindi la famiglia di reti delle PhiNet[8], ovvero un'architettura scalabile ottimizzata per l'elaborazione delle immagini basata sul deep learning per piattaforme con risorse limitate. Una volta individuata l'architettura bisogna essere in grado di comprendere quali sono le variabili che influenzano le performance finali (scelta del dataset, risoluzione delle immagini di input, utilizzo o meno della tecnica di data augmentation, numero di epoche per l'addestramento, ottimizzatore utilizzato durante il training, valore del learning rate) e definire un sottodominio di ricerca con una precisa configurazione di tali varibili; questo restringimento del dominio è necessario per poter rendere l'analisi coerente e replicabile e non spostare il focus su fattori esterni all'ambito di ricerca. Bisogna inoltre comprendere come i fattori di scala influenzano la struttura della rete e riuscire ad analizzare la variazione delle performance al variare della scala applicata.

Le tecniche proposte per svolgere questo tipo di ricerca mirano ad analizzare i livelli di performance del modello in funzione dei fattori di scala utilizzati. Questi approcci variano da metodologie più "applicative", quale lo studio del contributo degli strati convolutivi intermedi sulle performance finali di classificazione analizzando la separabiltà lineare degli output di ogni strato (Linear Probing), fino allo sviluppo di modelli di regressione e segmentazione di dati per l'individuazione dei fattori di scala migliori per determinati budget di risorse.

Attraverso questa analisi, il presente studio propone un framework di lavoro applicabile a diverse architetture per il TinyML, aprendo le strade a futuri studi e offre un contributo personale nella verifica dell'efficacia del Linear Probing per studiare le performance di una rete e nell'individuazione del numero di strati convoluzionali che ottimizza le performance di classificazione delle PhiNet per un certo budget computazionale. I risultati hanno rivelato che la scelta del numero di strati dipende strettamente dal budget di memoria disponibile (in termini di numero di parametri). In particolare, quando il numero di parametri disponibili è minore di $2 \cdot 10^3$, il numero di strati ottimale risulta essere 4. Nel range di parametri compreso tra $2 \cdot 10^3$ e 10^4 il numero di strati ideale diventa 5, mentre fino a $9 \cdot 10^5$ parametri il numero ottimale di strati si attesta a 7. Questi esiti forniscono una guida preziosa per la configurazione ottimale del modello tenendo conto delle specifiche risorse disponibili contribuendo così a una migliore gestione delle risorse computazionali nel contesto delle applicazioni di TinyML.

Sommario è un breve riassunto del lavoro svolto dove si descrive l'obiettivo, l'oggetto della tesi, le metodologie e le tecniche usate, i dati elaborati e la spiegazione delle conclusioni alle quali siete arrivati.

Il sommario dell'elaborato consiste al massimo di 3 pagine e deve contenere le seguenti informazioni:

- contesto e motivazioni
- breve riassunto del problema affrontato
- tecniche utilizzate e/o sviluppate
- risultati raggiunti, sottolineando il contributo personale del laureando/a

1 Lavori correlati

1.1 Convolutional Neural Networks

Le Convolutional Neural Networks (CNN) rappresentano un pilastro fondamentale nell'ambito della classificazione di immagini mediante algoritmi di machine learning e comprendere il loro funzionamento è essenziale per apprezzarne l'efficacia e per studiarne l'applicabilità nei vari contesti.

Esse si ispirano al funzionamento del sistema visivo umano, cercando di emulare la capacità del cervello di estrarre automaticamente caratteristiche rilevanti dalle immagini. Una delle proprietà distintive di queste reti è la struttura tipica a strati che svolge un ruolo fondamentale nell'elaborazione di immagini; questa architettura stratificata consente di scomporre il processo di classificazione in una serie di operazioni intermedie, ciascuna con una specifica funzione.

1.1.1 Funzionamento

Tipicamente in una CNN si susseguono diversi strati, o *layer*, di operazioni che modificano l'immagine presa in input e la trasformano in rappresentazioni intermedie modificate, che sono difficili da comprendere per l'occhio umano, ma funzionali per l'interpretazione dei layer successivi e del classificatore finale. Le operazioni svolte da ogni strato della rete comprendono la *convoluzione*, applicata tra l'input di uno strato ed il suo filtro (anche detto "kernel"), ed il *pooling*, ovvero operazioni per ridurre la dimensione delle raffigurazioni intermedie ai vari strati.

Durante la convoluzione, il filtro, solitamente di dimensione 3×3 o 5×5, scansiona l'immagine di input e calcola la somma del prodotto punto a punto tra i valori dell'input e i pesi del filtro. Il risultato di questa operazione è una rappresentazione, anche detta *feature map*, che evidenzia le caratteristiche rilevanti dell'immagine, come bordi, texture o forme. Le operazioni di pooling hanno invece lo scopo di ridurre la dimensione delle rappresentazioni intermedie ai vari layer, questa funzione viene applicata su una finestra di dimensione predefinita dell'input e consiste nella selezione del valore medio o massimo (rispettivamente average pooling e max pooling), riducendo la dimensione dell'input e mantenendo le informazioni salienti. Tale processo di riduzione della dimensionalità aiuta a semplificare il modello, diminuendo il numero di parametri e migliorando l'efficienza computazionale. Inoltre, per consentire alle CNN di modellare relazioni complesse, apprendere rappresentazioni gerarchiche e superare la limitazione della linearità intrinseca degli strati di convoluzione, diventa necessario l'utilizzo delle funzioni di attivazione che introducono una non-linearità nel processo di classificazione e aiutano la rete a modellare relazioni complesse nei dati. Queste funzioni trasformano l'output di un neurone in base a una determinata regola e le più utilizzate per il task di classificazione di immagini sono la funzione sigmoide e la ReLU (Rectified Linear Unit) che è particolarmente popolare perché introduce una non-linearità semplice e computazionalmente efficiente, permettendo anche una maggiore velocità di convergenza durante l'apprendimento.

Attraverso la combinazione di convoluzione e pooling in diversi strati, una CNN è in grado di estrarre progressivamente caratteristiche complesse dall'immagine. I layer iniziali si concentrano su caratteristiche più semplici, come linee e angoli, mentre quelli successivi combinano tali caratteristiche per identificare oggetti più complessi e definire classi specifiche. [9]

Alla fine degli strati di convoluzione e pooling va posizionato il classificatore, ovvero un layer lineare che prende in input tutte le feature in output dall'ultimo blocco convolutivo e restituisce le probabilità di appartenenza dell'immagine originaria ad ogni classe.

Finora abbiamo esaminato il funzionamento e la struttura del modello; tuttavia, è fondamentale anche l'addestramento del modello tramite l'utilizzo di un ampio dataset, che rappresenta un insieme di dati necessario per consentire al modello di apprendere e ottimizzare i suoi pesi.

1.1.2 Dataset

Il dataset è una raccolta di dati strutturata che contiene le informazioni necessarie per poter addestrare una rete neurale, in particolare nelle applicazioni di image classification il dataset è costituito da immagini e dalle relative etichette che ne definiscono la classe di appartenenza.

Questi set di dati sono caratterizzati da una vasta diversità e un numero significativo di immagini per ogni classe, fornendo una rappresentazione completa delle categorie di interesse. La qualità del dataset è essenziale: deve essere accurato e bilanciato, con etichette di classe corrette e coerenti. Inoltre, la varietà delle immagini presenti nel dataset deve coprire in modo adeguato le varie categorie e le possibili variazioni all'interno di ogni classe; ciò assicura che il modello sia esposto a una vasta gamma di esempi durante l'addestramento, rendendolo più robusto e garantendone l'efficacia e la capacità di generalizzazione.

Per valutare le prestazioni del modello in modo affidabile e imparziale, evitando il rischio di sovradattamento (overfitting) ai dati di addestramento è ampiamente usata la tecnica *data splitting* che consiste nel suddividere il dataset disponibile per l'addestramento in tre diversi raggruppamenti nella maniera più equa possibile; in tal modo si ottengono set di dati indipendenti che possono essere usati in maniera differente durante la fase di addestramento e di valutazione delle performance. La pratica più comune è quella di suddividere l'insieme rispettivamente in dati di allenamento, o *training set*, che di solito costituiscono l'80% del dataset originario e sono utilizzati per far imparare al modello i pesi che migliorano l'accuratezza di predizione, dati di validazione, o *validation set*, che ne costituiscono il 10% e si utilizzano per validare la capacità predittiva e per evitare l'*overfitting*, e dati di test, o *test set*, che ne costituiscono il rimanente 10% e vengono utilizzati una sola volta per valutare le prestazioni finali del modello ed ottenere una stima accurata delle capacità di generalizzazione della rete su dati mai visti prima, simulando il contesto di applicazione reale.

La scelta delle proporzini dei vari set di dati è soggettiva e le opzioni più gettonate per trainig, validation e test set sono 80%-10%-10%, 70%-15%-15% e 60%-20%-20%, ad ogni modo un aspetto cruciale è la suddivisione equa e rappresentativa delle classi e delle variazioni dei dati in ogni set. Ciò garantisce che il modello venga valutato in modo accurato su tutte le categorie e che le sue prestazioni siano generalizzabili.

Nel caso studio qui presentato sono stati utilizzati due datasets differenti per mostrare come la scelta del dominio di ricerca influenzi i risultati e poterli quindi confrontare; la scelta è ricaduta su *CIFAR-10* e *CIFAR-100*, due dataset composti da 60mila immagini RGB etichettate di dimensione 32×32 pixel che consistono rispettivamente di 10 classi da 6000 immagini ognuna (CIFAR-10) e 100 classi da 600 immagini ciascuna (CIFAR-100), suddivise in set secondo le proporzini 70%-15%-15%, principalmente raffiguranti esseri viventi, mezzi di trasporto e oggetti vari. [6]

1.1.3 Training e ottimizzazione

L'addestramento di una rete neurale per la classificazione di immagini è un passaggio cruciale per insegnare al modello a riconoscere e assegnare correttamente le diverse categorie alle immagini.

Dopo aver suddiviso il dataset secondo le logiche illustrate sopra, e soprattutto dopo aver definito l'architettura di rete da allenare, si passa all'inizializzazione dei parametri del modello da ottimizzare, in particolare per le CNN ci si riferisce ai *pesi* ed ai *bias* utilizzati dagli strati convolutivi. Come già anticipato, infatti, durante una convoluzione le rappresentazioni in input agli strati convolutivi vengono filtrate tramite un prodotto punto a punto tra i valori dei pixel della finestra scorrevole ed i pesi del kernel, i risultati di queste moltpilicazioni vengono poi sommati ed il tutto è addizionato al bias riferito al filtro. Sono proprio questi i parametri che vanno a definire i risultati di classificazione di un modello e l'obiettivo è trovare quella configurazione di pesi e bias dei vari layer che massimizza le performance della rete. È forse inutile affermare che una ricerca esaustiva della configurazione migliore non è una via praticabile data l'enorme quantità di parametri presenti, si rende quindi necessario lo sviluppo di un algoritmo di aggiustamento dei pesi che mira a minimizzare la discrepanza tra le predizioni del classificatore ed i risultati attesi; questo processo prende perciò il nome di *training*.

L'inizializzazione dei parametri consiste quindi nell'assegnare dei pesi predefiniti al modello che verranno in seguito ottimizzati; tale assegnazione può avvenire in diverse modalità, le più comuni sono l'inizializzazione con valori randomici e l'assegnazione secondo criteri che velocizzano la fase di

apprendimetno in base alle funzioni di attivazione utilizzate all'interno della rete. Ne è un esempio la tecinca di inzializzazione dei pesi proposta da Kaiming He (ottimizzata per la funzione ReLU) per la quale ogni peso viene inizilizzato secondo una distribuzione normale con media nulla e varianza uguale a $\sqrt{6/N}$, dove N è il numero di canali di input del layer convoluzionale. [3]

A seguito dell'inizializzazione dei pesi, inizia la fase di allenamento vero e proprio della rete; a questo stadio del processo viene eseguita quella che viene detta *forward propagation*, durante la quale le immagini del trainig set vengono "propagate" attraverso i vari strati del modello e si ottiene in uscita la predizione di classificazione ottenuta con lo stato presente dei pesi. Questi risultati vengono confrontati con quelli attesi (le etichette delle immagini) e viene calcolata una metrica che ne misura la discrepanza tramite una funzione di perdita, anche detta *loss function*, di cui la più comunemente usata è la Cross-Entropy Loss. L'obiettivo è quello di ridurre al minimo la loss in modo da avere dei risultati di predizione in linea con quelli attesi.

Proprio al fine di minimizzare la perdita, entra in gioco quella che viene detta *backward propagation*, ovvero un algoritmo utilizzato per calcolare il gradiente della funzione di perdita rispetto ai parametri della rete ed esprime il contributo di ciascun peso e bias all'errore totale. Questa operazione avviene utilizzando la regola della catena, o *chain rule*, che applica la derivata parziale in ogni layer per calcolare il contributo della loss rispetto ai parametri di tale strato.

Una volta calcolato il gradiente, l'ottimizzazione viene eseguita utilizzando un algoritmo specifico di aggiornamento dei pesi, i più noti dei quali sono il *gradient descent*, lo *stochastic gradient descent* (SGD) e gli ottimizzatori *Adam* e *Lamb*. Questi processi utilizzano il gradiente calcolato per aggiornare i parametri della rete in modo da ridurre l'errore e migliorare le prestazioni del modello durante l'addestramento.

L'intero processo di ottimizzazione dei parametri dovrebbe essere ripetuto per ogni campione del set di training, in modo che la rete possa imparare da tutte le informazioni a disposizione, e pure per molteplici volte. Tuttavia eseguire l'aggiustamento dei pesi per ognuna delle immagini potrebbe rallentare il processo di addestramento (dato che gli algoritmi del calcolo del gradiente e di ottmizzazione andrebbero ripetuti numerose volte) e potrebbe compromettere la stabilità e la capacità di generalizzazione del modello, visto che la rete si adatterebbe troppo ai singoli esempi che possono essere soggetti a maggiore variazione e rumore. Per ovviare a questi probelmi è diventata ormai standard la pratica di dividere il set di training in *batch*, ovvero insiemi di immagini (solitamente 64 o 128) che vengono elaborati contemporaneamente prima di eseguire l'aggiornamento dei parametri. Tale metodo consente di ottenere una stima del gradiente più stabile e rappresentativa, poiché il calcolo del gradiente si basa su più esempi e migliora l'efficienza del processo di addestramento.

Un solo passaggio attraverso il set di allenamento non è abbastanza per poter ottenere un modello con buone performance di classificazione, si rende quindi necessario la ripetizione di tutte le azioni precedenti per un certo numero di *epoche* in modo da peremttere alla rete di apprendere dai dati in modo progressivo e poter convergere ad una soluzione ottimale.

1.1.4 Valutazione delle prestazioni

Dopo aver compreso le tecniche più utilizzate per addestrare i modelli di calssificazione di immagini diventa necessario stabilire delle metriche di valutazione delle prestazioni delle reti per essere in grado di confrontarle. Ad ogni epoca infatti lo stato della rete, con i parametri ottimizzati fino a quel momento, può essere salvato al fine di confrontare le sue capacità di generalizzazione con altri modelli di epoche differenti. A tal scopo la metrica di precisione più impiegata è la *top-1 accuracy* che rappresenta la percentuale di volte che il modello ha predetto correttamente la classe come la sua scelta principale. Questa misura di accuratezza di classficazione, per poter essere significativa, è da calcolare sul validation set in maniera tale da avere una metrica di confronto basata su predizioni effettuate su immagini che non hanno influito sull'allenamento della rete.

La fase di confronto di prestazioni sul test di validazione per modelli provenienti da epoche differenti è necessario perchè non sempre addestrare un modello per più epoche comporta avere prestazioni di classificazione migliori su dati mai visti. Questo problema di generalizzazione prende il nome di *overfitting* che si verifica quando un modello di machine learning è eccessivamente adattato ai dati di addestramento e ha scarse capacità di predizione su nuovi esempi.

Per avere una stima delle reali performance di classificazione della rete si valuta il modello scelto dal confronto precedente sul set di immagini di test che fornisce una stima imparziale delle prestazioni della rete su dati che non ha mai visto prima, consentendo di valutare la capacità di generalizzazione del modello.

1.2 Architetture di deep learning per l'edge computing

Negli ultimi anni c'è stata una crescente necessità di spostare le architetture di deep learning come le Convolutional Neural Networks su dispositivi di edge computing; esigenza che è emersa a causa di diversi fattori spiegati in precedenza, tra cui il bisogno di ridurre la latenza e la larghezza di banda richiesta e la necessità di elaborare i dati in tempo reale.

Per soddisfare questa richiesta, sono state sviluppate architetture di tinyML come MobileNets, EfficientNet e MCUNet progettate appositamente per essere efficienti in termini di risorse computazionali, memoria e consumo energetico, al fine di poter essere eseguite su dispositivi con capacità limitate, come smartphone, dispositivi IoT e microcontrollori. [5] [10] [7]

1.2.1 MobileNets

MobileNets rappresenta una famiglia di CNN ampiamente utilizzata per applicazioni di computer vision, come la classificazione di immagini, per dispositivi mobili ed embedded.

Una delle caratteristiche distintive di MobileNets è la sua struttura innovativa che permette di ottenere modelli efficienti e leggeri e si basa sull'idea di convoluzioni depthwise separabili, che scindono il classico processo di convoluzione in due fasi distinte: la *depthwise convolution* e la *pointwise convolution*.

Le convoluzioni tradizionali applicano un filtro completo a tutti i canali di input, ciò significa che per ogni filtro vengono calcolate le convoluzioni su tutti i canali di input, generando un numero significativo di operazioni. D'altra parte, le convoluzioni depthwise separabili dividono il processo della convoluzione sui canali (*depthwise*) da quella sui singoli pixel (*pointwise*), in modo da ridurre l'onere computazionale. La convoluzione depthwise applica un singolo filtro a ciascun canale di input, riducendo il numero di parametri e operazioni computazionali; successivamente, la convoluzione pointwise combina linearmente i canali di output della convoluzione depthwise, generando un set di rappresentazioni finali (feature maps).

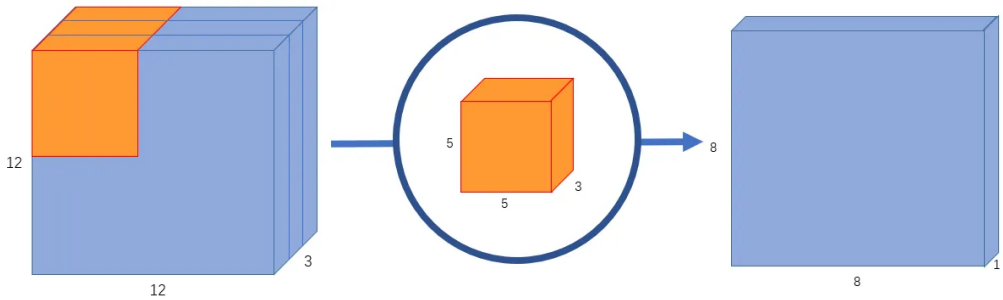

Figura 1.1: *Convoluzione tradizionale.* In questo caso l'immagine originaria ha risoluzione 12×12 ed è composta da 3 canali (come nel caso di una foto RGB). Un filtro convolutivo completo si estende su tutti i canali dell'immagine e genera in uscita una rappresentazione ad un singolo canale con risoluzione minore. Se si volesse avere come output una feature map composta da N canali (come spesso succede nelle CNN), bisognerebbe quindi avere N filtri convolutivi ed eseguire N convoluzioni. Nel caso qui rappresentato l'intero processo richiederebbe $4800 \times N$ moltiplicazioni.

Un altro aspetto innovativo delle MobileNets è l'introduzione dei fattori di scala che consentono di adattare le dimensioni e la complessità della rete in base a delle specifiche esigenze di risorse computazionali.

I due fattori di scaling presentati in questo tipo di architettura sono dei moltiplicatori che regolano rispettivamente il numero di canali in input e in output ad ogni layer (α *width multiplier*) e la risoluzione in ingresso delle immagini (ρ *resolution multiplier*).

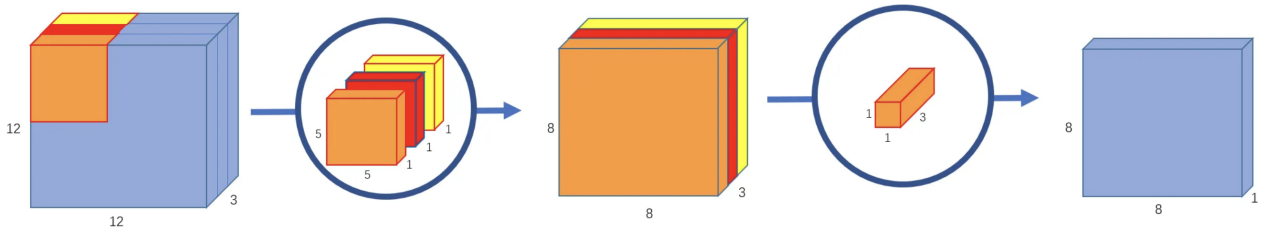

Figura 1.2: *Convoluzione depthwise separabile.* La stessa immagine di partenza viene prima fatta passare attraverso una convoluzione sui canali (depthwise) e si ottiene una rappresentazione intermedia ancora divisa secondo il numero di canali originario; per ottenere in output una rappresentazione ad un solo canale si esgue la convoluzione spaziale sui pixel dell'immagine (pointwise). Al fine di estrarre una feature map composta da N canali, bisognerebbe soltanto avere N filtri convolutivi pointwise ed eseguire N convoluzioni pointwise, che sono ovviamente più leggere ed efficienti. Nel caso qui rappresentato l'intero processo richiederebbe un totale di $4800 + 192 \times N$ moltiplicazioni, che nei casi di applicazione reali si traduce in un guadagno enorme nell'efficienza del modello.

Sebbene il fattore di regolazione della risoluzione non sia una colonna portante di questo tipo di reti, in quanto lo si può impostare implicitamente modificando la risoluzione delle immagini in ingresso al modello, esso aiuta a comprendere come la modifica delle dimensioni delle immagini influenzi il numero di operazioni eseguite dal modello. Scegliendo infatti valori del moltiplicatore ρ appartenenti all'intervallo $(0, 1]$ il costo computazionale si riduce di ρ^2 dato che il numero di operazioni di convoluzione necessarie per analizzare un immagine aumenta quadraticamente rispetto alle sue dimensioni.

Il fattore di regolazione dei canali ha invece il ruolo di assottigliare la rete uniformemente ad ogni strato; scegliendo infatti valori di α nell'intervallo $(0, 1]$ si va a modificare il numero di canali in ingresso M ed in uscita N da ogni layer convolutivo rispettivamente in αM e αN. Il *width multiplier* ha quindi l'effetto di ridurre il costo delle convoluzioni depthwise separabili ed il numero di parametri della rete di α^2, ottenendo così un modello più leggero.

Ovviamente non bisogna dimenticare che bilanciare il consumo di risorse di un modello può avere un impatto sull'accuratezza di predizione della rete, utilizzando ad esempio un fattore α molto piccolo è possibile ottenere una MobileNet leggera e con pochi consumi, ideale per dispositivi mobili con risorse limitate, ma con scarse prestazioni di classificazione. È pertanto necessario gestire attentamente il tradeoff tra efficienza e precisione per ottenere il miglior compromesso tra risorse e prestazioni.

1.2.2 EfficientNet ed il compound scaling

Un'altra tipologia di architettura che si propone di implementare reti convoluzionali efficienti su dispositivi embedded e mobili è EfficientNet.

Fino all'introduzione di questa famiglia di reti, tra le applicazioni di tinyML che intendevano adattare modelli di CNN a dispositivi con risorse limitate, era comune applicare fattori di scala su una sola delle dimensioni caratteristiche del modello; ad esempio le ResNet [2] possono esserre scalate modificando il numero di strati della rete, mentre le MobileNets possono essere ridimiensionate in base al numero di canali. L'idea principale dietro EfficientNet è invece quella di trovare un equilibrio tra la profondità (*depth*), la larghezza (*width*) e la risoluzione dell'immagine (*image size*) di input della rete, dato che studi empirici dimostrano che il bilanciamento congiunto di tali fattori è cruciale per ottenere buone performance.

Per ottenere ciò, EfficientNet utilizza una tecnica chiamata *Compound Scaling*, che regola in modo proporzionale il numero di layer, di canali e la dimensioni delle immagini in input della rete. Ciò significa che all'aumentare della profondità del modello, aumenta anche la larghezza e la risoluzione dell'immagine di input, mantenendo un equilibrio tra queste tre dimensioni chiave.

Compound scaling

Ragionando puramente a livello intuitivo, il metodo del compound scaling risulta efficace perchè all'aumentare delle dimensioni delle immagini in input al modello, diventa necessario l'utilizzo di più

canali nelle rappresentazioni intermedie per catturare pattern più dettagliati ed un numero maggiore di layer convoluzionali per aumentare la capacità percettiva della rete. Già altri studi avevano evidenziato l'esistenza di una certa correlazione tra la larghezza e la profondità di una CNN, ma questa tecnica di scaling qunatifica empiricamente la relazione tra le tre dimensioni del modello.

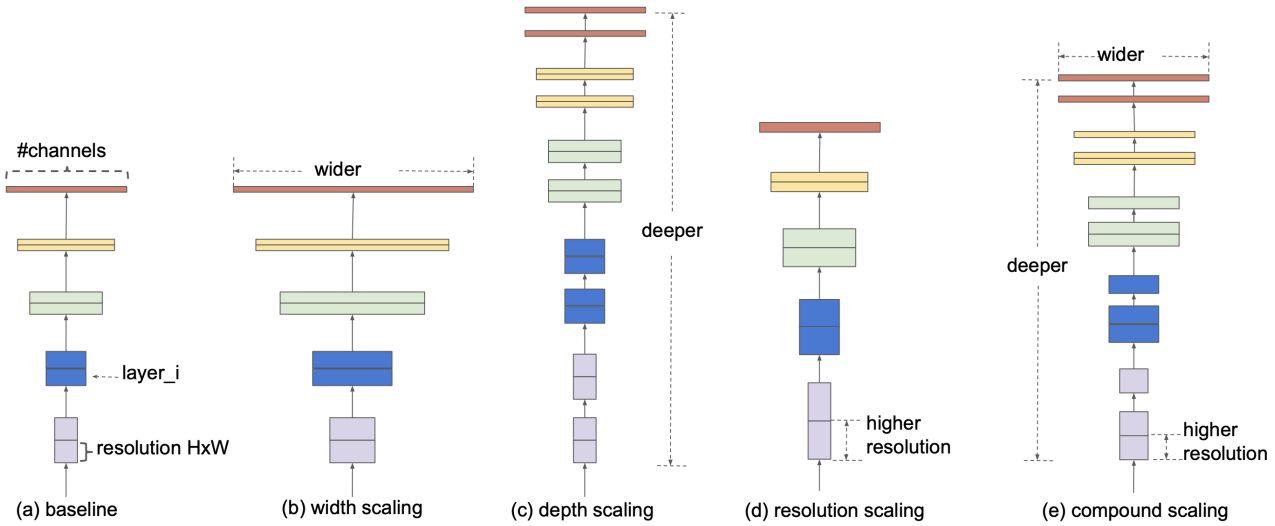

Figura 1.3: *Scaling delle Convolutional Neural Newtworks.* (a) è la rete di riferimento; (b)-(d) mostrano i metodi convenzionali di scaling in cui viene scalata solo una dimensione tra larghezza, profondità e risoluzione dell'immagine; (e) mostra il metodo di compound scaling che ridimensiona uniformemente tutte e tre le dimensioni con un rapporto fisso.

Come mostrato nella figura 1.3 la tecnica del compound scaling mira a conciliare tre diversi metodi di ridimensionamento di una rete neurale convoluzionale, che sono il *width*, *depth* e *resolution scaling*. Per lo scaling di queste dimensioni della rete l'architettura di EfficientNet propone tre differenti coefficienti di scala, o *iperparametri*, per regolare le caratteristiche della rete; tali fattori sono:

- α^{ϕ}: regola la profondità (depth) della rete cambiando il numero di layer presenti in modo da poter catturare rappresentazioni più complese;
- β^{ϕ}: modifica la larghezza (width) del modello tramite il numero di canali in ingresso e in uscita di ogni layer in maniera analoga al funzionamento del width multiplier presentato nella sezione 1.2.1;
- γ^{ϕ}: regola le dimensioni (image resolution) delle immagini in ingresso alla rete secondo le stesse logiche del resolution multiplier della sezione 1.2.1.

L'idea innovativa avanzata dal metodo del compund scaling è l'introduzione del parametro ϕ e l'impostazioni degli iperparametri in modo che:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \qquad \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \tag{1.1}$$

Le combinazioni di questi fattori sono potenzialmente infinite ma possono essere facilmente determinati con una griglia di ricerca basilare; il coefficiente di composizione ϕ invece è specificato dall'utente e regola le risorse necessarie per il funzionamento del modello. Dato che in una rete convoluzionale il numero di operazioni eseguite (paragonabile al consumo di potenza) è proporzionale a $depth, width^2, resolution^2$ del modello e che è stata impostata la relazione 1.1 tra gli iperparametri, ne consegue che scegliendo arbitrariamente un valore di ϕ, il numero di operazioni viene incrementato approssimativamente di 2^{ϕ}.

L'effetto combinato di questi fattori di scala permette quindi ad EfficientNet di raggiungere un giusto equilibrio tra potenza di calcolo, numero di parametri e prestazioni di classificazione.

1.2.3 MCUNet ed il tinyNAS

MCUNet rappresenta un'innovativa architettura progettata per affrontare il complesso problema di portare le convolutional neural networks su piccoli dispositivi per l'Internet of Things e su microcontrollori (MCU, microcontroller units) a basse risorse; questa risulta essere una sifda molto complicata dato che i budget di memoria per queste piattaforme è di 2-3 ordini di grandezza inferiore rispetto a quanto disponibile per i dispositivi mobili. Queste tipologie di reti si basano su tecniche avanzate di ottimizzazione e compressione del modello, al fine di ridurre drasticamente la complessità computazionale e la quantità di memoria richiesta; tali metodi comprendono la separazione delle operazioni di calcolo e memorizzazione e l'utilizzo della quantizzazione a bassa precisione per rispettare i vincoli di risorse del microcontrollore.

L'*inference engine* di MCUNet è un framework operativo progettato appositamente per massimizzare l'efficienza computazionale e ridurre al minimo l'accesso alla memoria. Invece di eseguire tutte le operazioni di calcolo e memorizzazione in modo sequenziale, MCUNet separa queste due fasi e le esegue in modo asincrono: durante la fase di calcolo, vengono eseguite le operazioni di convoluzione e moltiplicazione con i pesi del modello, che richiedono un'alta potenza computazionale ma un accesso limitato alla memoria; nella fase di memorizzazione, i risultati delle operazioni di calcolo vengono temporaneamente archiviati in una cache o in registri a bassa latenza.

Questo permette di ridurre al minimo l'accesso alla memoria principale, che è più lento e richiede più energia. Inoltre, la cache o i registri possono essere organizzati in modo ottimale per sfruttare le caratteristiche del modello e ridurre al minimo il trasferimento di dati tra la memoria principale e l'unità di calcolo. Grazie a questa separazione delle operazioni di calcolo e memorizzazione, MCUNet riesce a minimizzare l'*overhead* associato all'accesso alla memoria, massimizzando così l'efficienza computazionale.

Per quanto riguarda i limiti di memoria, questa architettura utilizza tecniche di quantizzazione a bassa precisione per ridurre la complessità del modello; tali metodi comportano la rappresentazione dei pesi e degli output intermedi del modello con una precisione inferiore rispetto allo standard di 32 bit. In particolare MCUNet utilizza la rappresentazione a 8 bit e ciò che permette di ridurre drasticamente lo spazio occupato dalla rete ed il costo computazionale necessario, dato che tali operazioni a precisione inferiore richiedono un dispendio di risorse significativamente inferiore. Questa riduzione della precisione può comportare una certa perdita di precisione nel modello, tuttavia spesso tale riduzione è accettabile e compensata dai notevoli vantaggi in termini di riduzione di consumo di risorse del modello.

Con queste accortezze MCUNet ottiene ottime performance di classificazione, in partcicolare raggiunge una *top-1 accuracy* del 70.7% valutata su ImageNet (un dataset contente oltre 1 milione di immagini divise tra 1000 classi, spesso preso come riferimento per le prestazioni di modelli di classificazione di immagini) sul microcontrollore STM32H743. Altre reti con caratteristiche simili avevano raggiunto risultati affini, MobileNetV2-0.75 ad esempio arriva ad un'accuratezza del 69.8%, tuttavia MCUNet lo fa riducendo l'utilizzo di memoria SRAM di un fattore 3.5× e l'utilizzo della Flash di un fattore 5.7× per adattarsi meglio ai dispositivi *tiny*.

Neural Architecture Search (NAS) e tinyNAS

La ricerca di architetture neurali ottimali per applicazioni specifiche è un ambito di studio noto come Neural Architecture Search (NAS). A differenza dei metodi tradizionali che richiedono un progetto manuale da parte di esperti, NAS utilizza algoritmi di ricerca automatica per esplorare un vasto spazio di possibili architetture, sfruttando tecniche di ottimizzazione e strategie di ricerca per individuare configurazioni neurali ottimali, consentendo di raggiungere risultati di elevata qualità in maniera più efficiente.

Il tinyNAS comprende invece metodi di ricerca di modelli di tinyML concentrandosi sul perfezionamento delle reti per adattarsi alle restrizioni di memoria e potenza computazionale dei dispositivi embedded. In particolare in MCUNet la fase di indagine inizia con la definizione di uno spazio di ricerca, che rappresenta l'insieme delle possibili configurazioni dell'architettura neurale; questo *search space* viene definito tramite due diversi paramteri: la dimensione delle immagini di input, che varia su 12 diverse risluzioni, ed il width multiplier, che assume 9 diversi valori compresi nell'intervallo

[0.1, 1]. In totale si ottengono quindi 108 diversi spazi di ricerca che vengono valutati tramite lo studio dei *FLOPs* (Floating point Operations Per Second) corrispondenti; questa analisi si basa infatti sull'assunzione che, per la stessa famiglia di modelli, le necessità computazionali siano positivamente correlate con le performance della rete.

1.3 Linear Probes

I modelli di Deep Learning, come le Convolutional Neural Networks, possono essere estremamente complessi, con milioni di parametri e strati di calcolo interconnessi. Questa complessità può rendere difficile per le persone comprendere come il modello giunge a un determinato risultato.

Per questo motivo uno dei trend crescenti nel campo del machine learning è quello dell'*interpretabilità* dei modelli; con ciò ci si riferisce appunto alla capacità di comprendere e spiegare il funzionamento e le decisioni prese da una rete neurale. Questo concetto di interpretabilità sta diventando sempre più rilevante nel campo della ricerca informatica sia per motivi etico/normativi (in alcuni settori come quello della sanità o della finanza infatti è richiesto che i modelli siano comprensibili e spiegabili per potersi conformare a regolamentazioni specifiche), sia per motivi di analisi ed ottmizzazione (il funzionamento completo e profondo di una rete spesso non è chiaro neanche agli sviluppatori, per questo motivo riuscire ad interpretare un modello può tornare utile per il suo miglioramento).

Una delle tecniche volte a questo tipo di studio è quella del *Linear Probing* [1] che consisite nell'analizzare il ruolo dei layer convolutivi intermedi all'interno di una CNN tramite dei classificatori lineari, anche detti *probes*, che vanno a valutare prestazioni di un modello a differenti profondità della rete. Le novità introdotte con questa metodologia hanno implicazioni sicuramente interessanti per quanto riguarda il tema dell'interpretabilità e di analisi delle performance dei modelli di ML, in particolare si nota come il livello di separabilità lineare delle feature in output dai vari strati convolutivi aumenta monotonicamente raggiugngendo layer più profondi della rete.

Funzionamento delle Linear Probes

Dato che l'obiettivo di questo tipo di analisi è lo studio delle performance di una determinato modello di convolutional neural network per la classificazione di immagini, è necessario essere già in possesso una rete neurale allenata a tale scopo. Tipicamente, a meno di dettagli implementativi, una CNN per l'image classification segue il funzionamento descritto nella sezione 1.1.1 ed è composta da diversi layer convolutivi, seguiti da operazioni di pooling, dediti all'estrazione delle feature e da uno strato classificatore finale, che raccoglie tutte le informazioni ricavate e restituisce una distribuzione di probabilità delle classi di appartenenza dell'immagine, come mostrato dallo schema in figura 1.4.

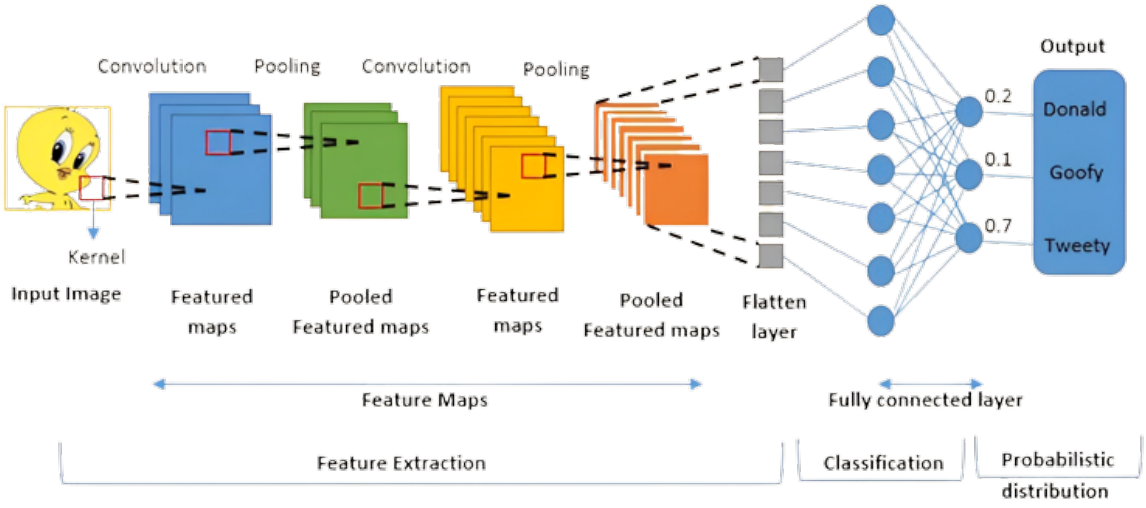

Figura 1.4: *Convolutional Neural Network per la classificazione di immagini.*

Il principio dietro alla tipologia di esame delle linear probes è quindi quello di posizionare in mezzo ai diversi strati della rete dei classificatori lineari (o probes), che hanno la stessa funzionalità di

quello locato al termine del modello, allenarli singolarmente con le sole informazioni estratte fino alla profondità in cui sono disposti ed analizzare le loro prestazioni di classificazione. Queste probes, oltre che per la loro disposizione, differiscono in dimensionalità, ogni strato convolutivo infatti restituisce una rappresentazione (o feature map) caratterizzata da un certo numero di canali e da una precisa dimensione che differsicono dagli output successivi; ogni probe deve essere quindi dimensionata in modo da poter ricevere in ingresso le feature in uscita dal layer ad essa precedente; ogni probe differirà dalle altre anche per i valori (ed il numero) dei pesi, ma avranno tutte la stessa dimensione di output, ovvero il numero di classi, dato che essendo classificatori vengono allenate con le etichette di classe delle immagini provenienti dallo stesso dataset di training della rete.

Analisi e valutazione

A seguito della fase di addestramento delle diverse probes si possono confrontare i risultati di classificazione per comprendere al meglio il funzionamento interno della rete. Per fare ciò la metrica più significativa da analizzare è la *loss* che descrive la discrepanza tra le predizioni effettuate dai classificatori e le reali etichette di classe delle immagini. Se il modello di CNN è stato architettato in maniera efficace ci si aspetta un andamento decrescente della metrica di perdita rispetto alla profonidtà del classificatore come illustrato in figura 1.5.

Questa tecnica del Linear Probing aiuta quindi a comprendere il comportamento di una rete neurale convoluzionale per la classificazione di immagini, mostrando come ogni layer della CNN contribuisce alle performance finali del modello, incremenentando il livello di separabilità lineare delle feature man mano che si raggiungono strati più profondi della rete. Tale metodo si dimostra inoltre molto utile per debuggare modelli durante le fasi di sviluppo o per avere concezione del progresso della fase di training in una rete ben strutturata.

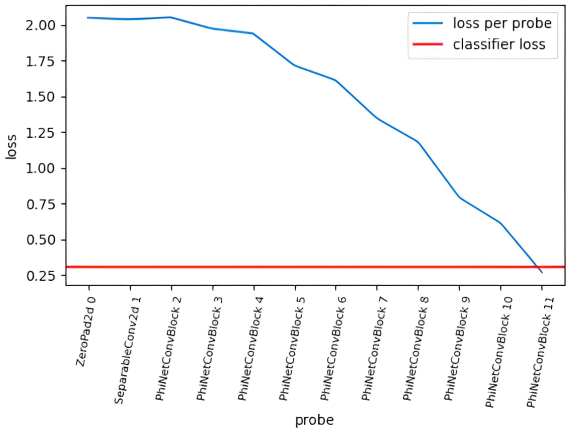

Figura 1.5: *Loss per probe*. Un valore inferiore della funzione di loss corrisponde ad una maggiore accurateza di classificazione.

1.4 Chinchilla scaling law

Nel complesso ambito delle architetture di machine e deep learning, i processi di addestramento e valutazione di reti neurali possono richiedere un notevole dispendio di risorse computazionali e temporali, soprattutto quando si considerano differenti configurazioni di un modello. Sorge quindi la necessità di capire se e come sia possibile stimare le performance di una rete senza il bisogno di allenarla, ma solamente conoscendo la sua complessità ed i metodi che si intendono utilizzare durante la fase di training.

A tal proposito nel marzo 2022 è stata proposta da un articolo di DeepMind [4] una legge di scalabilità che mira a legare le prestazioni di un modello di linguaggio naturale (quelli che vengono definiti Large Language Models, LLM, come ChatGPT) con le sue dimensioni e con la quantità di dati utilizzata durante la fase di addestramento. In particolare questa legge, che prende il nome di "Chinchilla scaling law" mette in relazione la metrica di *loss* con la dimensione del modello, in termini

di numero di parametri totali (N), e della quantità di dati di training, come numero di tokens (D), tramite una funzione iperbolica così definita:

$$Loss(N, D) = \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + E \tag{1.2}$$

con A e B che regolano rispettivamente il peso della complessità della rete e dei dati a dispozione sulla loss del modello ed E che misura la perdita per un processo generativo di testo ideale e dovrebbe corrispondere all'entropia del testo naturale.

È interessante notare come la configurazione di parametri (A, B, E, α, β) che meglio approssima l'andamento dei risultati attaulamente in possesso mostri che i ritorni derivanti dall'aumento delle dimensioni del modello sono scarsi, mentre quelli provenienti dall'aumento della quantità di dati sono elevati. Ciò sta a significare che, nel campo dei *transformers* (ovvero un tipo di architettura di rete neurale efficiente per il trattamento di dati sequenziali, come il caso del linguaggio naturale), al momento il limite principale per le prestazioni dei modelli di linguaggio è la quantità di dati disponibili e, qualora se ne disponga di una quantità sufficiente, non c'è motivo di addestrare modelli con enormi dimensioni (in questo ambito si parla di reti con oltre 500 miliardi di parametri dato che il compito è molto complesso).

Ovviamente le architetture dei transformers generativi per il linguaggio naturale hanno poco in comune con le reti neurali convoluzionali, sia per il compito che per la struttura di rete utilizzata, ma la *Chinchilla scaling law* offre buoni spunti per relazionare le performance di un modello di classificazione di immagini con la sua complessità.

2 Metodi

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

2.1 PhiNet

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

2.2 Regressione

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

2.3 Linear Probes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

3 Conclusioni

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Bibliografia

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2018.

[2] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[4] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[6] Alex Krizhevsky. Datasets: cifar-10 & cifar-100. https://www.cs.toronto.edu/∼kriz/cifar.html.

[7] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *ArXiv*, abs/2007.10319, 2020.

[8] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. Phinets: A scalable backbone for low-power ai at the edge. *ACM Trans. Embed. Comput. Syst.*, 2022.

[9] Shoaib Siddiqui, Ahmad Salman, Imran Malik, Faisal Shafait, Ajmal Mian, Mark Shortis, and Euan Harvey. Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75, 05 2017.

[10] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.

Allegato A Titolo primo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

A.1 Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

A.1.1 Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Allegato B Titolo secondo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

B.1 Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

B.1.1 Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.