



UNIVERSITÀ  
DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in  
Ingegneria Informatica, delle Comunicazioni ed Elettronica

ELABORATO FINALE

ESPLORAZIONE EMPIRICA DELLE SCELTE DI  
DESIGN PER RETI CONVOLUZIONALI CON  
RISORSE LIMITATE

Supervisore  
Prof. Elisa Ricci

Laureando  
Matteo Beltrami

Correlatori  
Francesco Paissan  
Elisabetta Farella, PhD

Anno accademico 2022/2023



# Ringraziamenti

*Al termine di questo lavoro, trovo essenziale utilizzare qualche riga per dare spazio a coloro che, senza la minima esitazione, hanno contribuito alla realizzazione del corrente elaborato. Innanzitutto, è doveroso ringraziare il mio supervisore Elisa Ricci, per la sua pazienza e disponibilità e per avermi accompagnato lungo questa importante fase del mio percorso accademico. Un sentito ringraziamento alla Fondazione Bruno Kessler per avermi dato l'opportunità di fare esperienza, in prima persona, in una grande e importante realtà di ricerca, permettendomi di sviluppare la curiosità e l'interesse che mi hanno successivamente condotto alla realizzazione di questo elaborato. Ringrazio inoltre i miei correlatori Francesco Paissan e la dott.ssa Elisabetta Farella, insieme a tutti i colleghi dell'ufficio E3DA, per i loro preziosi consigli e per avermi guidato all'interno della realtà FBK. Un solo grazie non sarebbe sufficiente per restituire alla mia famiglia tutto l'affetto e il supporto ricevuti nel corso di questo mio percorso di vita: grazie a papà, mamma e Tommaso per avermi permesso di arrivare fino a qui e per avere creduto in me fin dal primo momento. Ringrazio infine tutti gli amici ed i compagni di corso che, vivendo questa avventura con me, hanno dato un significato aggiunto al mio percorso accademico.*



# Indice

<b>Sommario</b>	<b>2</b>
0.1 Classificazione di immagini . . . . .	2
0.2 Tecniche di implementazione di tinyML . . . . .	3
0.3 Scaling di Convolutional Neural Networks . . . . .	4
<b>1 Lavori correlati</b>	<b>5</b>
1.1 Convolutional Neural Networks . . . . .	5
1.1.1 Funzionamento . . . . .	5
1.1.2 Dataset . . . . .	6
1.1.3 Training e ottimizzazione . . . . .	6
1.1.4 Valutazione delle prestazioni . . . . .	7
1.2 Architetture di deep learning per l'edge computing . . . . .	8
1.2.1 MobileNets . . . . .	8
1.2.2 EfficientNet ed il compound scaling . . . . .	9
1.2.3 MCUNet ed il tinyNAS . . . . .	11
1.3 Linear Probes . . . . .	12
1.4 Chinchilla scaling law . . . . .	13
<b>2 Metodi</b>	<b>15</b>
2.1 PhiNets . . . . .	15
2.1.1 Blocchi convoluzionali . . . . .	15
2.1.2 Fattori di scala . . . . .	15
2.2 Obiettivo di ricerca . . . . .	16
2.3 Raccolta dati . . . . .	17
2.4 Tecniche di regressione e segmentazione . . . . .	18
2.4.1 Metodi di regressione . . . . .	19
2.4.2 Metodi di segmentazione . . . . .	23
<b>3 Setup sperimentale</b>	<b>25</b>
<b>4 Risultati</b>	<b>26</b>
4.1 Analisi delle funzioni di regressione . . . . .	26
4.1.1 Analisi qualitativa . . . . .	26
4.1.2 Analisi quantitativa . . . . .	26
4.2 Smoothing . . . . .	27
4.3 Valutazione dell'efficacia del Linear Probing . . . . .	28
<b>5 Conclusioni</b>	<b>29</b>
<b>Bibliografia</b>	<b>29</b>

# 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 calcolo di queste piattaforme.

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 necessaria alcun tipo di connettività per l'inferenza, si possono elaborare i dati in tempo reale direttamente 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 notevolmente 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 riduce la risoluzione 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 supervisionato (o *supervised learning*) 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; le etichette, anche chiamate *label* rappresentano le classi di appartenenza delle immagini (come ad esempio "cane" o "automobile"). 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 costo, come ad esempio la *cross-entropy loss*. Una volta che la rete è stata addestrata, viene valutata utilizzando il set di validazione, composto da immagini nuove (non appartenenti al training set), 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 (tinyML), 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 e 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 indistintamente per diverse piattaforme con risorse limitate senza incorrere a 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* e il *neural architecture search*. Il *pruning* prevede di comprimere il modello già allenato in modo tale da ridurre le connessioni neurali eliminando 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/numero di filtri) in modo da adattarla ai vincoli computazionali imposti cercando di preservare l'accuratezza di predizione; il *neural architecture search* invece è un nuovo ambito di ricerca 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 la tecnica più utilizzata per implementare algoritmi efficienti di machine learning a basso consumo 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 intrinsecamente 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.

Tramite l’aggiustamento dei fattori di scala è quindi 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 computazionali ma con accuratezza diversa, da qui nasce quindi il mio obiettivo di ricerca.

Pertanto, ottimizzare una rete neurale con un determinato budget richiede di risolvere un problema di ottimizzazione degli iperparametri di scaling. Questo è descritto come: 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 [9], 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 epoch per l’addestramento, ottimizzatore utilizzato durante il training, valore del learning rate) e definire un sottodomino 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 separabilità lineare degli output di ogni strato (Linear Probing) [1], fino allo sviluppo di modelli di regressione per l’individuazione dei fattori di scala migliori per determinati budget di risorse.

Attraverso questa analisi, proponiamo un framework di lavoro applicabile a diverse architetture per il tinyML. Inoltre analizziamo l’efficacia del Linear Probing per studiare le performance di una rete e individuare il 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 duemila, il numero di strati ottimale risulta essere 4. Nel range di parametri compreso tra duemila e diecimila il numero di strati ideale diventa 5, mentre fino a novecentomila 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.

# 1 Lavori correlati

## 1.1 Convolutional Neural Networks

Le Convolutional Neural Networks (CNN) [15] 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 astratte 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 risoluzione spaziale delle raffigurazioni intermedie ai vari strati.

Durante la convoluzione, il filtro, solitamente di dimensione  $3 \times 3$  o  $5 \times 5$ , 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 operazioni e migliorando quindi 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. [12]

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 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 lo *split* più grande 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 circa il 10% e si utilizzano per validare la capacità predittiva e per controllare 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.

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 sessantamila immagini a colori etichettate di dimensione  $32 \times 32$  pixel che consistono rispettivamente di 10 classi da seimila immagini ognuna (*CIFAR-10*) e 100 classi da seicento immagini ciascuna (*CIFAR-100*), suddivise in set secondo le proporzioni 70%-15%-15%, principalmente raffiguranti esseri viventi, mezzi di trasporto e oggetti vari [7].

### 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* e 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 e i pesi del kernel, i risultati di queste molteplicazioni vengono poi sommati e 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 e i risultati attesi; questo processo prende perciò il nome di *training*.

L'inizializzazione dei parametri consiste quindi nell'assegnare dei pesi iniziali 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 apprendimento in base alle funzioni di attivazione utilizzate all'interno della rete. Ne è un esempio la

tecnica di inizializzazione dei pesi proposta da Kaiming He, ottimizzata per la funzione ReLU, per la quale ogni peso viene inizializzato 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. [4]

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 training 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 per il task di classificazione è la Cross-Entropy Loss. L'obiettivo è quello di minimizzare 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 *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. 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 ottimizzazione andrebbero ripetuti numerose volte) e potrebbe compromettere la stabilità 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 problemi 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 *epoch* in modo da permettere alla rete di apprendere dai dati in modo progressivo e poter convergere a una soluzione ottimale.

#### 1.1.4 Valutazione delle prestazioni

Dopo aver compreso le tecniche più utilizzate per addestrare i modelli di classificazione 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 epoch 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 classificazione, 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 epoch differenti è necessario perchè non sempre addestrare un modello per più epoch 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. [6, 13, 8]

### 1.2.1 MobileNets

MobileNets [6] 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 sulle operazioni di convoluzioni depthwise separabili, che separano 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 di convoluzione in due operazioni differenti (convoluzione *depthwise* e *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 di operazioni; successivamente, la convoluzione pointwise combina linearmente i canali di output della convoluzione depthwise, generando un set di rappresentazioni finali (feature maps).

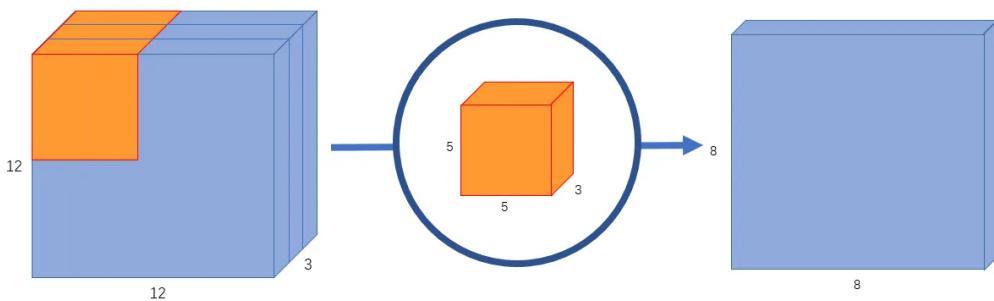


Figura 1.1: *Convoluzione tradizionale*. In questo caso l'immagine originaria ha risoluzione  $12 \times 12$  ed è composta da 3 canali (come nel caso di una foto RGB). Un filtro convolutivo completo (di dimensioni spaziali  $K \times K$ ) si estende su tutti i canali dell'immagine e genera in uscita una rappresentazione a un singolo canale con risoluzione minore. Se si volesse avere come output una feature map composta da  $C_{out}$  canali (come spesso succede nelle CNN), bisognerebbe quindi avere  $C_{out}$  filtri convolutivi ed eseguire  $C_{out}$  convoluzioni. Nel caso qui rappresentato l'intero processo richiederebbe  $4800 \cdot C_{out}$  moltiplicazioni e  $3 \cdot K^2 \cdot C_{out}$  parametri totali.

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 a ogni layer ( *$\alpha$  width multiplier*) e la risoluzione in ingresso delle immagini ( *$\rho$  resolution multiplier*).

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

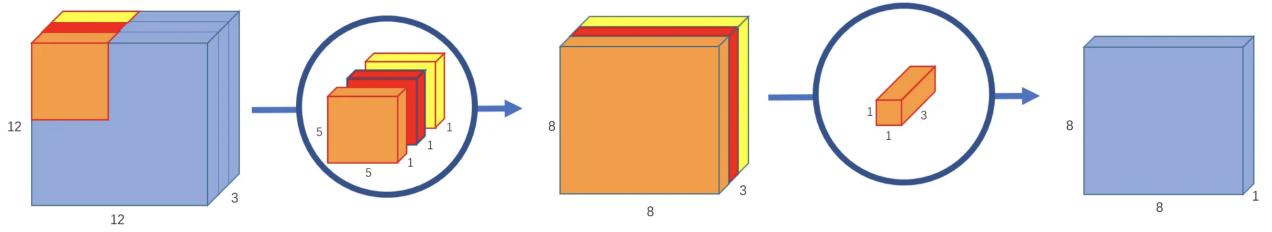


Figura 1.2: *Convoluzione depthwise separabile*. La stessa immagine di partenza viene prima fatta passare attraverso una convoluzione sui canali (depthwise) e si ottiene una rappresentazione intermedia ancora divisa secondo il numero di canali originario; per ottenere in output una rappresentazione ad un solo canale si esegue la convoluzione spaziale sui pixel dell’immagine (pointwise). Al fine di estrarre una feature map composta da  $C_{out}$  canali, bisognerebbe soltanto avere  $C_{out}$  filtri convolutivi pointwise ed eseguire una convoluzione pointwise con  $C_{out}$  filtri, che sono più leggere ed efficienti. Nel caso qui rappresentato l’intero processo richiederebbe un totale di  $4800 + 192 \cdot C_{out}$  moltiplicazioni e  $3 \cdot K^2 + 3 \cdot C_{out}$  parametri totali, che nei casi di applicazione reali si traduce in un guadagno enorme nell’efficienza della rete.

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  $\rho$  appartenenti all’intervallo  $(0, 1]$  il costo computazionale si riduce di  $\rho^2$  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  $\alpha$  nell’intervallo  $(0, 1]$  si va a modificare il numero di canali in ingresso  $C_{in}$  ed in uscita  $C_{out}$  da ogni layer convolutivo rispettivamente in  $\alpha C_{in}$  e  $\alpha C_{out}$ . Il *width multiplier* ha quindi l’effetto di ridurre il costo delle convoluzioni depthwise separabili ed il numero di parametri della rete di  $\alpha^2$ , 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  $\alpha$  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 [13].

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 [3] possono essere scalate modificando il numero di strati della rete, mentre le MobileNets possono essere ridimensionate 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 gli stessi autori dell’architettura mostrano come il bilanciamento congiunto di tali fattori sia cruciale per ottenere buone performance [13].

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 le dimensioni.

#### Compound scaling

Ragionando puramente a livello intuitivo, il metodo del compound scaling risulta efficace perché 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 quantifica empiricamente la relazione tra le tre dimensioni del modello.

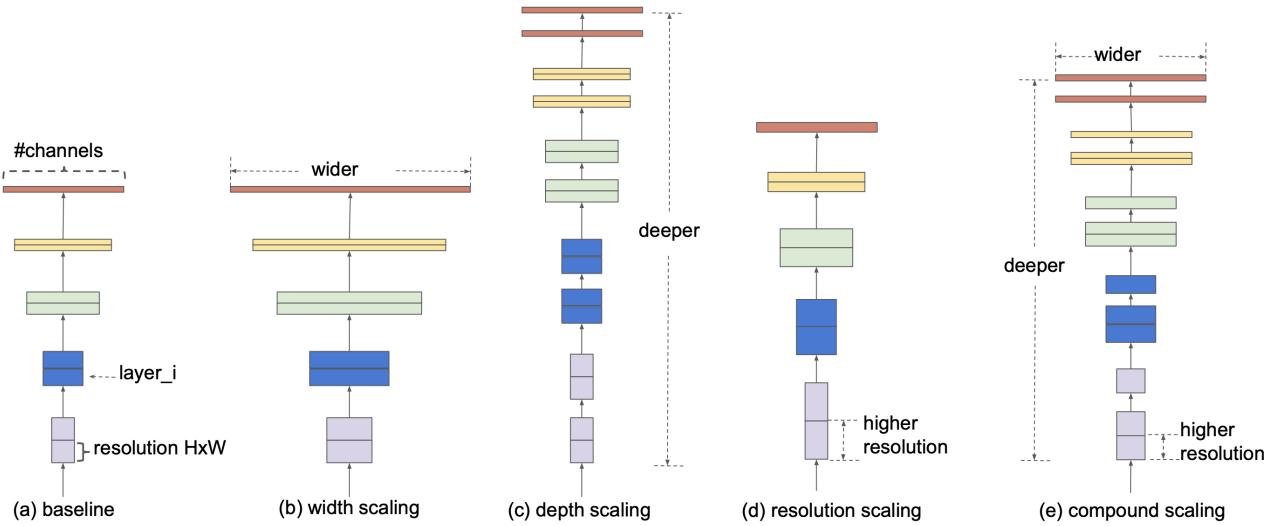


Figura 1.3: *Scaling delle Convolutional Neural Networks*. (a) è la rete di riferimento; (b)-(d) mostrano i metodi convenzionali di scaling in cui viene scalata solo una dimensione tra larghezza, profondità e risoluzione dell'immagine; (e) mostra il metodo di compound scaling che ridimensiona uniformemente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

- $\alpha$ : regola la profondità (depth) della rete cambiando il numero di layer presenti in modo da poter catturare rappresentazioni più complesse;
- $\beta$ : 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;
- $\gamma$ : 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 compound scaling è l'introduzione del parametro  $\phi$  e l'impostazioni degli iperparametri in modo che:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \quad \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \quad (1.1)$$

Le combinazioni di questi fattori sono potenzialmente infinite ma possono essere facilmente determinati con una griglia di ricerca basilare; il coefficiente di composizione  $\phi$  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 (linearmente correlato al consumo di potenza) è proporzionale a  $\alpha$ ,  $\beta^2$ ,  $\gamma^2$  del modello e che è stata impostata la relazione 1.1 tra gli iperparametri, ne consegue che scegliendo arbitrariamente un valore di  $\phi$ , il numero di operazioni viene incrementato approssimativamente di  $2^\phi$ .

L'effetto combinato di questi fattori di scala permette quindi a 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 sfida 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 particolare raggiunge una *top-1 accuracy* del 70.7% valutata su ImageNet (un dataset contenente 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 a 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 parametri: la dimensione delle immagini di input, che varia su 12 diverse risoluzioni, 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, cosa generalmente valida [13].

### 1.3 Linear Probes

I modelli di Deep Learning, come le Convolutional Neural Networks, possono essere estremamente complessi, con milioni di parametri e layer 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 e ottimizzazione (il funzionamento completo e profondo di una rete spesso non è chiaro neanche agli sviluppatori, per questo motivo riuscire a 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 consiste 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 raggiungendo 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 di 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.

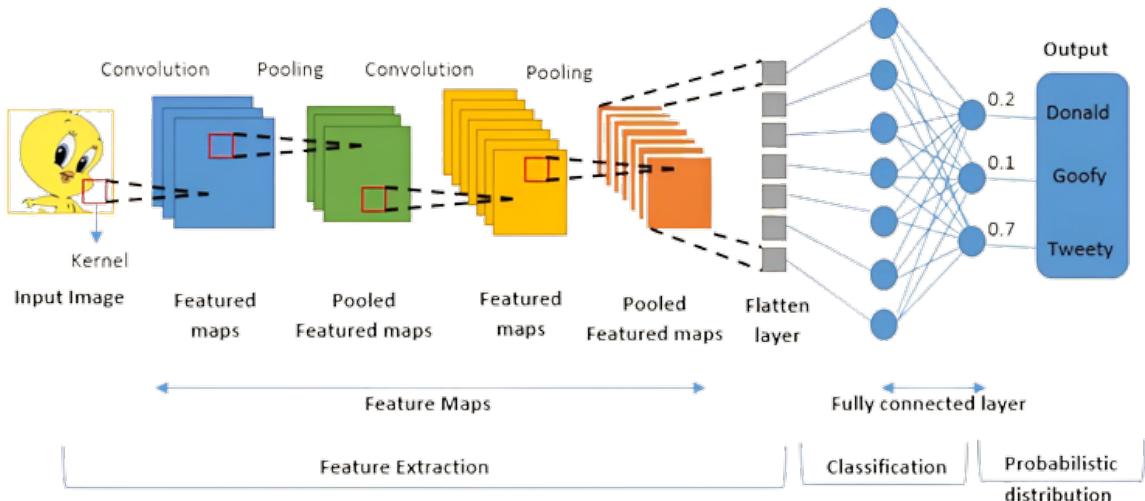


Figura 1.4: Esempio di una Convolutional Neural Network per la classificazione di immagini.

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 (chiamati *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 differiscono 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 (e 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 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 profondità 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, incrementando 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.

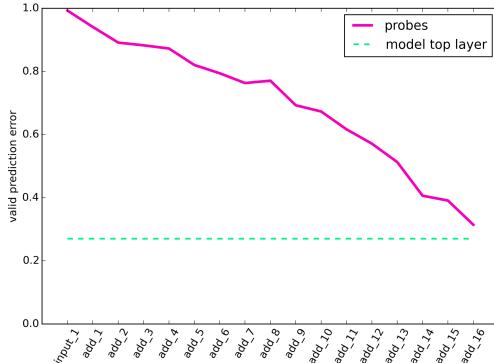


Figura 1.5: *Linear Probes*. Sull’asse delle ascisse si trovano le linear probes posizionate dopo ogni strato convolutivo di una rete (in questo caso una ResNet), sull’asse delle ordinate è presente il valore della loss corrispondente ad ogni probe. Un valore inferiore della funzione di loss corrisponde a una maggiore accuratezza 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 [5] 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 ( $P$ ), e della quantità di dati di training, come numero di tokens ( $D$ ), tramite una funzione iperbolica così definita:

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

con  $A$  e  $B$  che regolano rispettivamente il peso della complessità della rete e dei dati a disposizione 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.

Ovviamente le architetture dei *transformers* generativi per il linguaggio naturale (ovvero un tipo di rete neurale efficiente per il trattamento di dati sequenziali, come in questo caso di applicazione) 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.

In particolare durante l'analisi successivamente proposta verrà utilizzata una forma funzionale simile a quella proposta nella formula 1.2, che lega l'accuratezza di classificazione di un modello con il numero di parametri che lo compongono tramite la relazione 1.3.

$$Accuracy(P) = \frac{A}{P^\alpha} + C \quad (1.3)$$

In questa situazione il fattore che esprime la quantità di dati di training viene semplificato nella costante  $C$  dato che negli esperimenti considerati le condizioni riguardanti il dataset rimangono invariate.

## 2 Metodi

In questo capitolo verrà presentata l'architettura di rete utilizzata per lo scopo di ricerca e verranno discusse criticamente le tecniche di acquisizione, regressione e segmentazione dei dati utilizzate per condurre l'analisi.

### 2.1 PhiNets

L'architettura delle PhiNets consiste in una famiglia di reti scalabile e ottimizzata per l'elaborazione delle immagini basata sul deep learning per piattaforme con risorse limitate. [9]

In particolare per i diversi compiti di classificazione di immagini, rilevamento e tracking in tempo reale di oggetti, le reti convoluzionali di questa famiglia si distinguono per la struttura dell'ultimo layer (classificatore o detection head), mantenendo però in comune il *backbone*, ovvero l'insieme degli strati convolutivi atti all'estrazione delle feature. Per rendere tali modelli scalabili, è fondamentale ottimizzare il backbone della rete poiché esso è responsabile della maggior parte del consumo di risorse durante l'inferenza del modello. Le tecniche utilizzate a questo scopo riguardano principalmente lo sfruttamento di blocchi convoluzionali efficienti e l'utilizzo di un'architettura modificabile tramite degli opportuni fattori di scala.

#### 2.1.1 Blocchi convoluzionali

La famiglia di reti delle PhiNets è particolarmente ottimizzata a livello computazionale grazie a delle specifiche operazioni di convoluzione utilizzate in ogni layer, vengono infatti impiegati gli *inverted residual blocks*, ovvero dei blocchi convoluzionali efficienti introdotti in MobileNetV2. [11]

Per poter apprezzare i benefici portati dall'introduzione di questo tipo di operazioni, bisogna prima comprendere il funzionamento dei *residual blocks* tradizionali, implementati inizialmente nell'architettura delle ResNet [3]. Questi blocchi residuali prendono il nome dall'operazione di *skip connection* che lega l'input e l'output dell'operazione convolutiva, la quale propaga dei residui della rappresentazione in entrata. Il funzionamento basilare prevede infatti la riduzione del numero di canali della feature map in ingresso al blocco tramite un'operazione convolutiva non dispendiosa, la successiva espansione del numero di canali (riportati al valore iniziale) tipicamente tramite una *depthwise separable convolution* (sezione 1.2.1) e l'applicazione della skip connection, che lascia dei residui della rappresentazione in input nella feature map in output. Questo tipo di blocco residuale riduce fortemente il numero di parametri ed il numero di operazioni effettuate dalla rete ma ne degrada anche le prestazioni.

Proprio al fine di risolvere il problema di deterioramento delle performance di queste operazioni, sono stati introdotti gli *inverted residual blocks* che mirano a bilanciare le prestazioni di rete rinunciando ad una parte del guadagno sul consumo di risorse. La procedura di funzionamento è molto simile a quella dei blocchi residuali tradizionali con la differenza che il numero di canali della rappresentazione in entrata viene inizialmente espanso, per poter estrarre informazioni più dettagliate, per poi essere compresso alla fine del blocco allo scopo di ottenere in uscita una rappresentazione con la stessa dimensionalità di quella in ingresso. A questa struttura di base del blocco residuale invertito, l'architettura delle PhiNets aggiunge un'operazione denominata *squeeze and excitation* prima della compressione finale del numero di canali; questa funzione aiuta ad enfatizzare le informazioni presenti nei canali del tensore di input per poi effettuare la compressione.

#### 2.1.2 Fattori di scala

Come ampiamente anticipato, le architetture progettate per l'inferenza di modelli di machine learning su dispositivi con capacità limitate adottano l'utilizzo di determinati fattori di scala, anche detti *iperparametri*, per adattare la struttura e il consumo di risorse della rete alla specifica piattaforma utilizzata.

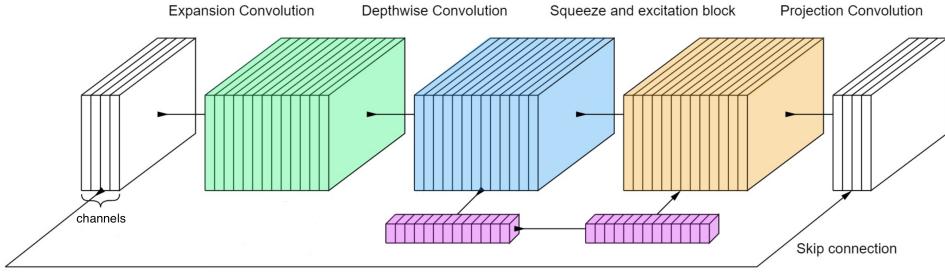


Figura 2.1: *Blocco convoluzionale delle PhiNets.* Il numero di canali viene innanzitutto aumentato con una convoluzione pointwise, seguita da una convoluzione depthwise e un blocco di *squeeze and excitation*. Infine, una seconda convoluzione pointwise si collega al blocco di output a bassa dimensionalità.

In particolare la famiglia delle PhiNets utilizza 4 differenti iperparametri che modificano l’architettura della rete:

- $\alpha$ : detto *width factor*, regola la larghezza della rete cambiando il numero di canali in uscita da ogni layer;
- $\beta$ : detto *shape factor*, modifica la risoluzione spaziale delle immagini in ingresso a ogni strato convolutivo;
- $t_0$ : detto *base expansion factor*, regola il numero di canali delle rappresentazioni espanso durante l’esecuzione ogni blocco residuale invertito, in particolare il fattore di espansione varia linearmente con la profondità della rete seguendo la legge  $t = t_0 \left( \frac{(\beta-1)n+N}{N} \right)$ , in cui  $n$  è l’indice del blocco convolutivo a cui si fa riferimento;
- $N$ : è il numero di layers totale della rete e ne modifica quindi la profondità.

La variazione di ogni singolo parametro ha effetti differenti sulla struttura del modello e sull’impiego di risorse; nello specifico sono metriche di interesse il numero di operazioni per l’inferenza, che determina il consumo di potenza del dispositivo, e il numero totale di parametri della rete, che ne stabilisce l’occupazione di memoria.

Il numero di operazioni totali per l’esecuzione del modello dipende dalla risoluzione  $w \times h$  dell’immagine in ingresso, dalla profondità della rete ( $N$ ) e dalla sua larghezza, che scala quadraticamente rispetto al fattore  $\alpha$ . Il numero di parametri è invece determinato dalla dimensione dei kernel dei blocchi convolutivi della rete, che aumenta linearmente con la sua profondità  $N$  e col fattore  $\beta$ .

## 2.2 Obiettivo di ricerca

Tra le difficoltà relative l’utilizzo di architetture flessibili su determinate piattaforme è di particolare rilevanza quella di individuare l’insieme dei fattori di scala che definiscono una rete compatibile con le risorse del dispositivo sulla quale si vuole operare. Parlando quindi delle PhiNets, ci sono molteplici combinazioni degli iperparametri  $\alpha$ ,  $\beta$ ,  $t_0$ ,  $N$  che soddisfano un determinato vincolo di capacità di memoria e che sarebbero quindi adatte all’utilizzo, tuttavia ognuna di queste configurazioni avrà prestazioni di classificazione differenti.

Tale ricerca ha quindi lo scopo di fornire delle logiche per l’individuazione dell’insieme dei fattori di scala che definiscono una PhiNet con le performance migliori per un determinato budget di risorse computazionali.

Nello specifico con “risorse computazionali” si intendono l’occupazione della memoria FLASH del modello, che è definita dal numero di parametri totali della rete, l’utilizzo della memoria RAM, che nel caso delle PhiNet è determinata dalla dimensione dei tensori nelle operazioni di espansione del primo blocco convoluzionale, e il consumo di energia, che dipende dal numero di operazioni necessarie per eseguire un’inferenza del modello.

Nel corso di questa analisi assume particolare importanza lo studio del numero di layer  $N$  della rete che massimizza l’accuratezza di classificazione per determinati intervalli di spazio di archiviazione

disponibile; questo è dovuto al fatto che, una volta trovato tale valore per un certo budget di memoria FLASH, rimangono da regolare 3 diversi iperparametri con a disposizione 3 vincoli (FLASH, RAM, operazioni) e ciò è facilmente risolvibile tramite un sistema a triplice incognita.

Per poter condurre questo tipo di ricerca sarebbe conveniente uno strumento che consenta di determinare l'accuratezza di classificazione di una rete convoluzionale, in particolare di una PhiNet, conoscendo le caratteristiche del modello, definite dagli iperparametri scelti. Dato che le variabili in gioco sono molto complesse e che non è possibile fornire un risultato esatto di tale metrica, diventa utile lo sviluppo di modelli di regressione che approssimino con precisione l'accuratezza di classificazione di una rete.

## 2.3 Raccolta dati

A questo scopo si rende necessaria la raccolta di una quantità considerevole di dati riguardanti le performance rispetto alle caratteristiche del modello, in particolare per poter individuare una tendenza dei dati riguardanti il numero di layer dell'architettura è rilevante osservare l'andamento dell'accuratezza di classificazione rispetto al numero di parametri totali delle reti.

Questi dati sono ottenibili solamente tramite l'addestramento di molteplici PhiNets con diverse caratteristiche e l'estrazione del numero totale di parametri e dell'accuratezza di classificazione sul dataset di test per ottenere le reali prestazioni. Naturalmente, per poter rendere l'analisi il più efficace possibile, le configurazioni di rete generate dovrebbero coprire uniformemente l'intervallo di dati (numero di parametri) di interesse, definito a priori.

Per le piattaforme embedded più popolari la quantità di memoria FLASH disponibile è molto limitata, si parla infatti di uno spazio di archiviazione nell'ordine di grandezza dei KB o MB. Per esempio il dispositivo *Arduino UNO ATmega328P* dispone di 32KB di memoria FLASH, la scheda *UC32* ne possiede 512KB, *STM32L476* ne ha disponibile 1MB, *STM32H7* e *MSP432P411* hanno uno spazio di archiviazione di 2MB, mentre dispositivi come il *Raspberry Pi 3 B+* concedono un flessibilità maggiore dato che dispongono di un ingresso per una scheda SD esterna di parecchi MB.

Considerando quindi che ogni parametro di una PhiNet implementata per piattaforme embedded occupa 8 bit di memoria, è immediato osservare che il possibile intervallo di interesse, che copre anche i casi limite, spazi da  $10^4$  a  $10^7$  parametri di rete.

Dato che si tratta di un range molto ampio e data la natura esponenziale degli ordini di grandezza nel mondo informatico, avrebbe poco senso una distribuzione di dati linearmente omogenea, assume invece una grande rilevanza una disposizione ed una visualizzazione dei dati in scala logaritmica, come mostrato in figura 2.2.

L'obiettivo principale diventa quindi l'inizializzazione e l'addestramento di PhiNets che siano adeguatamente distribuite all'interno di questo range logaritmico e per fare ciò è necessario definire uno spazio di ricerca degli iperparametri che soddisfi queste condizioni.

Le reti addestrate sono oltre 250, il loro numero di parametri copre in maniera omogenea l'intervallo scelto e per ottenerle sono state selezionate diverse combinazioni degli iperparametri  $\alpha$ ,  $\beta$ ,  $t_0$ ,  $N$  in maniera tale che il valore di ogni fattore fosse ragionevole e adatto al contesto di utilizzo di questo tipo di reti e che i modelli generati mantenessero una determinata coerenza di struttura. In particolare il fattore di larghezza  $\alpha$  può assumere 5 differenti valori rappresentati in questo insieme  $[0.15, 0.3, 0.5, 0.9, 1.5]$ , il fattore di forma  $\beta$  varia tra  $[0.75, 1.05, 1.35]$ , il fattore di espansione di base  $t_0$  viene scelto tra  $[4, 5, 6]$ , mentre il numero di strati convolutivi  $N$  può assumere 6 valori, ovvero  $[4, 5, 6, 7, 8, 9]$ .

Tramite le varie permutazioni lungo questo spazio di iperparametri si ottengono quindi configurazioni di PhiNets differenti che sono caratterizzate, oltre che da una diversa struttura di rete, da un numero di parametri totali differente e da una diversa accuratezza nella classificazione di immagini.

È importante notare come i dati presentati, ottenuti tramite l'utilizzo degli iperparametri sopra citati, siano omogeneamente distribuiti lungo la scala logaritmica, sottolineando ancora di più la natura esponenziale delle architetture scalabili e l'importanza di visualizzare tali dati nella scala logaritmica durante l'analisi.

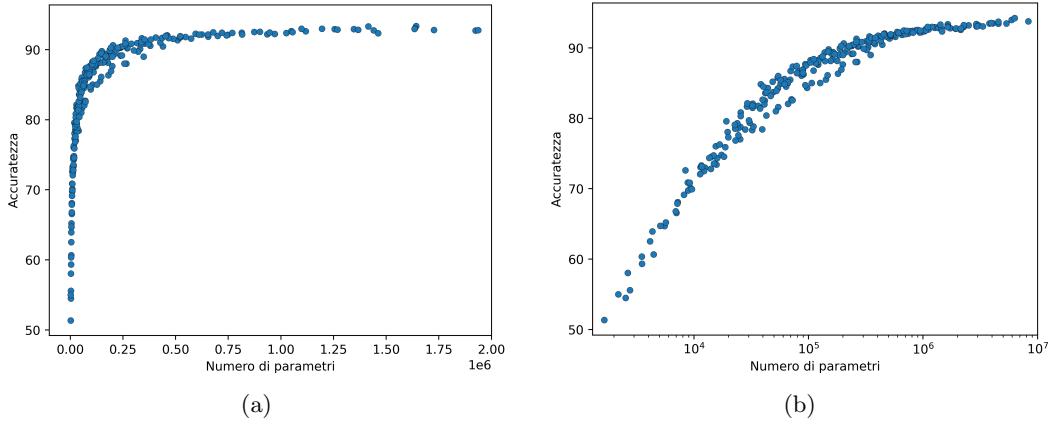


Figura 2.2: *Distribuzione dei dati per l’analisi.* Le due immagini mostrano entrambe la stessa distribuzione di dati, ovvero l’accuratezza di classificazione rispetto al numero di parametri di ogni PhiNet. La differenza risiede nella scala, la figura (a) visualizza i dati in scala lineare, (b) li mostra invece in scala logaritmica per una rappresentazione più chiara.

## 2.4 Tecniche di regressione e segmentazione

Come anticipato nella sezione 2.2, l’iperparametro delle PhiNets più significativo per lo scopo di ricerca è il numero di strati convolutivi  $N$ , dato che non sono disponibili logiche o procedure per la scelta ottimale di tale fattore al fine di massimizzare le prestazioni della rete.

A tal proposito è conveniente visualizzare i dati non solo rispetto al numero complessivo dei parametri, ma anche relativamente al numero di layer presenti nel modello al fine di poter scomporre l’analisi per avere dati tra loro più coerenti e facili da visualizzare.

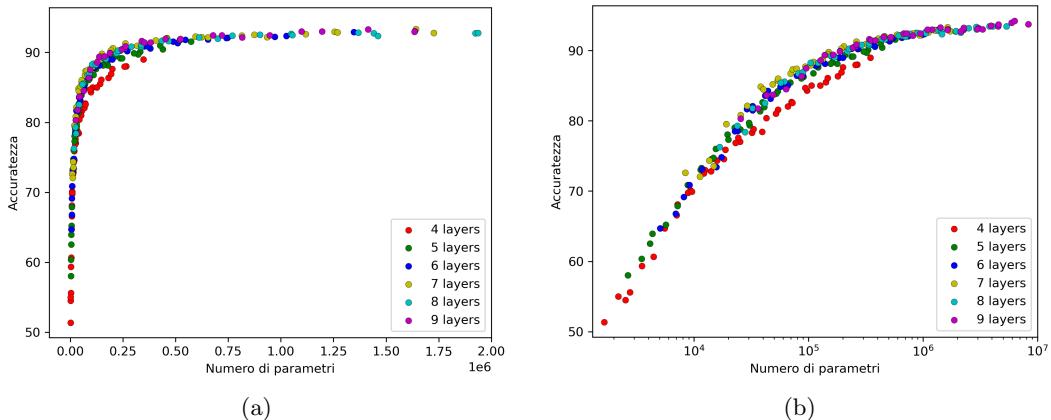


Figura 2.3: *Distribuzione dei dati in base al numero di strati.* I due grafici mostrano gli stessi dati del grafico 2.2, ma è rappresentata anche la dimensione rispetto al numero di layer della rete.

Come si può osservare qualitativamente in figura 2.3, determinati valori per il fattore  $N$  sono spesso da escludere dalla scelta per un determinato budget di memoria FLASH, mentre altri sono da preferire per capacità di numero di parametri più alto. L’intenzione è quindi quella di determinare quantitativamente il numero di strati convoluzionali che ottimizza le performance per ogni possibile valore di capacità di memoria e per fare ciò vengono utilizzate tecniche di regressione e segmentazione dei dati.

I metodi di regressione consentono di approssimare l’andamento di una distribuzione di dati con quello di una funzione e il loro scopo è quello di trovare un modello matematico che descriva al meglio la relazione tra una variabile indipendente (in questo caso il numero di parametri) e una variabile dipendente (in questo caso l’accuratezza di classificazione). Tale processo di regressione coinvolge, oltre la raccolta dei dati, la scelta del modello matematico appropriato, l’adattamento ai dati di addestramento e la valutazione delle prestazioni finali tramite determinate metriche, come ad esempio

l'errore quadratico medio (MSE). Le tecniche di segmentazione vengono invece utilizzate per dividere un insieme iniziale di dati in gruppi o segmenti omogenei con determinate caratteristiche, al fine di identificare pattern o tendenze all'interno dei dati.

In questa analisi verranno quindi impiegati sistemi di regressione per approssimare l'andamento dei risultati di classificazione rispetto al numero di parametri per ogni valore differente di numero di strati convolutivi presenti in modo da ottenere un modello matematico adeguatamente preciso per predire e confrontare le prestazioni delle reti; successivamente verranno utilizzati i dati ottenuti con la regressione per segmentare l'intervallo delle ascisse in modo da ottenere determinati range di memoria all'interno dei quali è preferibile scegliere un certo valore del fattore di profondità  $N$ .

### 2.4.1 Metodi di regressione

Per trovare la funzione matematica che meglio approssima la distribuzione di dati raccolti, bisogna innanzitutto definire il tipo di andamento che essa deve assumere (può essere lineare, logaritmico, esponenziale ecc.) in funzione di determinati parametri che definiscono i gradi di libertà del modello. Questi parametri sono le incognite che l'algoritmo di ottimizzazione deve trovare tramite il loro aggiustamento, in modo tale da ottenere una bassa discrepanza tra funzione matematica finale ed i dati effettivi disponibili.

Al fine di ottenere il miglior modello di regressione è necessario provare diversi andamenti e trovare il migliore tramite delle metriche di valutazione dell'errore totale.

## Regressione esponenziale

Osservando la distribuzione dei dati mostrata in figura 2.3 (a) è significativo notare all'interno del grafico la presenza di un asintoto orizzontale verso cui tendono gli andamenti per i diversi valori di  $N$ . Ciò significa che aumentando il numero di parametri della rete ci si aspetta un'accuratezza di classificazione più elevata che tende al valore stabilito dall'asintoto.

Diventa quindi logico supporre che il modello matematico scelto per approssimare l'andamento corrisponda a una funzione contenente un asintoto orizzontale e una possibile è la funzione esponenziale, in questa situazione modellata con coefficiente negativo secondo la legge:

$$acc = -a^{P-b} + c \quad (2.1)$$

In una funzione matematica di questo tipo  $acc$  si riferisce all'accuratezza di classificazione,  $P$  è il numero totale di parametri della rete e i tre fattori  $a$ ,  $b$ ,  $c$  regolano la funzione per adattarsi al meglio alla distribuzione dei dati da modellare. Al crescere del parametro  $a$  aumenta la ripidità della curva, mentre i fattori  $b$  e  $c$  ne controllano la traslazione rispettivamente lungo l'asse delle ascisse e delle ordinate, nello specifico  $c$  regola il livello dell'asintoto orizzontale di accuratezza.

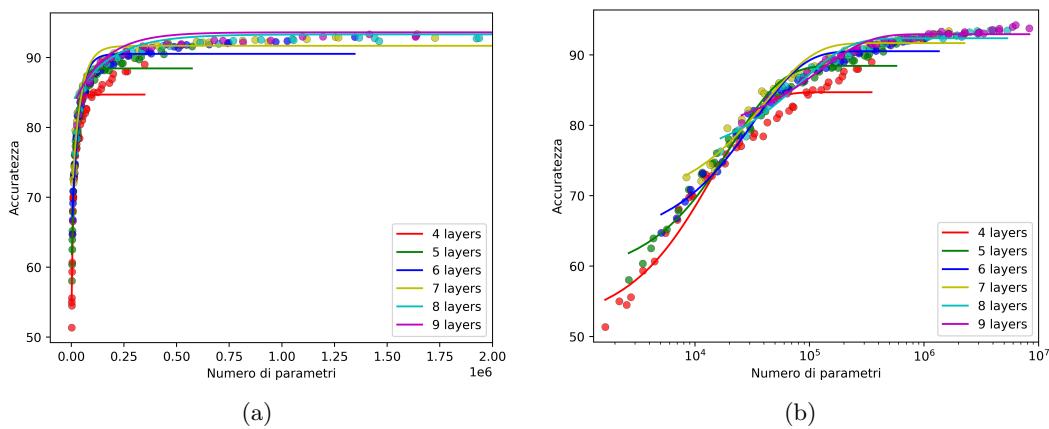


Figura 2.4: *Regressione esponenziale.*

Come si può notare un modello di regressione esponenziale non è adatto a questo tipo di distribuzione dati in quanto la tendenza della funzione ad appiattirsi sul suo asintoto è troppo marcata e tende ad approssimare meglio i valori intermedi trascurando invece quelli agli estremi dell'intervallo,

creando problemi di generalizzazione, infatti per valori elevati di numero di parametri la regressione tende ad approssimare al ribasso l'accuratezza di classificazione facendola tendere al valore  $c$ . Nella funzione esponenziale è inoltre assente un asintoto verticale che potrebbe rappresentare una possibile soluzione al problema.

## Regressione logaritmica

Tra le funzioni che possiedono un asintoto verticale con un andamento monotono decrescente, come nel caso desiderato, è possibile trovare le funzioni logaritmiche che tendono a crescere velocemente nella prima parte della funzione, mantenendo però un incremento significativo per elevati valori dell'asse delle ascisse.

Un possibile modello che possa quindi ricreare l'andamento delle curve logaritmiche è definito dalla relazione:

$$acc = a \cdot \log(P - b) + c \quad (2.2)$$

In questa funzione il parametro  $a$  regola la rapidità di crescita della curva, i fattori  $b$  e  $c$  controllano invece le traslazioni dei modelli lungo gli assi cartesiani, in particolare  $b$  rappresenta il valore delle ascisse degli asintoti verticali per le diverse configurazioni di layer.

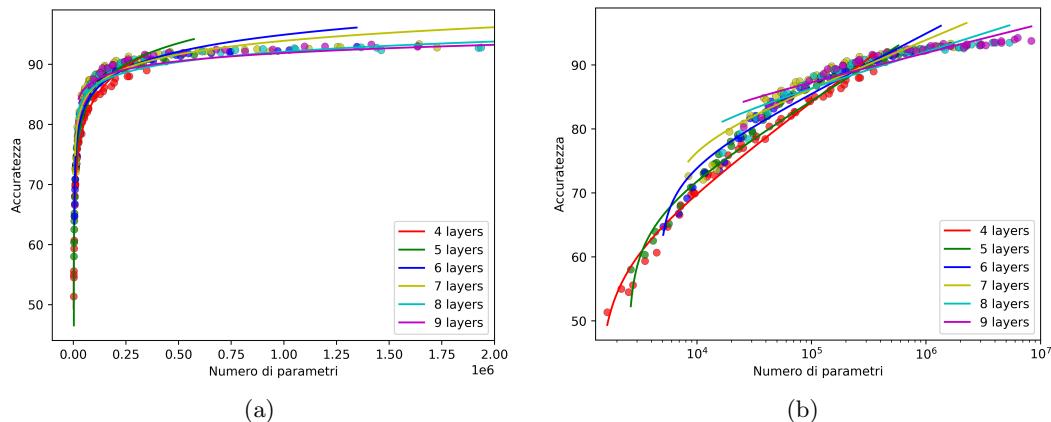


Figura 2.5: *Regressione logaritmica*.

Anche un modello di questa tipologia (2.5) non sembra l'ideale per approssimare al meglio la distribuzione di dati a disposizione. Sebbene l'introduzione di un asintoto verticale aiuti ad adattare il modello per valori bassi di parametri, rimane un grande problema di generalizzazione per valori elevati di occupazione di memoria. Questo è dovuto al fatto che la funzione logaritmica è di natura monotona crescente e di conseguenza non converge ad un valore sulle ordinate come accade con modelli che dispongono di un asintoto orizzontale; ciò ha l'effetto di sovrastimare le prestazioni di una rete con un numero elevato di parametri.

## Combinazione convessa tra funzione logaritmica ed esponenziale

Entrambe le regressioni, esponenziale e logaritmica, tendono ad approssimare bene l'andamento dei dati nella parte iniziale delle curve mentre faticano ad adattarsi adeguatamente ai risultati per elevati valori di numero di parametri, e quindi a generalizzare i risultati in un range diverso da quello osservato. In particolare le funzioni esponenziali tendono a sottostimare l'accuratezza per una rete con un elevato numero di parametri, invece le funzioni logaritmiche tendono a sovrastimarla. Diventa quindi legittimo supporre che l'effetto combinato di questi due tipi di modelli possa migliorare la precisione di predizione della funzione risolvendo i problemi di generalizzazione per un elevato numero di parametri.

Una funzione matematica che combina l'andamento di diverse tipologie di curve può essere ottenuta tramite la scelta di determinati moltiplicatori che vanno a determinare quanto una certa funzione incide sul calcolo del risultato. Per rendere la scelta di questi moltiplicatori il più imparziale e precisa possibile

è possibile parametrizzarli sotto forma di un'unica variabile  $k$  che modella la funzione matematica secondo la formula 2.3.

$$acc = k \cdot (-a^{P-b} + c) + (1 - k) \cdot (d \cdot \log(P - e) + f) \quad (2.3)$$

L'equazione 2.3 può essere riscritta in maniera semplificata come in 2.4, riducendone i gradi di libertà per rendere più facile trovare i parametri del modello. Infatti entrambi i parametri  $c$  ed  $f$  regolano la traslazione lungo l'asse verticale della curva e possono essere riscritti sotto forma di un'unica variabile  $c$ .

$$acc = -k \cdot a^{P-b} + (1 - k) \cdot (d \cdot \log(P - e)) + c \quad (2.4)$$

In questa situazione i parametri  $a$ ,  $b$ ,  $c$ ,  $d$ ,  $e$  si comportano analogamente a quanto spiegato in precedenza, mentre il parametro  $k$  può assumere valori interni all'intervallo  $[0, 1]$  e regola la dominanza della funzione logaritmica e di quella esponenziale sul modello finale. Nello specifico per  $k = 0$  si ottiene lo stesso andamento della funzione logaritmica, per  $k = 1$  si ha invece la funzione esponenziale e per  $k = 0.5$  si otterrebbe la media aritmetica delle due curve.

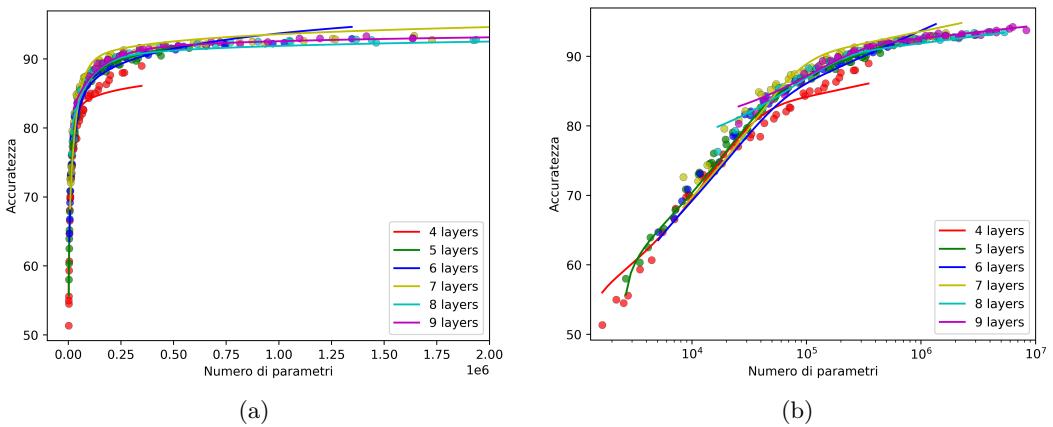


Figura 2.6: *Regressione logaritmica.*

Sebbene questo modello mostri miglioramenti sull'adattamento generale della curva alla distribuzione di dati, è evidente come per elevati valori di numero di parametri la funzione segua l'andamento logaritmico e continui quindi a crescere monotonicamente; ciò è chiaro osservando la figura 2.6 (b) in quanto la regressione si adagia sull'asintoto obliquo presente per l'influenza logaritmica della funzione.

### Regressione iperbolica

È evidente che le regressioni proposte in precedenza non sono adatte per poter studiare determinati intervalli di parametri in cui sia più opportuno scegliere un certo valore di numero di layer ( $N$ ); il principale problema dei modelli proposti sono la scarsa capacità di estrapolazione delle curve a valori nuovi di numero di parametri.

Una funzione che potrebbe ovviare a questi problemi è quella iperbolica, osservando infatti un ramo d'iperbole di questa tipologia  $y = -\frac{1}{x}$ , nel semipiano delle ascisse positive, si può notare una certa somiglianza con le distribuzioni in figura 2.3 (a). La famiglia delle funzioni iperboliche è caratterizzata da due diversi asintoti, uno verticale, utile per delimitare in questo caso il grafico verso sinistra, ed uno orizzontale, necessario per porre un limite di accuratezza massimo verso cui convergere ed evitare i problemi visti con le funzioni iperboliche che crescono monotonicamente.

Il vantaggio delle funzioni iperboliche rispetto a quelle esponenziali per quanto riguarda l'asintoto orizzontale risiede nel fatto che, pur convergendo entrambe al valore imposto dalla retta orizzontale, l'iperbole lo fa più lentamente, mentre l'esponenziale ci arriva in maniera immediata, senza fornire informazioni molto utili sulla differenza qualitativa tra punti distanti fra loro.

La relazione tra accuratezza di classificazione e numero di parametri può quindi essere modellata tramite l'equazione 2.5.

$$acc = -\frac{a}{P-b} + c \quad (2.5)$$

In una funzione iperbolica di questa tipologia si suppone che tutti e tre i parametri  $a$ ,  $b$ ,  $c$  assumano valori positivi, in modo da mantenere coerente l'andamento della curva senza ribaltamenti indesiderati. In particolare  $a$  regola la pendenza della funzione, mentre  $b$  e  $c$  sono responsabili della posizione rispettivamente dell'asintoto verticale e di quello obliquo della curva, ovvero della sua traslazione lungo gli assi cartesiani.

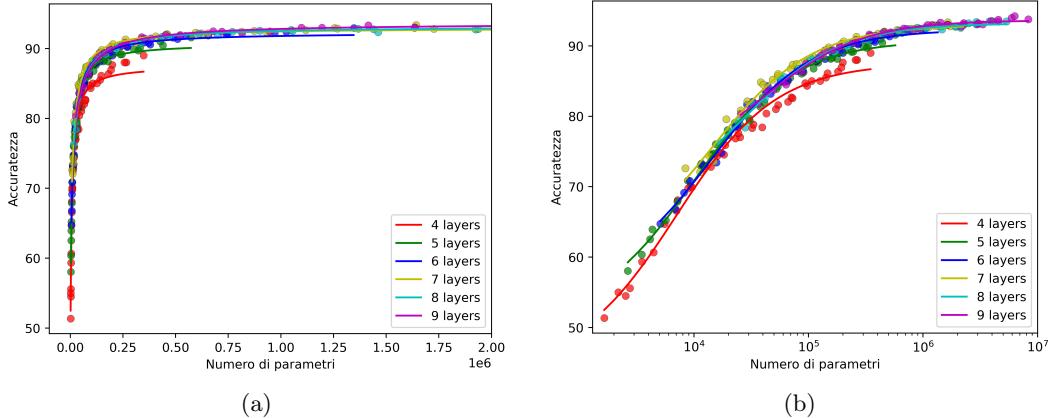


Figura 2.7: *Regressione iperbolica*.

Nonostante i miglioramenti su scala lineare possano sembrare lievi, si riesce ad apprezzare su scala logaritmica l'incremento di precisione di predizione e di adattamento generale ai dati a disposizione.

La scelta della famiglia delle funzioni iperboliche sembra quindi essere vincente, rimane tuttavia da risolvere il problema di adattamento delle curve per alti valori di numero di parametri, si nota per esempio la discrepanza tra i risultati di predizione ed i dati reali corrispondenti a circa centodiecimila parametri per quanto riguarda le PhiNets con 4 layers (colore rosso in figura 2.7).

### Regressione iperbolica generalizzata

Un modo per migliorare ulteriormente il modello di regressione iperbolico è quello di considerare un ulteriore grado di libertà che in precedenza è stato impostato implicitamente ad un valore fisso. Le funzioni iperboliche considerate fino ad ora, infatti, possono essere lette in maniera semplificata come una frazione del tipo  $y = -\frac{1}{x^\phi}$ , in cui l'esponente della variabile indipendente è fissato al valore unitario.

Si possono quindi ampliare i gradi di libertà della funzione per contenere anche un fattore di potenza all'esponente della variabile indipendente. In particolare la nuova funzione di regressione iperbolica ottimizzata segue la relazione descritta dall'equazione 2.6.

$$acc = -\frac{a}{P^\phi} + c \quad (2.6)$$

In questa funzione i parametri  $a$  e  $c$  si comportano esattamente come descritto per la funzione iperbolica, mentre il fattore di potenza  $\phi$ , che può assumere solamente valori positivi (per mantenere l'andamento iperbolico desiderato), regola la forma della curva.

È inoltre importante notare come nel modello descritto dalla formula 2.6 sia venuto a meno il parametro  $b$  presente precedentemente, questa scelta segue il principio del rasoio di Occam dato che il parametro assumeva valori irrilevanti per le grandezze considerate.

Questo modello di regressione sembra essere il migliore per tutti i criteri presi in considerazione, l'adattamento generale delle curve alla distribuzione dei dati è ottimale, gli estremi sinistri e destri sono ben approssimati e sono presenti sia un asintoto verticale, corrispondente all'asse *parametri* = 0, sia uno orizzontale, che varia in base al numero di layer considerato. Inoltre, come ci si potrebbe aspettare, all'aumentare del numero di layer cresce anche l'ordinata dell'asintoto orizzontale di accuratezza e ciò

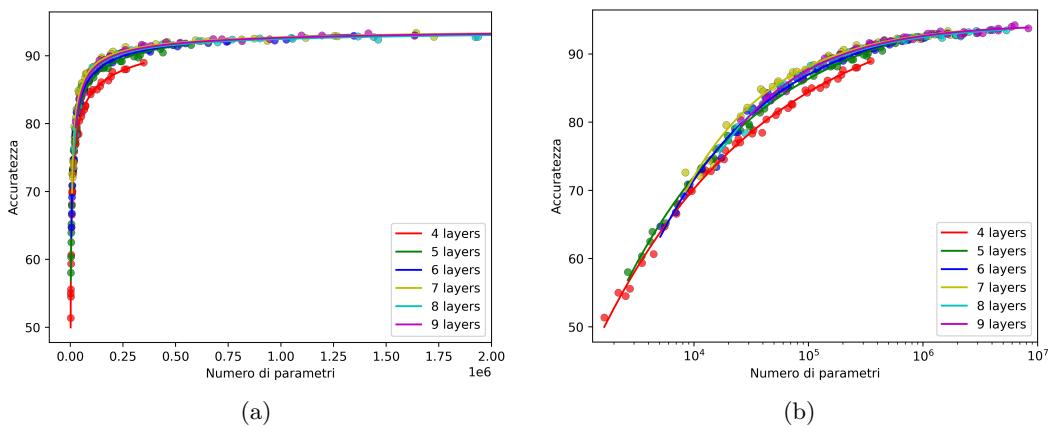


Figura 2.8: *Regressione iperbolica ottimizzata.*

è un risultato ragionevole dato che ci si aspetta una correlazione positiva tra il numero di parametri di una rete (che cresce linearmente con il numero di layer) e la sua accuratezza di classificazione.

### 2.4.2 Metodi di segmentazione

Essendo l’obiettivo della ricerca quello di determinare il miglior valore di  $N$  (numero di strati convoltivi) che ottimizza le prestazioni di una PhiNet per un determinato budget di numero di parametri, sono d’aiuto alcune tecniche di segmentazione di dati che permettono di determinare gli intervalli sull’asse dei parametri all’interno dei quali sia più vantaggioso scegliere un preciso valore di  $N$  per inizializzare la rete più performante.

Sia partendo dalla distribuzione dati originaria, sia dalle funzioni di regressione è possibile, con le giuste tecniche, ottenere una divisione tra intervalli in cui sia esplicito quale valore di numero di layer sia il migliore per ogni range.

Osservando solamente i dati originariamente raccolti dal punto di vista del numero di strati convolutivi è già possibile fornire una stima empirica su quale valore di  $N$  sia il migliore per un certo budget di numero di parametri. Dalla figura 2.3 si può ad esempio notare come per un valore di circa duecentomila parametri a disposizione la scelta peggiore per il numero di layer sia 4 (colore rosso), mentre sia molto vantaggioso scegliere una configurazione di rete con 7 strati convolutivi (colore giallo). Questo semplice processo di confronto andrebbe ripetuto molteplici volte in modo da ottenere dati sufficienti per stimare dei possibili intervalli in cui la scelta migliore di  $N$  sia comune.

Al fine di automatizzare questa procedura e garantire un risultato corretto sono state utilizzate tecniche di campionamento, segmentazione e smoothing delle distribuzioni dei dati.

## Campionamento

In particolare il campionamento adottato per questa analisi consiste nella generazione di 1000 valori differenti di numero di parametri equispaziati in base alla scala logaritmica sull'asse delle ascisse; tale valore di numero di campioni è stato scelto per garantire un risultato accurato e privo di imprecisioni. Per ognuno di questi valori è ora necessario attribuire il numero di strati convolutivi che performa meglio sotto questa condizione, in modo da ottenere coppie di valori di questa tipologia (**campione**,  $N$ ). A tal fine è necessario conoscere e confrontare l'accuratezza di classificazione di una rete con quel numero di parametri per ogni diverso valore di  $N$ . Il calcolo dell'accuracy varia nel caso in cui stiamo considerando la segmentazione della distribuzione di dati originaria o quella delle funzioni di regressione.

Nel primo caso è molto probabile che il valore di numero di parametri campionato non corrisponda a nessuno tra i dati a disposizione e non sia quindi possibile attribuire un risultato univoco per ogni campione. Si può ovviare a questo problema assegnando a ogni campione il valore di accuratezza corrispondente alla PhiNet con performance migliori caratterizzata un numero di parametri appartenenti all’intervallo  $[\text{campione}, \text{campione}/\delta]$ , in cui  $\delta$  è una costante (a cui è stato assegnato il valore di  $\delta = 1.35$ ) che permette di mantenere stabile sulla scala logaritmica l’ampiezza di questi intervalli. Nel

secondo caso invece si può sfruttare la funzione di regressione ottenuta in precedenza per attribuire un valore di accuracy approssimativo, ma verosimile, a ogni campione. Ottenuti quindi i valori di accuratezza, è sufficiente un semplice confronto per attribuire a ogni valore di numero di parametri il numero di layer ( $N$ ) che ne massimizza le performance.

### Segmentazione

Dopo aver generato il numero desiderato di coppie di valori *campione/accuracy*, è necessario eseguire un processo di segmentazione su queste informazioni, che mira ad estrarre gli intervalli che contengono solamente campioni che possiedono  $N$  come attributo comune. Per fare ciò bisogna innanzitutto ordinare le coppie di valori in ordine crescente rispetto al numero di parametri e generare tutti i possibili intervalli che contengono coppie consecutive che condividono lo stesso valore di  $N$ . Il risultato grafico di questa operazione è visibile in figura 4.2 (a).

## 3 Setup sperimentale

In questo capitolo vengono presentati i dettagli e le specifiche tenute in considerazione durante la fase di definizione dell'obiettivo e del sottodomainio di ricerca. Le informazioni di seguito riportate sono indispensabili per poter riprodurre i risultati ottenuti e possibilmente operare un confronto con tecniche o framework alternativi.

Come anticipato nella sezione 0.3, per poter condurre la ricerca in questione è innanzitutto necessario comprendere a pieno quali variabili possano influenzare i risultati finali e definirne un dominio di analisi. Ciò è fondamentale per non rendere lo studio più complesso del dovuto e per non deviare il focus di ricerca su altri aspetti.

I dati raccolti ed i risultati sono in particolare influenzati da aspetti che riguardano l'architettura di rete scelta e da dettagli implementativi in fase di training dei modelli. Nello specifico le variabili individuate sono: la scelta dell'architettura (già discussa nella sezione 2.1), lo spazio di ricerca dei fattori di scala (presentato nella sezione 2.3), la scelta del dataset utilizzato durante la fase di addestramento della rete, la risoluzione delle immagini utilizzate per la classificazione e l'impiego o meno delle tecniche di data augmentation, il numero di epoch utilizzate durante il training, la scelta del metodo di ottimizzazione dei parametri della rete in fase di allenamento ed il valore del learning rate.

Per quanto riguarda il dataset, la scelta è ricaduta su *CIFAR-10* [7], ovvero un insieme di immagini composto da sessantamila immagini a colori, equamente divise tra 10 classi, raffiguranti animali e mezzi di trasporto. Tali immagini possiedono una risoluzione di  $32 \times 32$  pixel, tuttavia in fase di elaborazione preliminare viene eseguito un *up-scaling* in modo tale da quintuplicare le loro dimensioni e ottenere immagini da  $160 \times 160$  pixel. Questo processo è fondamentale per definire le dimensioni, in termini di numero di parametri, della rete e per adattare la classificazione delle immagini al reale campo di applicazione, le PhiNets hanno appunto l'intento di essere utilizzate su dispositivi embedded che acquisiscono le immagini da semplici microcamere integrabili. Alle immagini è inoltre applicato un processo di data augmentation [14] che consiste nel generare nuovi dati artificiali tramite trasformazioni come rotazioni, zoom, riflessioni, al fine di aumentare la dimensione e la varietà del set di addestramento e possibilmente migliorare le prestazioni della rete [10].

Tutte le PhiNets i cui dati di accuratezza sono stati utilizzati in questa analisi sono state addestrate per 100 epoch, alle quali sono state aggiunte in fase iniziale 10 ulteriori epoch di *warm-up*, che consentono al modello di acquisire gradualmente una migliore comprensione dei dati, adattando i pesi dei parametri in modo più stabile e coerente durante le prime fasi del training. Come metodo di aggiornamento dei pesi è stato scelto l'ottimizzatore Lamb [16] (con learning rate impostato a  $lr = 0.01$ ), che migliora la stabilità e la convergenza dell'addestramento delle reti, mantenendo i pesi dei parametri bilanciati nel corso dell'addestramento e riducendo il problema del *vanishing weights*.

I dettagli implementativi sono disponibili nella repository ufficiale del toolkit di *micromind* [2].

# 4 Risultati

In questo capitolo vengono presentati e discussi i risultati ottenuti tramite le tecniche di analisi utilizzate, verificando anche la validità di strumenti di studio come l'applicazione delle Linear Probes.

## 4.1 Analisi delle funzioni di regressione

Nella sezione 2.4 sono state presentate le tecniche e le funzioni di regressione utilizzate per approssimare l'andamento dei dati, tuttavia è necessaria un'analisi più accurata per determinare quale modello sia quello che si predispone meglio al caso di studio.

### 4.1.1 Analisi qualitativa

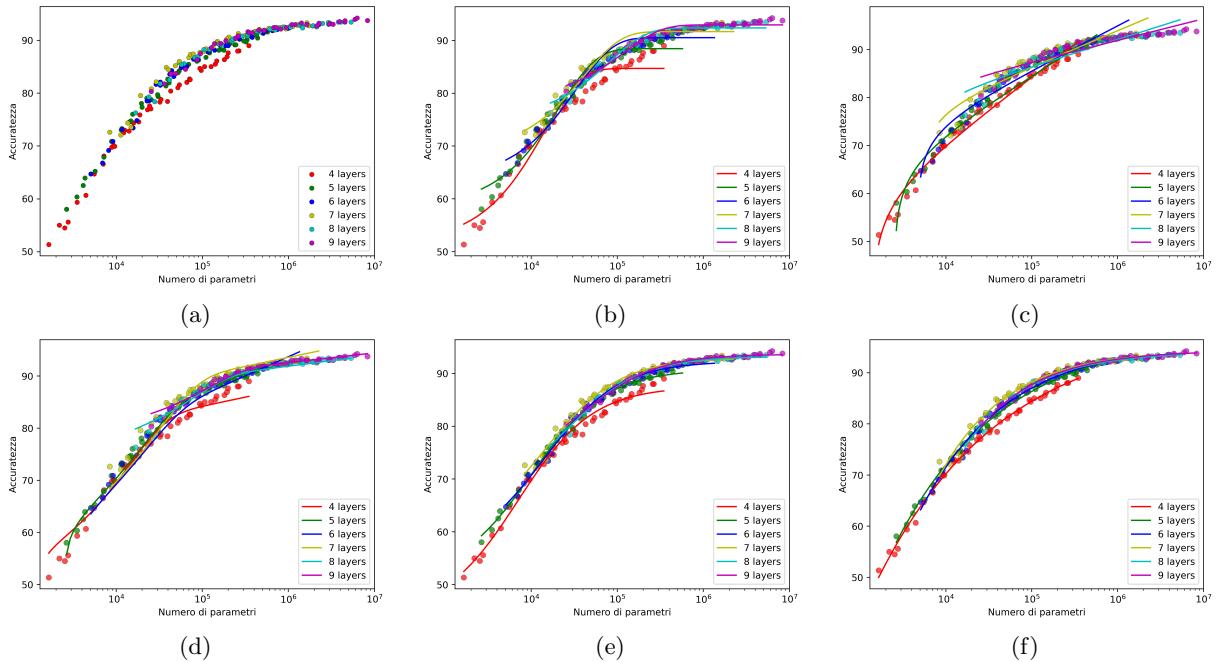


Figura 4.1: *Funzioni di regressione*. (a) Distribuzione dei dati raccolti, (b) regressione esponenziale, (c) regressione logaritmica, (d) combinazione convessa tra funzione logaritmica ed esponenziale, (e) regressione iperbolica, (f) regressione iperbolica generalizzata.

In figura 4.1 sono riportati gli andamenti delle funzioni di regressione testate, visualizzati sulla scala logaritmica per poter meglio apprezzare le differenze.

Osservando qualitativamente risulta immediato notare come alcuni dei modelli di regressione proposti non approssimino bene l'andamento dei dati in possesso e non siano quindi utili per fornire una stima precisa dell'accuratezza di classificazione di una PhiNet. In particolare è chiaro come le funzioni presenti nei grafici (b), (c) e (d) non si adattino in maniera accettabile alle distribuzioni proposte.

D'altra parte entrambi i modelli di regressione (e) ed (f) non solo sembrano adattarsi bene ai dati raccolti, ma rispettano anche le intersezioni tra le curve distintive dei vari layers.

### 4.1.2 Analisi quantitativa

Ai fini di quantificare concretamente l'efficacia e l'accuratezza delle previsioni di tali modelli di regressione è necessario utilizzare delle metriche di valutazione che offrano una misura oggettiva delle loro prestazioni. L'importanza di una corretta valutazione dei modelli di regressione risiede nel fatto che essa consente di identificare i punti di forza e di debolezza dei modelli (presenza o meno di asintoti verticali e/o orizzontali), nonché di effettuare confronti accurati tra diverse tecniche.

La metrica più comune per la valutazione di un modello di regressione è l'errore quadratico medio (o *Root Mean Square Error*, RMSE), che, tramite il calcolo del quadrato della differenza tra il valore predetto e quello atteso, penalizza maggiormente la presenza di errori di previsione elevati rispetto ad esempio all'errore assoluto medio (MAE), che invece tiene in considerazione soltanto il valore assoluto della differenza in questione.

In questa analisi viene utilizzata per la valutazione del modello la metrica del RMSE, calcolata su modelli di regressione ottenuti osservando solamente un sottoinsieme ridotto della totalità dei dati originariamente disponibili, tramite la tecnica della *cross-validation*.

Questa metodologia di valutazione della validazione incrociata consente di ottenere metriche affidabili per il confronto tra modelli di regressione in quanto ogni modello viene valutato (ed addestrato) su diverse combinazioni di dati. La tecnica della *cross-validation* consiste in un iniziale suddivisione dei dati a disposizione in diversi *fold*, ovvero insiemi equipotenti che si distinguono tra loro per le diverse partizioni dei dati tra set di addestramento e set di test. In questo modo si ottengono  $K$  insiemi contenenti gli stessi dati ma che si distinguono per la diversa suddivisione delle informazioni tra i vari set. Per ogni *fold* viene allenato il modello di regressione sul rispettivo set di addestramento e valutato, tramite il calcolo delle metriche scelte, sul set di test; l'intero processo viene ripetuto per ogni *fold* disponibile ( $K$  volte) in modo da garantire una valutazione affidabile e non influenzata da eventuali outlier.

Nel caso in questione, dato che si ottengono 6 diverse funzioni di regressioni (una per ogni valore diverso di numero di layer) verrà applicata la tecnica di cross-validation altrettante volte, calcolando l'errore quadratico medio per ogni valore di numero di layer. In particolare i dati a disposizione sono stati suddivisi in  $K = 500$  fold equipotenti, ognuno partizionato in maniera differente mantenendo però le proporzioni di 70%-30% tra il set di addestramento e quello di test.

La procedura inizia quindi con la creazione dei 500 fold, riordinati randomicamente, per ogni insieme di dati appartenenti ad uno specifico valore di numero di layer. Per ciascun fold viene calcolata la funzione che meglio si adatta al set di addestramento tramite l'allenamento del modello di regressione scelto e viene successivamente eseguita la valutazione sul set di test. In questo modo per ogni fold differente si ottiene un valore di errore quadratico medio che verrà mediato con quello di tutti i  $K$  fold generati. Ciò permette di ottenere un valore medio dei RMSE per ogni valore differente di numero di layer per un dato modello di regressione. La metrica finale utilizzata per il confronto è quindi la media degli errori quadratici medi per ogni funzione che approssima gli andamenti dei dati distribuiti sui vari layer.

In questa analisi i modelli di regressione che sembravano adattarsi meglio alla distribuzione di dati sono la regressione iperbolica e la sua generalizzazione. I calcoli delle metriche confermano che la funzione iperbolica generalizzata sia caratterizzata da una misura di errore inferiore rispetto alla semplice funzione iperbolica, per quest'ultima si ottiene infatti un valore medio dei RMSE di 0.807%, mentre il modello iperbolico ottimizzato riduce l'errore del 6% facendolo attestare ad un valore di 0.758%. In entrambi i casi, l'errore ottenuto può essere assimilato a una statistica che descrive l'incertezza del processo di training.

## 4.2 Smoothing

Come si può notare dalla figura 4.2 (a), la segmentazione ottenuta è molto rumorosa, soggetta ed outliers ed influenzata dal fatto di avere solamente un numero discreto di dati. Per rendere il tutto più semplice ed efficace da visualizzare viene utilizzato quindi un processo di *smoothing* dei dati, che va ad eliminare dai segmenti ottenuti tutti quelli la cui ampiezza sta sotto ad una certa soglia, detta *threshold*, e ricostruisce gli intervalli rimanenti. In questa situazione come "ampiezza" si intende il logaritmo in base 10 che ha come argomento il rapporto tra l'estremo superiore e quello inferiore del segmento; in particolare la threshold utilizzata assume il valore di  $threshold = 0.13$  ed un intervallo verrà quindi considerato da scartare quando  $\log_{10}\left(\frac{\text{estremo superiore}}{\text{estremo inferiore}}\right) < threshold$ . Il risultato finale ottenuto sulla distribuzione di dati originaria è quindi visibile in figura 4.2 (b), mentre quello ottenuto tramite la sola funzione di regressione (che non ha bisogno di alcun processo di smoothing) è mostrato in figura 4.2 (c).

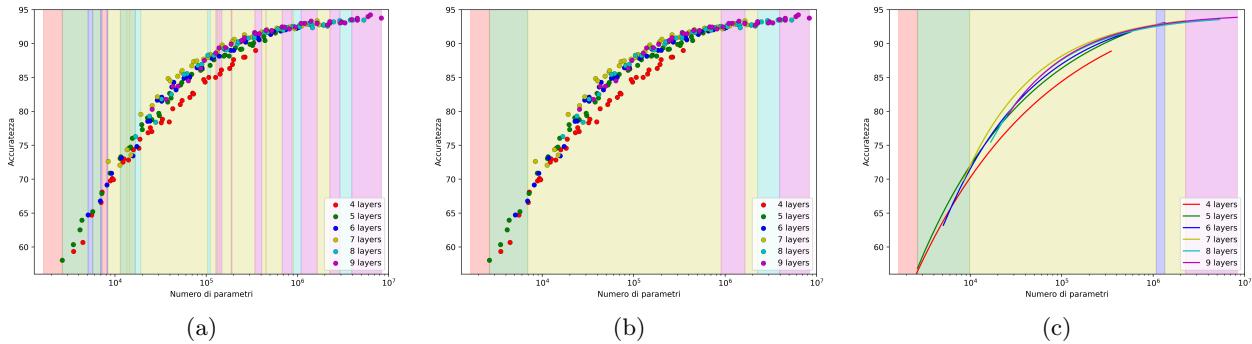


Figura 4.2: *Segmentazione e smoothing*. L’immagine (a) rappresenta graficamente la segmentazione eseguita sulla distribuzione di dati, (b) aggiunge il processo di smoothing, nella figura (c) la segmentazione è stata eseguita solamente basandosi sulle informazioni derivanti dalla funzione di regressione.

### 4.3 Valutazione dell’efficacia del Linear Probing

Dato che è stato mostrato come la tecnica del Linear Probing (sezione 1.3) sia in grado di mostrare il contributo ed il ruolo di ogni layer all’interno di una rete convoluzionale, è ragionevole pensare che tale metodo possa anche aiutare ad analizzare e valutare le performance di determinati modelli di PhiNets per poter trarre una conclusioni su quale sia il miglior numero di strati convolutivi da scegliere.

Per poter mostrare come una determinata configurazione di iperparametri per una PhiNet con un certo budget di risorse computazionali sia più conveniente rispetto ad un’altra, sarebbe utile mostrare una qualche correlazione tra gli andamenti delle loss delle linear probes per specifici valori di numero di layer e le performance di classificazione finali della rete.

In figura 4.3 è mostrato l’andamento dei valori di loss per le linear probes applicate ad una PhiNet.

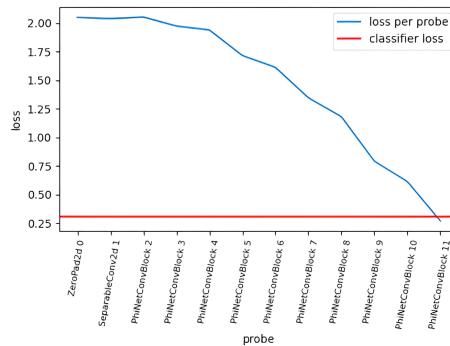


Figura 4.3: *Loss per probe di una PhiNet*.

A differenza di quanto visto nel grafico 1.5, che mostra i risultati dell’applicazione delle linear probes su una *ResNet-50* [3], nel caso delle PhiNets l’andamento mostrato dalla metrica di loss è monotono decrescente all’aumentare del blocco convoluzionale considerato. Dato che il trend monotono decrescente è comune alla quasi totalità dei modelli considerati, si può facilmente trarre la conclusione che l’architettura di rete delle PhiNets è particolarmente efficiente per il task di classificazione di immagini e che non sono presenti layer convolutivi che degradano le prestazioni.

Tuttavia tale considerazione riguarda le performance generali delle reti e non tiene conto delle differenze di configurazione degli iperparametri, come ad esempio i diversi numeri di layer. Indagando più nello specifico non sono state però trovate metriche che mostrassero una chiara correlazione tra l’andamento della funzione di loss delle linear probes per determinati valori di numeri di layer e le prestazioni di classificazione delle reti.

Con ciò non si intende dire che le linear probes non siano uno strumento di analisi utile, ma solamente che non mostrano un chiaro supporto per il confronto tra modelli di PhiNets, e rimangono altresì una preziosa risorsa per il crescente trend dell’interpretabilità dei modelli nell’ambito delle reti convoluzionali.

## 5 Conclusioni

Come anticipato nel sommario, nell'ampio ambito degli algoritmi di machine learning assumono grande importanza gli applicativi scalabili che ben si adeguano alla vasta diversità di dispositivi con scarse risorse computazionali disponibili sul mercato. Queste tipo di architetture rappresentano lo stato dell'arte per i modelli di apprendimento volti all'elaborazione delle immagini in quanto la loro flessibilità, data dai fattori di scala, semplifica il processo di adattamento delle reti alla piattaforma specifica di applicazione.

Tuttavia non è ancora chiaro, o per lo meno non lo è nel caso della famiglia di reti delle PhiNets, quali siano i criteri per scegliere la configurazione dei fattori di scala che massimizzi le prestazioni del modello e soddisfi i vincoli computazionali imposti.

Come visto nella sezione 2.1, i fattori che modificano la struttura di rete di una PhiNet sono 4, ovvero  $\alpha$ ,  $\beta$ ,  $t_0$ ,  $N$ , che regolano rispettivamente la larghezza, la forma, il numero di canali ed il numero di layer rete. D'altra parte si hanno a disposizione solamente 3 vincoli (occupazione di memoria FLASH, occupazione di memoria RAM e consumo energetico) che aiutano a regolare tali iperparametri, c'è quindi la necessità di trovare una condizione ulteriore che aiuti a determinare gli iperparametri migliori.

Questa analisi ha mostrato come ci sia una chiara correlazione tra il numero di layer, il numero totale di parametri di una rete e le sue prestazioni ed è quindi un'informazione utile per poter determinare a priori uno dei 4 fattori di scala. In particolare si può vedere dalla figura 4.2 come nell'intervallo di spazio di archiviazione di interesse, che va da circa 3KB (trentamila parametri) a 2MB (duemilioni di parametri), ci siano differenze nella scelta del numero di layer che ottimizza le performance di rete. Si può vedere facilmente come per un budget di numero di parametri inferiore a 2500 (che nel caso di codifica ad 8 bit per parametro corrisponde a 2.5KB) il numero di strati convolutivi migliore da scegliere è 4, ad ogni modo tale situazione è improbabile in un campo di applicazione reale dato che i dispositivi con vincoli di risorse così stringenti sono rari e che queste reti avrebbero un'accuratezza di classificazione molto bassa. Fino a circa diecimila parametri (corrispondenti a 10KB) è invece conveniente l'utilizzo di una PhiNet composta da 5 layers, utile ad esempio se si vuole implementare la classificazione di immagini su un dispositivo molto limitato come quelli della famiglia Arduino, mentre fino a circa un milione di parametri (1MB) la configurazione migliore risulta quella con 7 layers; questo ultimo risultato ricopre gran parte dei casi applicativi reali, dato che sono poche le piattaforme embedded che eccedono questo limite di spazio di archiviazione. Per quanto riguarda quelle reti che possiedono più di un milione di parametri invece, sembra esserci una dominanza di prestazioni per le PhiNet che possiedono 9 layers, tuttavia il vantaggio di accuratezza rispetto agli altri numeri di strati convolutivi sembra minimo. Tali risultati offrono quindi una guida utile alla configurazione del numero di strati convolutivi di una PhiNet, dato che solamente conoscendo il budget di memoria FLASH del dispositivo di applicazione è possibile determinare a priori il numero di layers che definiscono una PhiNet performante.

L'obiettivo di ricerca era tuttavia quello di determinare i 4 iperparametri che definiscono la rete migliore in termini di accuratezza e con le informazioni ora disponibili le decisioni di configurazione sono meno complesse. Rimanendo infatti gli stessi 3 vincoli (RAM, FLASH, operazioni) è ora più semplice scegliere i restanti 3 fattori di scala e ciò è fattibile seguendo la strategia di *tuning* proposta nel paper di presentazione delle PhiNets [9].



# Bibliografia

- [1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2018.
- [2] Francesco Paissan, Alberto Ancilotto, Matteo Beltrami, Mariam Jamal, Elisabetta Farella. micromind toolkit. <https://github.com/micromind-toolkit/micromind>.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] Alex Krizhevsky. Datasets: cifar-10 & cifar-100. <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [8] 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.
- [9] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. Phinets: A scalable backbone for low-power ai at the edge. *ACM Trans. Embed. Comput. Syst.*, 2022.
- [10] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.
- [11] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.
- [12] 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.
- [13] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.
- [14] Ross Wightman. Pytorch image models (timm). <https://github.com/huggingface/pytorch-image-models>.
- [15] LeCun Yann, Bengio Yoshua, and Hinton Geoffrey. Deep learning. *Nature*, 2015.

- [16] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes, 2020.