UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Algoritmi di Class Incremental Learning per classificazione di immagini

Laureando: Nicola Maritan
Relatore: Prof. Stefano Ghidoni
Correlatore: Dott. Leonardo Barcellona

Anno Accademico 2022-2023

Indice

Sommario

Molti sistemi moderni sfruttano la *Computer Vision* per estrarre informazioni dalle immagini. Una importante applicazione di questo settore è la classificazione, che prevede di assegnare un'etichetta a un'immagine in base al suo contenuto visivo. Molti approcci in letteratura si basano sul *Deep Learning* (DL) e utilizzano modelli allenati per distinguere un insieme di classi fissato durante la fase di addestramento. Una volta che questa si è conclusa, non vengono apportate ulteriori modifiche al modello. Tuttavia, nella realtà possono verificarsi scenari in cui il modello deve essere in grado di apprendere nuove classi man mano che si incontrano nuovi dati. Se il modello viene riaddestrato esclusivamente sulle classi appena introdotte, si verifica il *Catastrophic Forgetting* (CF), fenomeno che consiste nella perdita di accuratezza nel riconoscimento delle classi apprese in precedenza. D'altro canto, riaddestrare il modello sulle classi nuove e passate può essere molto oneroso in termini di tempo e costo computazionale. Per ovviare a tali limitazioni, la ricerca ha prodotto degli algoritmi di *Class Incremental Learning* (CIL), cioè degli approcci che permettono al modello di apprendere un nuovo insieme di classi, mantenendo al contempo le capacità di riconoscimento apprese in precedenza. Questa tesi esplora i principali tipi di algoritmi proposti per sviluppare un'entità in grado di ampliare il numero di classi riconosciute nel tempo. Viene formalizzato il problema del CIL e vengono introdotti i tre tipi di approcci presenti in letteratura: *Model-Growth* (MG), *Fixed-Representation* (FR) e *Fine-Tuning* (FT). Inoltre, per ciascuno di essi, vengono trattati due approcci. Successivamente, i tre tipi vengono messi a confronto sulla base di alcune proprietà. Segue una discussione dei risultati ottenuti dagli esperimenti riportati in letteratura, i quali confermano la validità della teoria. In seguito, vengono presentate delle applicazioni reali del CIL, identificando quale approccio risulta più adatto in ognuna di esse. Inoltre, vengono delineati gli elementi che motivano la scelta per ciascuno degli approcci trattati.

Capitolo 1

Introduzione

L'essere umano è capace di apprendere e migliorare continuamente grazie alla sua capacità di adattarsi alle nuove situazioni e di apprendere dai propri errori. Questa capacità di apprendere in modo incrementale è stata una caratteristica fondamentale della nostra evoluzione come specie e ha permesso alla civiltà di progredire nel tempo. La stessa umanità, con tale capacità assimilative, è stata in grado di realizzare delle entità intelligenti capaci di apprendere da un insieme di dati a loro disposizione. La disciplina che si cura di tali agenti è chiamata *machine learning* (ML) [25, 82] e ricopre una branca dell'intelligenza artificiale. Però, i modelli di ML tradizionali non sono in grado di classificare istanze per le quali non sono stati addestrati in precedenza [13, 16, 28, 77, 93]. In questi casi, è necessario ripetere il processo di addestramento del modello, integrando i dati già conosciuti con quelli nuovi [7, 8, 77]. Se si decide di addestrare il modello solo sulle nuove classi, si riscontra un fenomeno noto in letteratura come *catastrophic forgetting* (CF) [10, 30, 73, 77]. Esso si manifesta come una significativa perdita di accuratezza, poiché il modello tende a "dimenticare" come riconoscere le classi precedentemente apprese una volta che le nuove vengono introdotte [30, 43].

1.1 Problema e risvolti pratici

Il CF rappresenta un ostacolo alla realizzazione di una entità in grado di acquisire conoscenza nel tempo, pur conservando quella passata [74]. La conoscenza appresa precedentemente a partire da un insieme di dati viene persa non appena se ne acquisisce di nuova da un contesto differente [81]. Risolvere il problema del CF diventa semplice se si ha a disposizione sufficiente potenza di calcolo, memoria e tempo, poiché in questo caso è possibile riallenare il modello utilizzando tutti i dati disponibili [8]. In tale situazione, i parametri vengono ottimizzati contemporaneamente per tutte le classi fin dall'inizio, evitando così il rischio di dimenticare la conoscenza appresa in precedenza [30]. Tuttavia, nella realtà, questi requisiti spesso non sono soddisfatti. Addestrare un modello con un numero elevato di classi in input richiede molto tempo e potenza di calcolo [35, 41, 84].

La capacità di far acquisire conoscenza a una entità in modo progressivo è divenuta possibile grazie a degli algoritmi di *Incremental Learning* (IL) [28, 59], capaci di contrastare il CF. L'IL si riferisce a un sistema di apprendimento in grado di imparare continuamente nuove conoscenze da campioni sconosciuti e mantenere la maggior parte delle conoscenze precedentemente acquisite [81]. In questo modo, è possibile realizzare una entità in grado di integrare nel tempo nuove capacità, senza dimenticarsi di quelle già in possesso.

Un tale sistema di apprendimento ha dei notevoli risvolti pratici. Ad esempio, nella robotica, i *robot assistivi sociali* sono un particolare tipo di robot assistivo progettato per interagire socialmente con gli esseri umani [14]. Essi potrebbero svolgere un ruolo importante per quanto riguarda la salute e il benessere psicologico degli anziani. I robot assistivi richiedono di adattarsi rapidamente a nuove situazioni e ambienti [27]. Ciò è essenziale per garantire l'efficacia e l'efficienza dell'azione del robot, nonché la sicurezza delle persone coinvolte. Inoltre, in tale contesto non sempre è plausibile la presenza di memoria atta a conservare tutti i campioni del training set [31].

In campo medico, un modello di apprendimento addestrato con dati etichettati da pazienti precedenti può aiutare a formulare diagnosi o a prendere decisioni [4, 44, 80]. Con l'arrivo di un nuovo individuo, si vogliono introdurre nel sistema di decisione le analisi su di esso sotto forma di segnali

fisiologici monodimensionali [85]. Quando un modello viene utilizzato per trattare tali segnali, esso potrebbe incontrare nuove patologie mai viste prima [85]. Di conseguenza, il sistema deve essere in grado di riconoscere le nuove condizioni assieme a quelle passate [49].

L'IL può venire applicato nel momento in cui le informazioni vengono acquisite dinamicamente nel tempo. Ciò accade specialmente nella raccolta dati da parte di un sensore o dal network monitoring [5, 75].

Nel campo delle applicazioni mobili mancano potenza di calcolo e tempo. L'utilizzo di una GPU ad alte prestazioni può accelerare notevolmente il processo di addestramento di una rete neurale [61, 67], il che rende impensabile riallenare un modello di grandi dimensioni su un dispositivo mobile. Questo è particolarmente problematico per applicazioni di riconoscimento facciale [66], dove il sistema deve imparare a riconoscere nuovi volti nel corso del tempo.

Da questi esempi e motivazioni si deduce l'importanza pratica di sviluppare degli algoritmi di IL veloci, parsimoniosi di memoria e che mantengano una accuratezza comparabile con il modello addestrato con tutte le istanze del training set disponibili fin da subito.

1.2 Deep Learning

Il *Deep Learning* (DL) è diventato uno strumento fondamentale per la risoluzione dei problemi classificazione nella computer vision [26, 88]. Esso è una fa parte di una ampia famiglia di metodi di ML, basati sulle *Artificial Neural Networks* (ANN) [47], permettendo a modelli computazionali composti da più layer di elaborazione (da cui l'aggettivo *deep*) di apprendere rappresentazioni dei dati con molteplici livelli di astrazione [48]. L'elemento computazionale base viene chiamato *perceptron* o *neurone* [4], che riceve input da fonti esterne e ha alcuni parametri interni, appresi durante l'addestramento, che producono output. Una ANN consiste in un *Multilayer Perceptron* (MLP), il quale contiene uno o più layer nascosti, con molteplici neuroni al loro interno. Un MLP è una funzione matematica ottenuta componendo molteplici funzioni più semplici, rappresentate dai singoli neuroni [11].

1.3 Convolutional Neural Networks

Le Convolutional Neural Networks (CNN) sono un particolare tipo di ANN, sviluppate appositamente per l'elaborazione delle immagini. Le CNN permettono di estrarre in modo automatico forme e contorni, da cui vengono ricavate delle *features* utilizzate per la classificazione da una rete neurale [4]. Una CNN è principalmente composta da tre componenti: *convolutional layer*, *pooling layer* e *classification stage*.

1.3.1 Convolutional layer

L'immagine in input viene convoluta utilizzando dei *kernel* che possono essere appresi dal modello [4, 65]. Un singolo convolutional layer può individuare bordi e forme elementari, ma l'utilizzo consecutivo di tali strati permette di rilevare la presenza di composizioni complesse, come volti o oggetti completi [2]. L'operazione di convoluzione rappresenta una funzione lineare [94]. Tuttavia, nell'ambito di questa operazione, è comune osservare che l'output subisce una successiva trasformazione non lineare. Si è dimostrato che tale introduzione migliora le performance e la velocità di apprendimento del modello [18, 39]. Alcune trasformazioni non lineari sono la *Rectified Linear Unit* (ReLU) e la funzione sigmoidale [2].

1.3.2 Pooling layer

Un pooling layer si occupa di ridurre la dimensione e la complessità dei dati in input, pur mantenendo le informazioni più importanti di una data regione dell'immagine [2]. Un pooling layer sussegue un convolutional layer e restituisce una matrice di valori in output [4]. I due più noti tipi di pooling sono *average-pooling* e *max-pooling* [4, 53]. Nel caso dell'average-pooling, una regione di $N \times N$ valori in input viene mappata a un valore in output pari alla loro media. D'altra parte, nel caso del max-pooling, viene selezionato il valore più alto tra quelli presenti in una regione $N \times N$ [4]. In entrambi i casi, a una regione $N \times N$ dell'input viene associata una regione 1×1 dell'output. Il parametro N viene scelto a priori ed è fissato.

1.3.3 Classification stage

Il classification stage è composto da uno o più *fully-connected* (FC) layer, un particolare tipo di layer in cui ogni neurone è connesso a tutti i neuroni dello strato precedente. L'ultimo FC layer calcola il punteggio di ogni classe a partire dai layer precedenti [4]. Il primo FC layer accetta in input vettore, ottenuto dal *flattening* (i.e. una trasformazione da matrice 2-dimensionale a vettore 1-dimensionale) dell'output dell'ultimo pooling layer [94].

1.4 Class Incremental Learning

Questa tesi si concentra sullo studio di algoritmi di *Class Incremental Learning* (CIL) per il riconoscimento di immagini, il cui problema consiste nell'apprendere a riconoscere nuove classi assieme a quelle già note. I diversi metodi presenti in letteratura vengono raggruppati in tre principali gruppi [10], ovvero: *Model-Growth* (MG), *Fixed-Representation* (FR) e *Fine-Tuning* (FT). Per ognuno di questi vengono trattati due algoritmi, cioé delle tecniche che permettono al modello di apprendere un nuovo insieme di classi, mantenendo al contempo le capacità di riconoscimento apprese in precedenza. L'approccio seguito dal particolare algoritmo permette di poterlo assegnare ad uno dei tre metodi sopra elencati. Il lavoro confronta tali algoritmi in diversi contesti di memoria disponibile e numero di passi incrementali. L'obiettivo è quello di fornire un'ampia panoramica delle tecniche disponibili, dalla loro introduzione fino alle versioni più recenti, e di valutarne le prestazioni in diverse condizioni di apprendimento incrementale. Il Capitolo 2 formalizza il problema del CIL, definisce i principali tipi di algoritmi e le proprietà ricercate in essi. I Capitoli 3, 4 e 5 analizzano, rispettivamente, degli algoritmi Model-Growth, Fixed-Representation e Fine-Tuning. Il Capitolo 6 mette a confronto le proprietà di ciascun tipo di algoritmo e individua delle possibili applicazioni per ciascuno di essi. Inoltre, vengono discussi i risultati per verificare se essi confermano o meno ciò che viene discusso nella teoria. Infine, il Capitolo 7 conclude la tesi discutendo alcune prospettive future del CIL.

Capitolo 2

Class Incremental Learning

In questo capitolo viene formalizzato il problema del CIL e se ne introducono le nozioni fondamentali. Il CIL si occupa dell'aggiunta di nuove classi a un modello esistente senza doverlo riaddestrare completamente. In altre parole, il CIL mira a consentire a un modello di apprendere nuove classi in modo incrementale senza dimenticare ciò che ha imparato dalle classi precedenti. Viene discusso il problema del CF, il quale si verifica quando un modello che apprende in modo incrementale scorda le conoscenze precedentemente acquisite durante l'aggiunta di nuove classi [10, 30, 73, 77]. Un algoritmo di CIL deve essere capace di contrastare tale fenomeno [10]. Il capitolo definisce poi delle proprietà ricercate negli algoritmi di CIL. Gli approcci presenti in letteratura vengono suddivisi in tre gruppi, differenziandoli nel modo in cui tentano di contrastare il CF.

2.1 Formalizzazione del problema del CIL

Belouadah et al. [10] hanno proposto una formalizzazione rigorosa del problema del CIL. Dati uno stato iniziale non incrementale S_0, un modello M_0 è allenato su un dataset D_0, definito come

$$D_0 = \{(X_0^j, Y_0^j); j = 1, 2, \ldots, P_0\}, \tag{2.1}$$

dove X_0^j e Y_0^j sono rispettivamente l'insieme delle immagini e l'insieme delle label per la classe j-esima in S_0. Il numero di classi nel primo stato non incrementale si esprime con N_0 e vale $N_0 = P_0$. Si denota con T il numero di stati includendo lo stato iniziale e gli $T - 1$ stati incrementali.

Un nuovo batch di P_t classi viene inserito in ogni stato incrementale S_t e l'obiettivo è addestrare un modello M_t in grado di riconoscere N_t classi (*task* t):

$$N_t = \sum_{i=0}^{t} P_i. \tag{2.2}$$

Questo modello viene allenato utilizzando lo stato precedente M_{t-1} su un dataset D_t formulato, in modo analogo a (2.1), come

$$D_t = \{(X_t^j, Y_t^j); j = 1, 2, \ldots, P_t\} \cup K. \tag{2.3}$$

Tutti i dati delle nuove P_t classi sono disponibili, mentre solo un sottoinsieme limitato K è disponibile per i dati delle $N_{t-1} = \sum_{i=0}^{t-1} P_i$ classi passate.

I più recenti algoritmi di CIL sono implementati utilizzando CNN come struttura di base [7, 55, 77, 78, 92]. Un modello M_t include due principali componenti: un *feature extractor* F_t e un classificatore C_t.

Il feature extractor è definito come

$$F_t \colon X_t \to \mathbb{R}^d, \tag{2.4}$$

$$\mathbf{x} \mapsto F_t(\mathbf{x}) = \mathbf{f}_t^{\mathbf{x}}, \tag{2.5}$$

dove $\mathbf{f}_t^{\mathbf{x}}$ è una rappresentazione vettoriale d-dimensionale dell'immagine $\mathbf{x}$, detta anche *deep representation*.

Il classificatore è definito come

$$C_t \colon \mathbb{R}^d \to \mathbb{R}^{N_t}, \tag{2.6}$$

$$\mathbf{f}_t^{\mathbf{x}} \mapsto C_t(\mathbf{f}_t^{\mathbf{x}}) = \mathbf{o}_t, \tag{2.7}$$

dove $\mathbf{o}_t = \left(o_t^1, o_t^2, \ldots, o_t^{N_t}\right)$ è il vettore dei punteggi di grandezza N_t. Esso rappresenta quantitativamente il punteggio di predizione per ogni classe. L'implementazione del classificatore dipende dall'algoritmo [vedasi 78]. Nel caso in cui vi sia presente un FC layer come output layer vale

$$\mathbf{f}_t^{\mathbf{x}} \mapsto C_t(\mathbf{f}_t^{\mathbf{x}}) = \mathbf{f}_t^{\mathbf{x}} \times \mathbf{W}_t + \mathbf{b}_t = \mathbf{o}_t, \tag{2.8}$$

dove $\mathbf{W}_t$ e $\mathbf{b}_t$ sono la matrice dei pesi e il vettore di bias dell'ultimo fully-conncected layer di grandezza (d, N_t) e N_t rispettivamente. Il valore d del vettore delle feature dipende dalla architettura CNN utilizzata.

In una *Deep Neural Networks* (DNN) end-to-end [36] una funzione di softmax è applicata a $\mathbf{o}_t$ per ottenere un vettore $\mathbf{p}_t = \left(p_t^1, p_t^2, \ldots, p_t^{N_t}\right)$, dove p_t^i è la probabilità che l'immagine in input appartenga alla classe i-esima. Per la classe j-esima allo stato S_t, la funzione è definita come segue:

$$p_t^j(\mathbf{x}) = \frac{e^{o_t^j(\mathbf{x})}}{\sum_{l=1}^{N_t} e^{o_t^l(\mathbf{x})}}. \tag{2.9}$$

Le proprietà della funzione di softmax assicurano che [6]

$$\sum_{i=1}^{N_t} p_t^i = 1. \tag{2.10}$$

In tal caso la classe predetta è quella per cui si è ottenuto il valore di p_t^i maggiore. Nonostante ciò, è possibile implementare C_t come un classificatore esterno [77].

2.2 Catastrophic Forgetting

Le DNNs si sono rivelate fondamentali per risolvere problemi di computer vision. In un contesto di IL le DNNs soffrono del CF, fenomeno ben noto in letteratura [10, 30, 73, 77]. Il CF consiste nella tendenza della rete neurale a perdere conoscenza acquisita in un task precedente nel momento in cui il modello vuole apprenderne uno nuovo. Nel caso del CIL, la conoscenza allo stato S_t risulta nella capacità di classificare un'immagine in una delle N_t classi. Una rete che soffre di CF è *biased* [92], cioè è più propensa ad assegnare una immagine alle classi $[N_{t-1}+1, N_t]$ piuttosto che a quelle in $[1, N_{t-1}]$. Questo significa che un modello che soffre di bias predilige l'attribuzione di una classe appena appresa a una qualsiasi istanza.

2.3 Confusion matrix

Uno strumento grafico efficace per visualizzare il bias è la *confusion matrix*. Essa è una tabella utilizzata per definire le performance di un algoritmo di classificazione [15, 40]. Ogni riga e colonna della matrice rappresentano una classe. L'elemento a_{ij} della matrice indica quante istanze della classe i sono state assegnate alla classe j. Buone performance si hanno se la matrice di confusione ha valori elevati lungo la diagonale e bassi altrove. Un modello che soffre di bias ha alti valori lungo le ultime colonne della matrice, poiché molte istanze vengono assegnate alle ultime classi. La sua utilità è dovuta al fatto che rende semplice vedere se il modello confonde una o più classi. Uno schema di una matrice di confusione è riportata in Figura 2.1, nella quale la riga o colonna i-esima rappresentano la classe A_i appresa dal modello. Intuitivamente, l'elemento N_{ij} indica quante volte la classe i-esima è stata "confusa" con la classe i-esima dal modello.

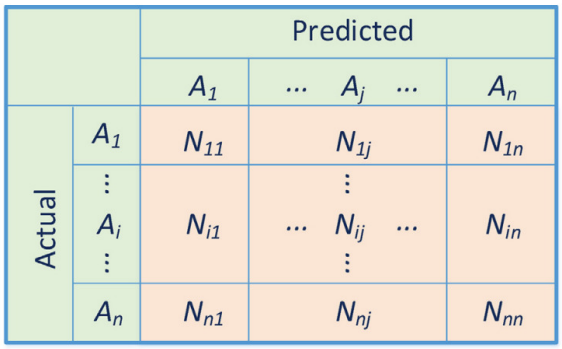

Figura 2.1: Schema di una matrice di confusione di un modello, con classi apprese $A_1, A_2, \dots A_n$. L'elemento N_{ij} rappresenta il numero di campioni effettivamente appartenenti alla classe A_i ma classificati dal modello come classe A_j. Tratto da [22].

2.4 Knowledge distillation

2.4.1 Loss function

Il modello M_t viene allenato minimizzando la loss function utilizzando algoritmi di discesa del gradiente, con i quali si ricavano i parametri aggiornati della rete. Una loss function molto utilizzata è la *cross-entropy loss*, definita come:

$$L_t^c(\mathbf{x}) = -\sum_{(\mathbf{x},y)\in D_t\cup K}\sum_{j=1}^{N_t} \mathbb{1}_{y=j}\log p_t^j(\mathbf{x}), \tag{2.11}$$

dove $\mathbb{1}_{y=j}$ è la funzione indicatrice, cioè:

$$\mathbb{1}_{y=j} = \begin{cases} 1, & \text{se } y=j, \\ 0, & \text{altrimenti.} \end{cases}. \tag{2.12}$$

Spesso la loss function è composta anche da un termine di *distillazione*. Il suo scopo è mitigare l'effetto del catastrophic forgetting "distillando" la nuova conoscenza con quella passata. Idealmente, essa permette di tramandare la conoscenza dal precedente modello in S_{t-1} a quello attuale in S_t [19, 72], riallenando i parametri con istanze elaborate in stati incrementali precedenti. Si definisce la *distillation loss* come

$$L_t^d(\mathbf{x}) = -\sum_{(\mathbf{x},y)\in D_t\cup K}\sum_{j=1}^{N_t-1} \phi_{t-1}^j(\mathbf{x})\log \phi_t^j(\mathbf{x}), \tag{2.13}$$

dove ϕ è una funzione di softmax con temperatura applicata ai punteggi grezzi predetti dalla rete. Per la classe j-esima allo stato S_t la funzione è definita nel seguente modo:

$$\phi_t^j(\mathbf{x}) = \frac{e^{o_t^j(\mathbf{x})/T}}{\sum_{l=1}^{N_t} e^{o_t^l(\mathbf{x})/T}}, \tag{2.14}$$

dove T è uno scalare detto *temperatura*. Se $T=1$ allora tale formulazione è equivalente all'Equazione 2.9. Un alto valore di temperatura porta i $\phi_t^j(\mathbf{x})$ a convergere allo stesso valore. Intuitivamente, aumentando la temperatura i valori di $\phi_t^j(\mathbf{x})$ più elevati vengono decrementati e quelli più bassi vengono incrementati. Ciò si può vedere matematicamente applicando il limite per $T\to\infty$:

$$\lim_{T\to\infty} \frac{e^{o_t^j(\mathbf{x})/T}}{\sum_{l=1}^{N_t} e^{o_t^l(\mathbf{x})/T}} = \frac{1}{N_t}. \tag{2.15}$$

Per T prossimi allo zero, invece, la softmax con temperatura accentua la differenza fra valori di $\phi_t^j(\mathbf{x})$ piccoli e grandi. Infatti:

$$\lim_{T\to 0} \frac{e^{o_t^j(\mathbf{x})/T}}{\sum_{l=1}^{N_t} e^{o_t^l(\mathbf{x})/T}} = \begin{cases} 1, & \text{se } o_t^j(\mathbf{x}) = \max\{o_t^1(\mathbf{x}), o_t^2(\mathbf{x}), \dots, o_t^{N_t}(\mathbf{x})\}, \\ 0, & \text{altrimenti.} \end{cases}. \tag{2.16}$$

Una rappresentazione grafica dell'effetto della softmax con temperatura sui punteggi grezzi predetti dalla rete è riportata in Figura 2.2.

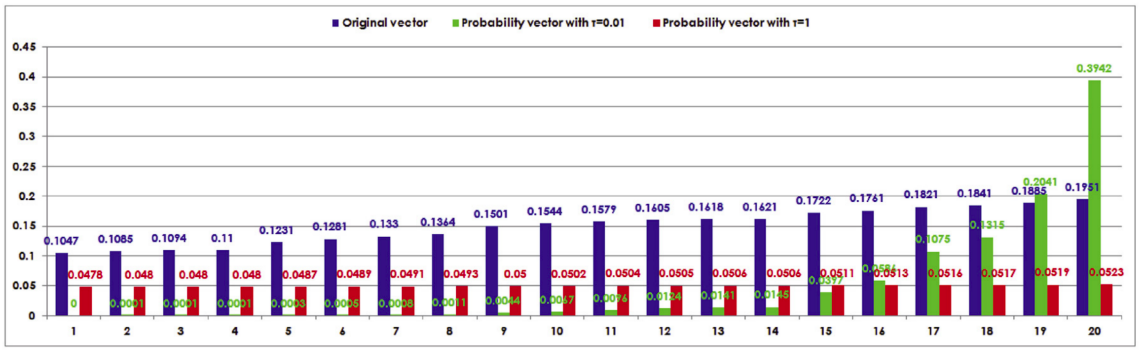

Figura 2.2: Valori ottenuti applicando la softmax con $T = 0.01$ e $T = 1$ a parità di $o_t^j(\mathbf{x}), j = 1, \ldots, 20$. Tratto da [37].

La distillation loss è presente nella maggior parte degli algoritmi FT [8, 55, 77, 92], uno dei tre approcci trattati nella tesi. Nel caso in cui vi è impiego della distillation loss, la loss function vale

$$L = \lambda L^d + (1 - \lambda)L^c, \tag{2.17}$$

dove $\lambda \in [0, 1]$ è un iperparametro che pesa l'influenza dei due termini. L'utilizzo della loss function così formulata è conosciuto in letteratura come *knowledge distillation* [8, 10, 19, 55, 72, 77, 92].

2.4.2 General KD e Task-wise KD

In letteratura sono presenti delle varianti della distillation loss in Equazione 2.13 [1, 16, 55, 77]. Ahn et al. [1] definiscono due tipi di knowledge distillation: la *General Knowledge Distillation* (GKD) [presente in 50, 92, 97] e la *Task-wise Knowledge Distillation* (TKD) [presente in 16, 55, 77]. Tale distinzione è dovuta al fatto che la GKD propaga il bias [92] causato dal precedente calcolo della cross-entropy [1]. Al contrario, ciò non avviene nella TKD [1].

Lo schema delle due KD viene riportato in Figura 2.3. La GKD applica la funzione di softmax con temperatura su tutte le classi $[1, N_t]$ e calcolando la distillation loss, quindi coincide con la definizione sopra riportata nell'Equazione 2.13. La TKD, invece, applica la funzione di softmax con temperatura su tutte le classi di ogni stato S_t. Per ognuno di essi viene calcolata una distillation loss e la distillation loss totale si ottiene sommando tutti i contributi. La differenza fra i due tipi di loss è esplicitabile tramite la *divergenza di Kullback-Leibler*:

$$L_t^{d,GKD}(\mathbf{x}) = D_{KL}\left(\phi_{t-1}^{1:t-1}(\mathbf{x}) || \phi_t^{1:t-1}(\mathbf{x})\right), \tag{2.18}$$

$$L_t^{d,TKD}(\mathbf{x}) = \sum_{s=1}^{t-1} D_{KL}\left(\phi_{t-1}^{s}(\mathbf{x}) || \phi_t^{s}(\mathbf{x})\right), \tag{2.19}$$

dove:

$$\phi_t^{1:s,j}(\mathbf{x}) = \frac{e^{o_t^j(\mathbf{x})/T}}{\sum_{l=1}^{N_s} e^{o_t^l(\mathbf{x})/T}}, \tag{2.20}$$

$$\phi_t^{s,j}(\mathbf{x}) = \frac{e^{o_t^j(\mathbf{x})/T}}{\sum_{l=N_{s-1}+1}^{N_s} e^{o_t^l(\mathbf{x})/T}}. \tag{2.21}$$

Si noti che tale formulazione implica la possibilità di calcolare N_{-1} per $s = 0$. Per semplicità si pone $N_{-1} = 0$. In questo modo, quando si ha $s = 0$ la sommatoria considera le classi $[1, N_0]$.

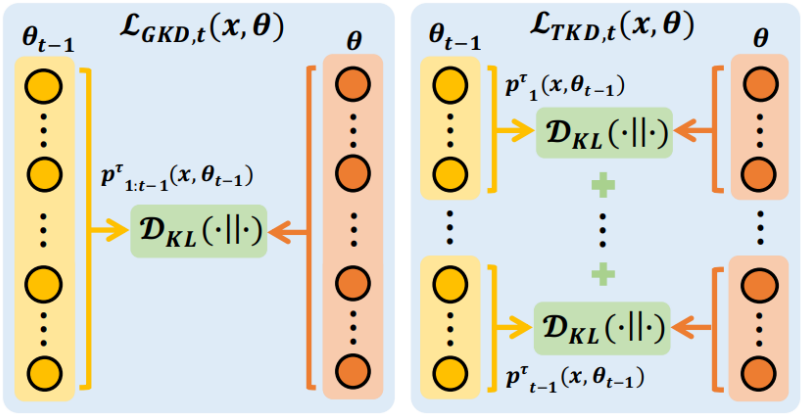

Figura 2.3: Illustrazione di $L_t^{d,GKD} = \mathcal{L}_{GKD,t}(\mathtt{x}, \boldsymbol{\theta})$ (sinistra) e $L_t^{d,TKD} = \mathcal{L}_{TKD,t}(\mathtt{x}, \boldsymbol{\theta})$ (destra), dove $p^{\tau}_{1:s,j} = \phi^{1:s,j}$ con temperatura τ. Tratto da [1].

2.5 Memoria

La memoria K svolge un ruolo chiave negli algoritmi di IL. Essa può essere impiegata per conservare istanze di classi passate oppure per ospitare i parametri aggiuntivi dovuti a una espansione del modello. Nel primo caso, per ogni classe j vengono salvate $|K_t^j|$ immagini nella memoria. Tale valore viene calcolato nel seguente modo:

$$|K_t^j| = \frac{|K|}{N_t}. \tag{2.22}$$

Dal momento che la memoria è limitata il numero di immagini salvate per ogni classe viene ridotto alla fine di ogni stato incrementale. Pertanto la rappresentazione delle classi passate degrada nel tempo e il modello sviluppa un bias verso quelle nuove. Un algoritmo di CIL ha il compito di decidere il metodo di selezione dei campioni da inserire in K. Si distinguono due principali approcci:

- Selezione casuale.
- Selezione tramite *herding*.

L'herding consiste nella scelta di immagini più rappresentative per una data classe [16, 77, 90]. In tal modo è possibile applicare la knowledge distillation per reimparare i parametri utili a classificare istanze simili [56]. Un algoritmo di CIL deve definire in che modo un esemplare è più rappresentativo di un altro. Alcuni algoritmi decidono di non dedicare la memoria disponibile alle immagini ma bensì a un altro tipo di dato. Ad esempio, l'algoritmo *IL2M* [8] sfrutta la memoria disponibile per conservare delle statistiche delle classificazioni passate con lo scopo di regolarizzare i punteggi.

2.6 Proprietà ricercate

Dopo aver introdotto le nozioni fondamentali del CIL, è necessario definire delle caratteristiche auspicate in un algoritmo in tale ambito, così da poterli valutare e mettere a confronto. Belouadah et al. in [10] hanno definito sei proprietà ricercate in un algoritmo di IL. I principali tipi di algoritmi di CIL trattati nella tesi si differenziano a partire dalle seguenti proprietà.

La *semplicità* misura la capacità di integrare nuove informazioni con modifiche minimali alla struttura del modello (in [10] ci si riferisce a tale proprietà con il termine *complexity*). La minima modifica ottenibile nel passaggio da uno stato incrementale all'altro è l'aumento della dimensione del layer di classificazione. La crescita spropositata del modello porta a un forte incremento del numero di parametri.

L'*indipendenza dalla memoria* misura l'abilità di lavorare senza la possibilità di conservare delle istanze di classi passate. Come trattato nel Capitolo 1, gli algoritmi che hanno forte indipendenza dalla memoria sono preferibili, nonostante da questi ci si aspetti una performance peggiore.

L'*accuratezza* misura la performance del modello nella classificazione. L'upper bound di tale proprietà consiste del modello *fully-trained*, cioè allenato con tutti i dati in un processo non incrementale. Questo modello rappresenta un limite superiore in quanto i suoi parametri vengono calcolati minimizzando la loss function ottenuta dalle istanze di tutte le classi che devono essere apprese. In altre parole, il modello fully-trained è ottimizzato per ridurre al minimo l'errore su tutti i dati disponibili. Due misure di accuratezza comuni sono la *top-1 accuracy* [1, 8, 33] e la *top-5 accuracy* [1, 7, 8, 33, 77]. La top-1 accuracy è la percentuale di istanze classificate correttamente durante la fase di testing. La top-5 accuracy si riferisce alla percentuale di volte in cui l'immagine corretta è inclusa tra le prime 5 previsioni di un modello di classificazione delle immagini. Metodi alternativi per misurare l'accuratezza sono definiti in [10, 33].

La *tempestività* misura il ritardo necessario fra l'arrivo dei nuovi dati e la loro integrazione nel modello incrementale. Come già visto nel Capitolo 1, sono molti i casi in cui si richiede un rapido aggiornamento della conoscenza.

La *plasticità* misura la capacità di trattare nuove classi significativamente diverse da quelle già apprese nel passato. La proprietà è legata al *concept drifting* [58, 96], un radicale cambiamento di contesto delle immagini da classificare. Tale fenomeno impedisce di sfruttare feature già apprese dal modello utilizzabili con le istanze delle nuove classi. Infatti. i parametri del modello vengono aggiornati in modo tale da apprendere le feature delle nuove classi a discapito di quelle passate. Ciò porta al CF, poiché il modello perde la capacità di individuare tratti particolari delle classi meno recenti [23].

La *scalabilità* misura l'abilità di apprendere un grande numero di classi. Nelle applicazioni, come la face recognition [92], si richiede di poter classificarne sino a decine di migliaia.

2.7 Dataset

Il problema del riconoscimento di immagini rientra nel paradigma del *supervised learning* [4] dal momento che richiede un insieme di immagini classificate manualmente da degli essere umani, detto *training set*. Infatti, la rete neurale necessita tali dati etichettati per calcolare i propri parametri ed essere in grado di classificare istanze aggiuntive, assenti nel training set. In tale contesto, i modelli di CIL vengono addestrati e valutati su set di dati comunemente utilizzati. Di seguito vengono elencati quelli nominati nella tesi.

Il dataset *ImageNet ILSVRC2012* contiene 10 000 000 immagini assegnate a più di 10 000 classi [79] di animali, oggetti, alimenti e veicoli. Venne utilizzato per la ImageNet Large Scale Visual Recognition Challenge 2012, vinta dalla celebre rete *AlexNet* [46]. Infatti, questo dataset è utilizzato per problemi di image classification classica, cioè senza step incrementali [21, 46, 87].

Due dataset spesso utilizzati sono *CIFAR-100* e *CIFAR-10*. CIFAR-100 contiene 60 000 immagini RGB 32×32 assegnate a 100 classi [92] appartenenti a diverse superclassi, come animali, arredi, insetti, paesaggi e veicoli. D'altro canto, CIFAR-10 è una versione più minuta di CIFAR-100. Esso contiene 6000 immagini RGB 32×32 assegnate a sole 10 classi [45] di animali e veicoli.

Il dataset *CORe50* [57] contiene sequenze video di 15 secondi di un oggetto in movimento con diversi background. I frame vengono suddivisi in 10 categorie (come smartphone, forbici, lampadine e pennarelli) e 50 classi. I video hanno un framerate di 20 fps. A causa di tale fluidità, è pratica comune sotto-campionare il framerate dei video del dataset [32, 33, 69].

Google Landmarks Dataset v2 (Landmark-v2) [91] è un dataset ideato per valutare le prestazioni di un modello a larga scala e ad alta precisione. Esso contiene immagini di paesaggi naturali ed oggetti artificiali. Contiene oltre 5 milioni di immagini e 200 000 label.

2.8 Tipi di algoritmi

Belouadah et al. [10] hanno individuato tre principali tipi di algoritmi di CIL. Questi si differenziano nell'approccio in cui tentano di integrare nuova conoscenza pur mantenendo quella passata, contrastando gli effetti del CF. In più, essi soddisfano diverse proprietà riportate nella Sezione 2.6. I tre tipi di algoritmi sono analizzati più nel dettaglio nelle Sezioni 2.8.1, 2.8.2 e 2.8.3. Come anticipato nel Capitolo 1, i tre tipi di algoritmi sono:

- Model-Growth

- Fixed-Representation
- Fine-Tuning

2.8.1 Model-Growth

Gli algoritmi che seguono l'approccio MG mirano ad aumentare la grandezza del modello per introdurre nuova conoscenza. Risulta chiaro che la complessità cresce nel tempo dal momento che parametri vengono aggiunti al modello. Gli algoritmi MG sono particolarmente efficaci contro il CF, classificando correttamente classi passate e recenti. In tale contesto, l'obiettivo è minimizzare il numero di nuovi parametri introdotti per limitare la crescita di complessità, l'uso di memoria e il tempo di allenamento.

2.8.2 Fixed-Representation

Gli algoritmi di tipo FR non aggiornano la deep representation del modello a ogni stato incrementale. Ciò significa che i pesi e parametri coinvolti nell'estrazione delle features non cambia nel tempo, con la speranza di poterli riutilizzare per diverse classi. Li e Hoiem [54] hanno descritto un approccio chiamato Feature Extraction (FE), il quale appare molto simile al FR. La sottile differenza tra i due è che i primi congelano anche i parametri del classificatore, a differenza dei secondi, dove ciò non è obbligatorio. FE ottiene scarsi risultati [77], dal momento che un algoritmo che fissa i propri parametri e non si adatta alle nuovi classi è destinato a degradare in accuratezza nel tempo. L'approccio FR risulta vincente nel caso in cui la presenza di memoria non è contemplata e la celerità nell'apprendimento è fondamentale. È possibile raggiungere alti livelli di accuratezza nel momento in cui il modello riutilizza efficacemente le feature imparate in S_0.

2.8.3 Fine-Tuning

I metodi basati sul FT fanno utilizzo della knowledge distillation per combattere il catastrophic forgetting. L'algoritmo simbolo di questo approccio è *Learning without Forgetting* (*LwF*) [54], da cui molti metodi FT si sono poi ispirati. Una sua versione migliorata e più recente è *Incremental Classifier and Representation Learning* (*iCaRL*) [77]. I metodi basati sul FT sono utilizzabili quando è disponibile memoria per le classi passate, le immagini variano di molto fra gli stati incrementali e quando il modello non è necessario nel breve termine. Però, è importante notare che conservare una grande quantità di classi passate in memoria va contro gli obiettivi del CIL [10, 77], come indicato nella Sezione 2.6.

Capitolo 3

Model-Growth

L'approccio MG può portare a ottimi risultati se la crescita del modello è contemplata [10]. Ciò deve avvenire minimizzando l'introduzione di nuovi parametri per evitare che la complessità aumenti in modo incontrollato. In sostanza, è importante garantire che il modello mantenga un'alta precisione senza diventare eccessivamente esigente in termini di risorse spaziali e temporali. La tesi affronta i seguenti algoritmi MG:

- *Tree-CNN* [78] di Roy et al. (2020)
- *DER* [95] di Yan et al. (2021)

3.1 *Tree-CNN*

Tree-CNN è una rete di CNN che cresce gerarchicamente all'introduzione di nuove classi. Essa è composta da una struttura gerarchica ad albero, nella quale ogni nodo interno è una CNN, mentre a ogni nodo esterno (foglia dell'albero) corrisponde una classe che il modello è in grado di riconoscere. Un nodo interno accetta una immagine in input, e a ciascun output della CNN corrisponde un'altra CNN oppure a una classe appresa dal modello. Nel primo caso la classificazione continua attraverso la nuova CNN contenuta nel nodo interno figlio. Nel secondo caso, invece, l'algoritmo assegna all'immagine la classe corrispondente al nodo esterno, concludendo la classificazione. Intuitivamente, una volta addestrato il modello, un nodo vicino alla root distingue le immagini fra ampie categorie di classi (e.g. classifica se una immagine contiene un animale o un oggetto), mentre un nodo vicino alle foglie è specializzato a individuare la classe corretta all'interno della categoria stessa (e.g. riconoscere se un animale è un vertebrato o un invertebrato).

3.1.1 Inferenza e crescita del modello

Inferenza

Come introdotto nella Sezione 3.1, il modello di *Tree-CNN* è composto da una struttura gerarchica ad albero. I nodi interni contengono una CNN allenata per classificare l'input in uno dei suoi nodi figli, mentre le foglie contengono le classi apprese. Un esempio di modello è raffigurato in Figura 3.1. L'immagine da classificare viene data in input alla root della struttura, dove avviene la prima classificazione. La root passa l'istanza al nodo figlio che, se nodo interno, la classificherà a sua volta in uno dei figli. Ciò avviene ricorsivamente finché non si raggiunge una foglia. Ogni foglia è in corrispondenza biunivoca con una delle classi apprese. L'algoritmo di inferenza viene formalizzato nell'Algoritmo 1.

Livello di una *Tree-CNN*

Il livello di una *Tree-CNN* corrisponde all'altezza dell'albero. Il livello di un nodo è la sua distanza dalla root. Per esempio, la *Tree-CNN* in Figura 3.1 è a 2 livelli dal momento che il livello della root è 0, quello dei nodi interni (*branch node* in figura) è 1 e quello delle foglie (*leaf node* in figura) è 2.

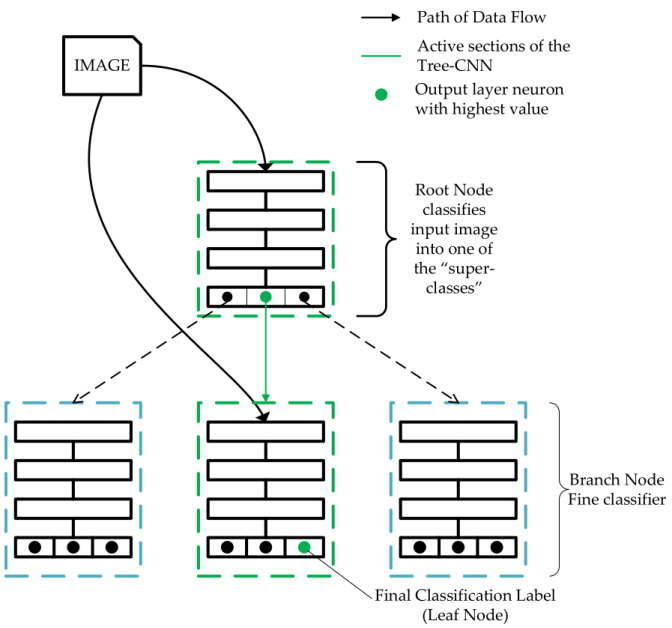

Figura 3.1: Esempio di modello *Tree-CNN* a 2 livelli. Nello stato in figura il modello riconosce 9 classi dal momento che ha 9 foglie. Tratto da [78].

Algorithm 1 Inferenza

x = Immagine di input, $node$ = Root del modello.
procedure ClassPredict(x,$node$)
    $count$ = number of children of $node$
    if $count == 0$ **then**
        $label$ = class label of $node$
        return $label$
    else
        $next = EvaluateNode(x, node)$ ▷ $EvaluateNode(x, node)$ ritorna il nodo figlio con output maggiore.
        return $ClassPredict(x, next)$
    end if
end procedure

Algoritmo di apprendimento per una *Tree-CNN* a 2 livelli

Si analizza ora l'apprendimento di S_t in S_{t-1} per $t > 0$. Di conseguenza, si assume che il modello sia in grado di riconoscere un certo numero di classi. L'algoritmo deve permettere al modello di imparare $P_t = M$ nuove classi. Si selezionano I immagini per classe dal nuovo training set da utilizzare per l'addestramento, che viene dunque effettuato con $I \times M$ istanze. Per ogni immagine si predice una classe a partire dalla root come indicato in Alg 1, producendo un tensore di 3-dimensionale $O^{K \times M \times I}$, dove K è il numero di figli del nodo radice, M e I come definiti precedentemente. L'elemento $O(k, m.i)$ rappresenta l'output associato al nodo figlio k dell'i-esima immagine appartenente alla classe m, con $k \in [1, K]$, $m \in [1, M]$ e $i \in [1, I]$. La media degli output sulle I immagini viene espressa come

$$O_{avg}^{K \times M} = \sum_{i=1}^{I} \frac{O(k, m, i)}{I}. \tag{3.1}$$

Si definisce *softmax likelihood matrix* $L^{K \times M}$, dove l'elemento $L(k, m)$ vale

$$L(k, m) = \frac{e^{O_{avg}^{K \times M}}}{\sum_{k=1}^{K} O_{avg}^{K \times M}}. \tag{3.2}$$

Dopo aver calcolato tali matrici, si genera una lista ordinata S da $L^{K\times M}$ aventi le seguenti proprietà:

- La lista S ha M elementi, ognuno corrispondente a una classe.
- Ogni elemento $S[i]$ ha i seguenti attributi:
  - $S[i].label$ è la label della nuova classe.
  - $S[i].value = [v_1, v_2, v_3]$ sono i tre più grandi valori della matrice $L^{K\times M}$ tali per cui $v_1 \geq v_2 \geq v_3$.
  - $S[i].node = [n_1, n_2, n_3]$ sono i nodi figli in output associati a $[v_1, v_2, v_3]$.
- La lista S è ordinata in valore decrescente del primo elemento di $S[i].value$, cioè $S[i].value[1]$.

Tale ordinamento assicura che le classi con un alto valore di verosimiglianza in $L^{K\times M}$ siano aggiunte per prima al modello.

Dopo la costruzione della lista, si osserva $S[1]$. Siano $\alpha > \beta > 0$ iperparametri che rappresentano dei valori di soglia.

1. Se $v_1 - v_2 > \alpha$ allora la nuova classe ha una forte somiglianza con il nodo figlio n_1. Dunque la nuova classe viene aggiunta come nodo figlio di n_1, stabilendo una gerarchia fra le due.
2. Se $v_1 - v_2 < \alpha$ e $v_1 - v_2 > \beta$ allora ci sono 2 nodi figli somiglianti alla nuova classe. In tal caso si combinano n_1 e n_2 per formare un nuovo nodo figlio e si aggiunge la nuova classe a esso.
3. Se $v_1 - v_2 < \alpha$ e $v_1 - v_2 < \beta$ allora la nuova classe non assomiglia a nessuno dei suoi nodi figli. In tal caso, la rete si espande orizzontalmente aggiungendo una nuova classe come nodo figlio. Questa sarà una foglia dell'albero dal momento che viene aggiunta singolarmente.

Con il passare degli stati incrementali i sotto-alberi con più nodi figli diventano "più pesanti": le nuove classi tendono ad avere un alto valore della softmax likelihood in Equazione 3.2 per nodi con un più grande numero di nodi figli. Al fine di prevenire tale sbilanciamento, è possibile definire il numero massimo di nodi figli che un nodo può avere. Dal punto di vista matematico, si pone $L(k, m) = 0$ per il nodo figlio k pieno e $\forall m \in [1, M]$. Per come è ordinato S, il modello non assegna ai nodi pieni alcun figlio. Dopo aver osservato il primo elemento di S e averlo aggiunto all'albero come sopra discusso, lo si rimuove e non si considera più la sua classe associata nel calcolo di $O^{K\times M\times I}$ e $L^{K\times M}$. Si ripete tale procedura iterativamente finché tutte le nuove classi non vengono assegnate a una locazione sotto la root.

Quando l'assegnamento della posizione alle nuove classi è conclusa, la root, i nodi nuovi e i nodi modificati vengono allenati con le istanze dello stato incrementale attuale. L'allenamento della root a ogni S_t è necessario in quanto essa deve sempre eseguire la prima classificazione, come discusso nell'Algoritmo 1. Se la classificazione alla root è errata allora qualunque classificazione avvenga nei nodi figli porterà a una inferenza errata.

Algoritmo di apprendimento per una *Tree-CNN* a n livelli

Per creare *Tree-CNN* a n livelli è possibile riapplicare l'algoritmo descritto in 3.1.1 per i nodi del livello sottostante. È possibile definire la massima profondità dell'albero (cioè il livello massimo) e il numero massimo di figli di un nodo.

3.1.2 Esperimenti

In [78] l'algoritmo viene testato su CIFAR-100. Le 100 classi del dataset vengono divise casualmente in 10 gruppi da 10 classi ciascuna. Ogni gruppo viene introdotto nel modello in modo incrementale. La *Tree-CNN* viene inizializzata con un solo livello essendo composta dalla root e dalle 10 foglie associate alle prime 10 classi. Esse vengono apprese dalla CNN nella root in S_0. Gli altri 9 gruppi vengono appresi come descritto in 3.1.1. La massima profondità dell'albero è 2, $\alpha = 0.1$ e $\beta = 0.1$. Il numero massimo di figli per nodo è fissato a 5, 10 e 20 per le tre rispettive varianti *Tree-CNN-5*, *Tree-CNN-10* e *Tree-CNN-20*. La grandezza delle tre varianti viene confrontata con una Network B. Essa è una VGG-net [83] con 11 layer, 4 blocchi convoluzionali, ciascun blocco ha 2 insiemi di kernel convoluzionali 3×3. I risultati sono riportati nella Figura 3.3.

Figura 3.2: La Figura 3.3 riporta la grandezza delle tre *Tree-CNN* normalizzate rispetto alla rete B in funzione del numero di classi apprese. La Figura 3.4 riporta l'accuratezza su CIFAR-100 degli algoritmi discussi in funzione del numero di classi apprese. Tratte da [78].

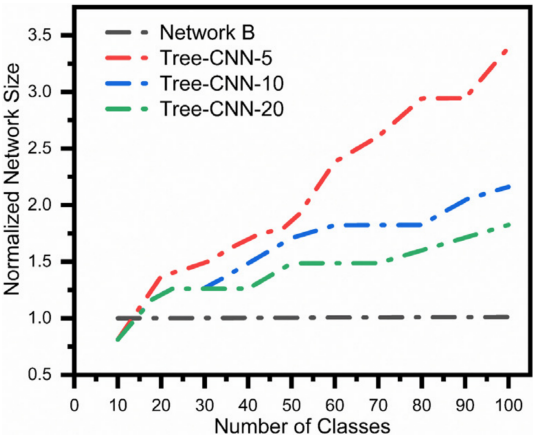

Figura 3.3

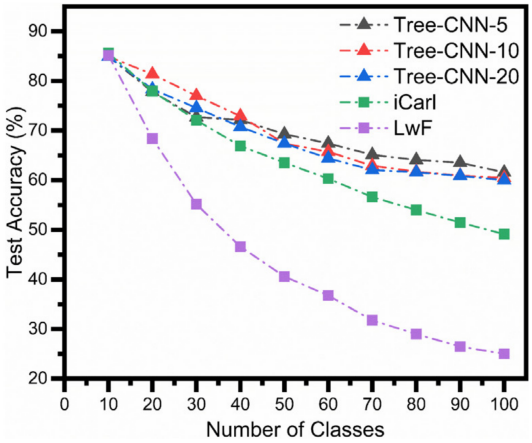

Figura 3.4

Le tre varianti di *Tree-CNN* vengono confrontate con 2 metodi FT: *iCaRL* [76], *LwF* [55]. Il protocollo di valutazione è analogo a quello presentato in [76]. I risultati vengono riportati in Tabella 3.1 e Figura 3.4.

Tabella 3.1: Accuratezza sulle 100 classi di CIFAR-100. Il migliore risultato è evidenziato in **grassetto**. Tratto da [78]

Model	Final Test Accuracy	Average Test Accuracy
Tree-CNN-5	**0.6157**	**0.6985**
Tree-CNN-10	0.6046	0.6953
Tree-CNN-20	0.5999	0.6849
iCaRL	0.4911	0.6410
LwF	0.25	0.4449

La Figura 3.4 e la Tabella 3.1 evidenziano che tutte le varianti di *Tree-CNN* ottengono una accuratezza migliore dei metodi FT *LwF* e *iCaRL*. Secondo Roy et al. [78] ciò è dovuto alla struttura gerarchica del modello, grazie alla quale è possibile allenare miratamente solo alcuni suoi nodi a ogni stato incrementale. Se solo alcuni nodi vengono aggiornati con un nuovo set di classi, allora è probabile che i nodi che si occupavano di riconoscere classi passate manterranno i parametri appresi precedentemente. Di conseguenza, l'accuratezza rimane alta nel tempo per le classi di tutti i stati incrementali. Nonostante ciò, la rete continua a crescere nel tempo, dunque è necessario introdurre una memoria per ospitare i nuovi nodi.

Il modello cresce in modo tale da raggruppare classi con feature comuni. Gli autori suggeriscono di investigare ulteriormente tale aspetto, poiché con *Tree-CNN* è possibile classificare gerarchicamente un dataset e raggruppare classi simili. Un esempio è mostrato in Figura 3.5.

3.2 *DER*: Dynamically Expandable Representation

DER è un modello composto da una rete di feature extractor e un classificatore lineare. Ad ogni step incrementale viene aggiunto un nuovo feature extractor, mantenendo inalterati i parametri di quelli aggiunti precedentemente. Le feature generate da tutti i feature extractor vengono concatenate e date in input al classificatore per la predizione. Il feature extractor appena aggiunto alla rete viene allenato sui nuovi dati e sulla memoria K, come descritto dalla Equazione 2.3.

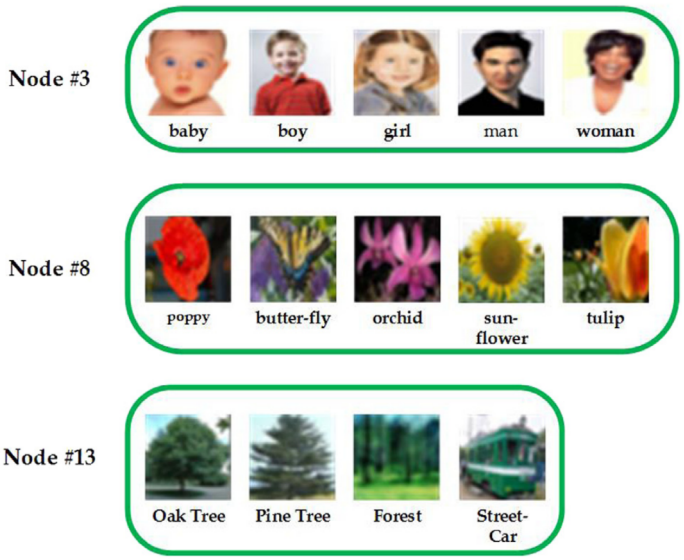

Figura 3.5: Esempi di gruppi di classi formati all'introduzione di nuove classi in *Tree-CNN-10* a gruppi di 10 di CIFAR-100. Per esempio, dalle classi baby, boy, girl, man e woman vengono estratte feature molto simili, dunque vengono raggruppate sotto il terzo nodo. Si noti la scarsa qualità delle immagini, caratteristica del dataset a cui appartengono. Tratto da [78].

3.2.1 Representation Learning e Dynamical Expansion

L'addestramento del modello allo stato S_t è composto da due fasi:

1. Representation Learning Stage: Il modello aggiunge un nuovo feature extractor e lo allena sul dataset definito nella Equazione 2.3. Yan et al. ritengono che in tal modo si possa ottenere un compromesso fra stabilità e plasticità e constrastare il CF [95].

2. Classification Learning Stage: Il classificatore lineare C_t viene allenato sul dataset definito nella Equazione 2.3.

Representation Learning Stage

La rete di feature extractor viene chiamata *Super Feature Extractor* (SFE). Il SFE in S_t è indicato da Φ_t:

$$\Phi_t \colon X_t \to \mathbb{R}^d. \tag{3.3}$$

Essa è ricavato a partire dalla SFE in S_{t-1} e dal nuovo feature extractor F_t. Data una immagine $\mathbf{x}$, la feature $\mathbf{u}$ estratta da Φ_t è ottenuta dalla seguente concatenazione:

$$\mathbf{u} = \Phi_t(\mathbf{x}) = [\Phi_{t-1}(\mathbf{x}), F_t(\mathbf{x})]. \tag{3.4}$$

Il classificatore lineare C_t è formulato in modo analogo all'Equazione 2.8, al cui vettore di punteggi $\mathbf{o}_t$ viene applicata la funzione di softmax definita in Equazione 2.9 per calcolare il vettore di probabilità condizionate

$$p_{C_t}(\mathbf{y}|\mathbf{x}) = softmax(C_t(\mathbf{u})). \tag{3.5}$$

La classe predetta $\hat{y}$ per l'immagine $\mathbf{x}$ è calcolata nel seguente modo:

$$\hat{y} = \arg\max y_i \{p_{C_t}(y_i|\mathbf{x})\}_{i=1}^{N_t}. \tag{3.6}$$

I parametri di C_t per le feature passate sono ereditate da C_{t-1}, mentre i nuovi vengono inizializzati casualmente.

L'apprendimento in S_t avviene minimizzando la cross entropy L_{C_t} (Equazione 2.11) e una loss ausiliaria $L_{C_t^a}$ che opera su F_t. Si introduce un classificatore C_t^a che predice le probabilità condizionate $p_{C_t^a}(\mathbf{y}|\mathbf{x})$, le quali valgono

$$p_{C_t^a}(\mathbf{y}|\mathbf{x}) = softmax(C_t^a(\mathbf{u})). \tag{3.7}$$

Tale classificatore viene introdotto per incoraggiare il modello ad apprendere feature per discriminare le classi passate con quelle nuove. Infatti si ha che

$$|p_{C_t^a}(\mathbf{y}|\mathbf{x})| = P_t + 1, \tag{3.8}$$

cioè le label predicibili da C_t^a sono quelle associate alle nuove P_t classi e una label fittizia corrispondente alle P_{t-1} classi passate. La loss $L_{C_t^a}$ è definita nel seguente modo:

$$L_{C_t^a} = -\frac{1}{|\tilde{D}_t|}\sum_{i=1}^{|\tilde{D}_t|}\log(p_{C_t^a}(y = y_i|\mathbf{x}_i), \tag{3.9}$$

dove $\mathbf{x}_i$ è l'immagine i-esima di $\tilde{D}_t = D_t \cup K$ in S_t e y_i è la sua classe. La *expandable representation loss* L_{ER} è una combinazione lineare delle loss definite nelle Equazioni 2.11 e 3.9:

$$L_{\text{ER}} = L_{C_t} + \lambda_a L_{C_t^a}, \tag{3.10}$$

dove λ_a è un iperparametro che controlla l'effetto del classificatore ausiliario. In S_0 si ha che $\lambda_a = 0$ poiché non vi sono ancora classi passate.

Dynamical Expansion

Il SFE viene espanso dinamicamente con lo scopo di compattare le dimensioni del modello. Infatti, una volta aggiunto F_t, questo viene ridotto a F_t^P. Fra il layer convoluzionale $l-1$ e l di F_t viene introdotta una maschera [81] $\mathbf{m}_l \in \mathbb{R}^{c_l}$, dove $m_l^i \in [0,1]$ e c_l è il numero di canali del layer l. Data l'immagine $\mathbf{x}$ in input al modello, l'input al layer l viene denominato $\mathbf{f}_l$. L'input mascherato $\mathbf{f}_l'$ è definito come

$$\mathbf{f}_l' = \mathbf{f}_l \odot \mathbf{m}_l, \tag{3.11}$$

dove $\odot$ è la moltiplicazione membro a membro a livello di canale. Dunque, $\mathbf{f}_{l,i}'$ è l'input del canale i-esimo del layer l. L'idea è quella di *inibire delle sinapsi* [60], cioè attivare e disattivare degli output di canali ad ogni layer.

Per mantenere ogni elemento di $\mathbf{m}_l$ compreso fra 0 e 1, esso viene calcolato come segue:

$$\mathbf{m}_l = \sigma(s\mathbf{e}_l), \tag{3.12}$$

dove $\mathbf{e}_l$ sono i parametri di masking apprendibili dal modello, σ è la funzione sigmoide e s è un fattore di scala che controlla la ripidità della funzione. Tale meccanismo è chiamato *gating*. In S_t il dataset viene diviso in B batches. Il fattore di scala s viene calcolato durante l'allenamento come segue:

$$s = \frac{1}{s_{\max}} + \left(s_{\max} - \frac{1}{s_{\max}}\right)\frac{b-1}{B-1}, \tag{3.13}$$

dove b indica l'indice del batch considerato e $s_{\max} \gg 1$ è controlla la plasticità degli elementi del layer [81]. Infatti, se $s_{\max}$ è vicino ad 1, l'Equazione 3.13 diventa una regolare sigmoide. Invece, se $s_{\max}$ è grande, l'Equazione 3.13 si approssima alla funzione gradino.

Si aggiunge una *sparsity loss* L_S per ridurre il numero di parametri con un abbassamento delle performance minimale [95]. Per far ciò, si pongono i moduli di $\mathbf{m}_l$ e di $\mathbf{m}_{l-1}$ al numeratore, che verranno minimizzati assieme a L_S. Essa è formulata come

$$L_S = \frac{\sum_{l=1}^{L} K_l|\mathbf{m}_l||\mathbf{m}_{l-1}|}{\sum_{l=1}^{L} K_l}, \tag{3.14}$$

dove L è il numero di layer convoluzionali, K_l è la dimensione del kernel nel layer convoluzionale l. L'algoritmo pone $|\mathbf{m}_0| = 3$. Una panoramica completa dell'algoritmo è riportata in Figura 3.6.

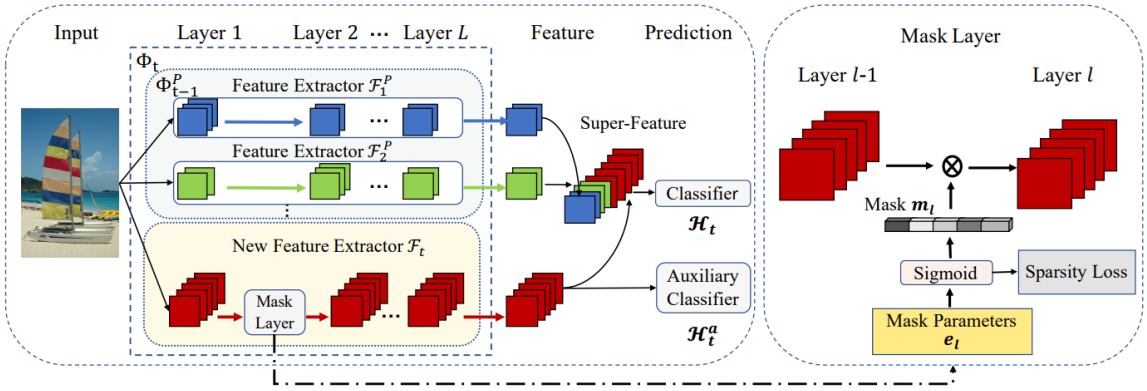

Figura 3.6: Panoramica di *DER*. In S_t, il modello è composto da un DFE Φ_t e un classificatore C_t ($\mathcal{H}_t$ in figura), dove Φ_t è costruito espandendo il DFE esistente Φ_{t-1} con un nuovo feature extractor F_t. L'algoritmo utilizza anche un classificatore C_t^a ($\mathcal{H}_t^a$ in figura) ausiliario che discrimina le classi passate e nuove. Infine, nella Dynamical Expansion vengono apprese delle maschere, con lo scopo di ridurre la dimensione del modello. Tratto da [95].

Loss function

La loss function minimizzata dall'algoritmo è

$$L_{\text{DER}} = L_{C_t} + \lambda_a L_{C_t^a} + \lambda_s L_S, \tag{3.15}$$

dove λ_s è un iperparametro che controlla l'espansione del modello descritta nella Sezione 3.2.1. Il termine L_{C_t} è la cross entropy definita nella Equazione 2.11, impiegata per addestrare il modello sulle nuove istanze in S_t e sulla memoria K. In aggiunta, il termine $L_{C_t^a}$ aiuta a distinguere se delle feature sono tipiche di una nuova particolare classe o di una appartenente a degli step incrementali precedenti. Infine, il termine L_S serve per apprendere le maschere definite in Equazione 3.12. Queste devono essere tali per cui l'inibizione dei parametri non degradi troppo le prestazioni.

Classifier Learning Stage

Dopo aver il rappresentation learning stage in Sezione 3.2.1, i parametri del classificatore vengono reinizializzati casualmente e riallenati su $\tilde{D}_t = D_t \cup K$. Per l'addestramento viene utilizzata solamente la cross entropy loss in Equazione 2.11 assieme la softmax con temperatura dell'Equazione 2.14.

3.2.2 Esperimenti

Yan et al. [95] sperimentano l'algoritmo sul dataset CIFAR-100 [45]. Vengono seguiti due protocolli:

- CIFAR-100-B0 [45]: Il modello viene allenato su tutte le 100 classi in 5, 10, 20 o 50 step incrementali, con memoria contenente 2000 esemplari.
- CIFAR-100-B50 [38]: Il modello parte avendo già appreso 50 classi in S_0 e viene allenato sulle 50 rimanenti in 2, 5 e 10 step incrementali, con memoria contenente 20 esemplari per classe.

La misura di accuratezza utilizzata è la *top-1 average incremental accuracy*. Essa consiste nella media delle *top-1 accuracy* ottenute in 3 esperimenti. Nell'esperimento, *DER* viene implementato in PyTorch e adotta una ResNet-18 come F_t. L'algoritmo viene testato assieme a *iCaRL* [77], *UCIR* [38], *BiC* [92], *WA* [97] e *PODNet* [24]. Le Tabelle 3.2 e 3.3 riportano i risultati dell'esperimento.

Dalla Tabella 3.2 è evidente che le due varianti di *DER* performano meglio degli altri metodi di un grande margine. Inoltre, all'aumentare del numero di step incrementali aumenta anche il margine fra *DER* e gli altri metodi. Il margine fra *DER* e *DER* (w/o P) è trascurabile, nonostante la riduzione del numero di parametri: ciò dimostra l'efficacia del metodo di riduzione di *DER*. Risultati simili si ottengono dalla Tabella 3.3. Da entrambe le tabelle si dimostra che la crescita del modello è legata al mantenimento di alte performance fra le varie fasi dell'esperimento.

Tabella 3.2: Valori del numero di parametri e delle *top-1 average incremental accuracy* (3 esperimenti) su CIFAR-100-B0. Il modello *DER* (w/o P) fa riferimento all'algoritmo senza riduzione dei feature extractor. Il migliore risultato è evidenziato in **grassetto**. Tratto da [95].

Model	5 steps		10 steps		20 steps		50 steps	
	Params	Avg	Params	Avg	Params	Avg	Params	Avg
Fully-trained	11.2	0.8040	11.2	0.8041	11.2	0.8149	11.2	0.8174
iCaRL	11.2	0.7114	11.2	0.6527	11.2	0.6120	11.2	0.5608
UCIR	11.2	0.6277	11.2	0.5866	11.2	0.5817	11.2	0.5686
BiC	11.2	0.7310	11.2	0.6880	11.2	0.6648	11.2	0.6209
WA	11.2	0.7281	11.2	0.6946	11.2	0.6733	11.2	0.6432
PODNet	11.2	0.6670	11.2	0.5803	11.2	0.5397	11.2	0.5119
DER (w/o P)	33.6	**0.7680**	61.6	**0.7536**	117.6	**0.7409**	285.6	**0.7241**
DER	**2.89**	0.7555	**4.96**	0.7464	**7.21**	0.7398	**10.15**	0.7205

Tabella 3.3: Valori del numero di parametri e delle *top-1 average incremental accuracy* (3 esperimenti) su CIFAR-100-B50. Il modello *DER* (w/o P) fa riferimento all'algoritmo senza riduzione dei feature extractor. Il migliore risultato è evidenziato in **grassetto**. Tratto da [95].

Model	2 steps		5 steps		10 steps	
	Params	Avg	Params	Avg	Params	Avg
Fully-trained	11.2	0.7722	11.2	0.7989	11.2	0.7991
iCaRL	11.2	0.7133	11.2	0.6506	11.2	0.5859
UCIR	11.2	0.6721	11.2	0.6428	11.2	0.5992
BiC	11.2	0.7247	11.2	0.6662	11.2	0.6025
WA	11.2	0.7143	11.2	0.6401	11.2	0.5786
PODNet	11.2	0.7130	11.2	0.6725	11.2	0.6404
DER (w/o P)	22.4	**0.7461**	39.2	**0.7321**	67.2	**0.7281**
DER	**3.90**	0.7457	**6.13**	0.7260	**8.79**	0.7245

Il metodo trattato in [95] dimostra l'efficacia dell'approccio MG. L'algoritmo riesce a mantenere alte performance nel tempo aumentando il numero di parametri. La versione *DER* senza riduzione non riesce a limitare la crescita del modello. D'altra parte, la versione *DER* con riduzione ha un numero di parametri inferiore rispetto agli altri algoritmi trattati, pur mantenendo un'accuratezza molto simile al metodo senza riduzione. Considerando i risultati ottenuti, si può concludere che il metodo con riduzione è in generale preferibile rispetto alla versione senza riduzione.

Capitolo 4

Fixed-Representation

Gli algoritmi FR sono i meno presenti in letteratura e possono essere visti come una semplice variante di quelli FT. Come discusso nella Sezione 2.8.2, essi sono caratterizzati dal fatto di congelare i pesi e parametri della rappresentazione profonda a ogni S_t. Di conseguenza, la loro plasticità e accuratezza sono limitate. Nonostante questo, l'approccio FR ottiene i risultati migliori in contesti in cui la memoria non è disponibile [10]. La tesi affronta i seguenti algoritmi FR:

- *DeeSIL* [7] di Belouadah e Popescu (2018)
- *Deep-SLDA* [33] di Hayes e Kanan (2020)

4.1 *DeeSIL*: Deep-Shallow Incremental Learning

DeeSIL è un'adattamento del transfer learning all'incremental learning. Il transfer learning è una tecnica che consiste nel trasferire la conoscenza da un modello *source* a un altro modello chiamato *target* [89, 98]. Il modello target differisce dal modello source perché ha un diverso insieme di istanze o una diversa distribuzione di probabilità di queste [89]. In altre parole, il modello target utilizza la conoscenza appresa dal modello source per migliorare la propria capacità di apprendimento. In questo caso, il modello in S_{t-1} rappresenta la source e il modello in S_t il target. Lo scopo è trasferire la conoscenza dal modello passato a quello presente.

4.1.1 Deep Features Extractor e classificatori

L'algoritmo disaccoppia due parti fondamentali:

- *Deep Features Extractor* (DFE), il quale viene fissato.
- Linear shallow classifier, allenati utilizzando la deep representation.

Lo schema dell'algoritmo viene illustrato in Figura 4.1. Il DFE è un feature extractor analogo a quello specificato dalla Definizione 2.4. Un linear shallow classifier è un classificatore come riportato nella Definizione 2.6. Il termine "shallow" rimarca il fatto che il classifier non esegue alcuna feature extraction [71] e che esso non ha più layer come un modello di DL [12]. Inoltre, tali classifier sono lineari, cioè separano linearmente un insieme di istanze in due gruppi [82], come una SVM lineare o il Perceptron [62]. L'algoritmo si rivela tempestivo in quanto è necessario allenare solo i classifier a ogni stato incrementale. L'algoritmo segue l'approccio FR in quanto la deep representation viene fissata dopo S_0. Gli autori ritengono che i classifier indipendenti compensino la struttura rigida del modello.

Le feature di una immagine vengono estratte da un DFE. L'output del DFE è dato in input a dei classifier binari. Questi vengono allenati utilizzando delle feature positive F^i e delle feature negative F^N secondo una classificazione *One vs All*. Le feature positive di una istanza sono estratte dal DFE. Data la classe i-esima, le sue feature negative $F^N(i)$ sono una rappresentazione delle feature estratte dalle classi $j = 1, \ldots, N_t, j \neq i$. Data una istanza, la classificazione One vs All permette di massimizzare la separazione [82] tra le sue feature F^i e le feature di istanze di altre classi $F^N(i)$. Le features F^N sono conservate all'interno della memoria e vengono selezionate da un *negative selector* (NS). Esse sono

comuni a tutti i classifier di S_t. Naturalmente, durante l'allenamento di C_i, i rappresentanti della classe i-esima in F^N non vengono considerati. I cliassifier binari vengono implementati con delle *Support Vector Machine* (SVM) lineari. L'algoritmo di aggiornamento dello stato S_t è mostrato nell'Algoritmo 2.

Algorithm 2 Train S_{t+1}

Extract features for the new P_{t+1} classes.
Update the pool of negatives F_t using the NS component
Train P_{t+1} shallow classifier

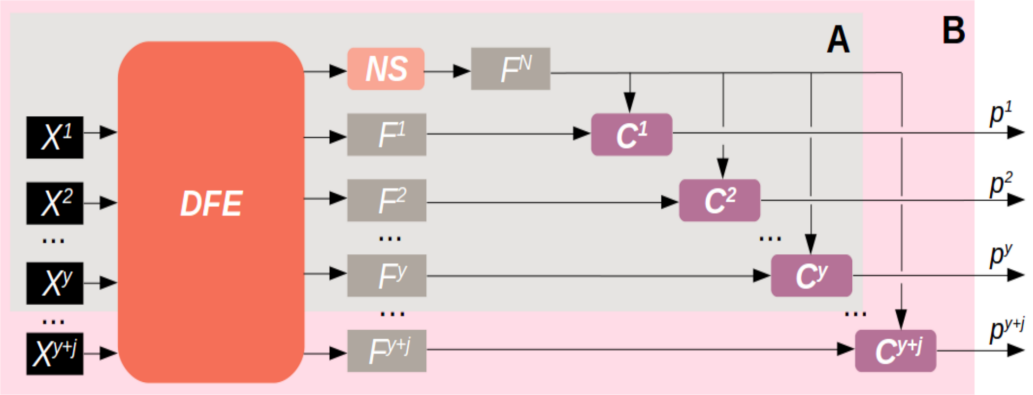

Figura 4.1: Schema di *DeeSIL*. A e B rappresentano due stati incrementali del sistema, S_t e S_{t+1}. $y = N_t$ e $j = P_{t+1}$. p_i è lo score predetto per la classe i-esima dal classifier C_i. Tratto da [7].

4.1.2 Addestramento del DFE

DeeSIL implementa un DFE fissato. È necessario definire come viene allenato il DFE in S_0. Per valutare l'effetto della quantità di dati di training e la sua somiglianza con i dati di test, il paper propone tre varianti:

- *IN100*: il DFE viene allenato con 100 classi ImageNet.
- *IN1000*: il DFE viene allenato con 1000 classi ImageNet.
- *FL1000*: il DFE viene allenato con 1000 classi di un dataset ottenuto da gruppi Flickr.

Negli esperimenti in Sezione 4.1.4 il test set è ImageNet ILSVRC 2012. Quest'ultimo è semanticamente diverso dal dataset Flickr [7]. Infatti, i gruppi Flickr possono essere formati intorno a concetti specifici (marche di auto, animali, ecc.), concetti astratti (bellezza, immagini spaventose) oppure possono raccogliere immagini scattate con una marca specifica di fotocamera o con una specifica impostazione della fotocamera (bianco e nero, impostazione della luce) [29]. D'altro canto, ImageNet è basato su una struttura gerarchica di oggetti e animali [21].

4.1.3 Negatives selection

Gli shallow classifier vengono allenati utilizzando le features negative F^N nella memoria disponibile. È necessario definire una strategia di selezione:

- *ind*: F^N è composto da $|K|$ features di immagini del dataset *Yahoo Flick Creative Commons* dataset (YFCC, [86]), selezionate tali da rappresentare classi frequenti ma diversificate.
- *rand*: le features vengono bilanciatamente selezionate a caso dalle classi passate e presenti.
- *div*: vengono selezionate features diversificate fra le classi passate e presenti.

Nelle strategie *rand* e *div*, in S_t ogni classe ha $|K|/N_t$ esemplari in F^N. Come sopra accennato, durante l'allenamento del shallow classifier C_i, i rappresentanti della classe i-esima in F^N non vengono considerati.

4.1.4 Esperimenti

DeeSIL viene testato sul dataset ImageNet ILSVRC 2012. Il protocollo di valutazione ricalca quello di Rebuffi et al. [77] in *iCaRL*. Le classi vengono ordinate casualmente e l'algoritmo viene allenato in modo incrementale sul traning data. Dopo ogni batch di classi, il classifier viene valutato sulle classi già imparate. L'algoritmo viene confrontato con *iCaRL*, *LwF* e *FR*. La rete utilizzata è ResNet-18 allenata con 100 ($DeeSIL^{IN100}$) e 1000 classi ($DeeSIL^{IN1000}$) e 1000 gruppi Flickr ($DeeSIL^{FL1000}$).

I risultati vengono riassunti in Figura 4.2. *DeeSIL* ha la meglio in tutti gli stati incrementali e mantiene un'alta accuratezza del tempo, alleviando gli effetti del CF. Come era prevedibile, la versione di *DeeSIL* inizializzata con 1000 classi ImageNet mantiene una più alta accuratezza al progredire degli apprendimenti incrementali ispetto alla versione con 100 classi. Però, $DeeSIL^{FL1000}_{rand}$ mantiene una accuratezza simile a $DeeSIL^{IN100}_{div}$. Ciò è dovuto al fatto che fra classi ImageNet le features vengono riutilizzate efficacemente e sono più rappresentative. Inoltre, la scelta fra *div*, *rand* e *ind* come strategia di selezione è quasi ininfluente. *rand* e *div* portano a un incremento di 0.5 punti rispetto a *ind* dopo 1000 classi. *rand* è più semplice da calcolare rispetto a *div*, il che lo rende una sceltà migliore.

Dai risultati degli esperimenti, si evince che l'algoritmo di Belouadah e Popescu [7] risulta il migliore fra quelli trattati. È veloce, non necessita l'allenamento dell'intera rete e usa poca memoria. Inoltre, è in grado di funzionare anche senza di essa, come evidenziato da [8] nel loro esperimento.

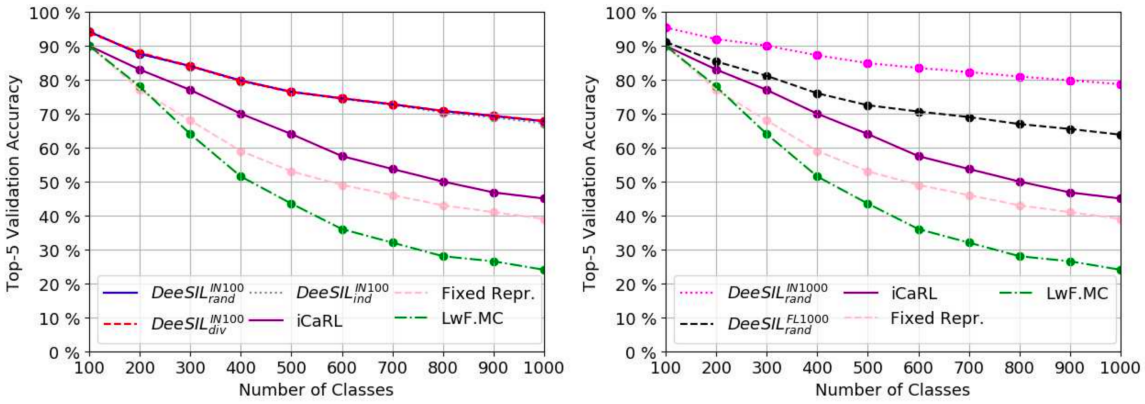

Figura 4.2: Top-5 accuracy per *iCaRL*, *LwF*, *FR* e *DeeSIL*. Quest'ultimo è presente nelle varianti sopra discusse. Tratto da [7].

4.2 *DSLDA*: Deep Streaming Linear Discriminant Analysis

La stragrande maggioranza degli algoritmi di CIL allena M_t considerando gruppi delle istanze in $D_t \cup K$ (*batch learning*). In *DSLDA*, il modello impara da istanza-per-istanza in tempo reale. Tale approccio viene detto *streaming learning*. L'algoritmo non conserva immagini di classi passate e fissa il feature extractor in S_0, rendendo l'approccio vincente in piattaforme embedded.

Nel batch learning il modello viene addestrato utilizzando tutti gli esemplari a disposizione in quello stato incrementale. Per gli algoritmi che fanno uso di KD ciò è evidente dalla Equazione 2.11, dal momento che vengono considerate tutte le coppie immagine-label in $D_t \cup K$. Al contrario, nello streaming learning il modello apprende da un esemplare alla volta.

4.2.1 Streaming Learning Discriminant Analysis

L'algoritmo utilizza lo *Streaming Linear Discriminant Analysis* (SLDA) proposto da Pang et al. [68] per allenare l'output layer della CNN in modo incrementale. Il decodificatore viene modellato dagli

autori come segue:

$$F\left(G\left(\mathbf{x}\right)\right) = \mathbf{W} \times \mathbf{z} + \mathbf{b}, \tag{4.1}$$

dove $\mathbf{x}$ è l'immagine in input, G è il feature extractor, F è il fully connected layer, $\mathbf{z} = G\left(\mathbf{x}\right)$, $\mathbf{W} \in \mathbb{R}^{N_t \times d}$ matrice dei pesi. L'algoritmo fissa i pesi di G e addestra F via streaming learning.

SLDA conserva un vettore medio per classe $\boldsymbol{\mu}_k \in \mathbb{R}^d$, un contatore di numero di esemplari per la classe $c_k \in \mathbb{R}$ $k \in [1, \ldots, N_t]$ e una matrice di covarianza $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$. Quando si presenta una nuova istanza $(\mathbf{z}_t, y)$ tramite streaming learning, il vettore medio e il contatore vengono aggiornati nel seguente modo:

$$\boldsymbol{\mu}_{(k=y,t+1)} = \frac{c_{(k=y,t)}\boldsymbol{\mu}_{(k=y,t)} + \mathbf{z}_t}{c_{(k=y,t)} + 1}, \tag{4.2}$$

$$c_{(k=y,t+1)} = c_{(k=y,t)} + 1, \tag{4.3}$$

dove $\boldsymbol{\mu}_{(k=y,t)}$ è il vettore medio per la classe y al tempo t e $c_{(k=y,t)}$ è il contatore associato. La matrice di precisione $\boldsymbol{\Lambda}$ è definita come

$$\boldsymbol{\Lambda} = \left[(1-\epsilon)\boldsymbol{\Sigma} + \epsilon \boldsymbol{I}\right]^{-1}, \tag{4.4}$$

con $\epsilon = 10^{-4}$.

Gli autori propongono due modi in cui trattare $\boldsymbol{\Sigma}$:

- $\boldsymbol{\Sigma}$ viene fissata in S_0.
- $\boldsymbol{\Sigma}$ viene aggiornata tramite streaming learning.

Nel secondo caso è possibile aggiornare la matrice di covarianza nel seguente modo:

$$\boldsymbol{\Sigma}_{t+1} = \frac{t\boldsymbol{\Sigma} + \Delta_t}{t+1}, \tag{4.5}$$

dove Δ_t è calcolato come

$$\Delta_t = \frac{t\left(\mathbf{z}_t - \boldsymbol{\mu}_{(k=y,t)}\right)\left(\mathbf{z}_t - \boldsymbol{\mu}_{(k=y,t)}\right)^T}{t+1}. \tag{4.6}$$

La predizione viene effettuata utilizzando (4.1). $\mathbf{w}_k$ è la a riga k-esima di $\mathbf{W}$ e viene calcolata come:

$$\mathbf{w}_k = \boldsymbol{\Lambda}\boldsymbol{\mu}_k. \tag{4.7}$$

La componente k-esima del bias $\mathbf{b}$ viene calcolata come

$$b_k = -\frac{1}{2}\left(\boldsymbol{\mu}_k \cdot \boldsymbol{\Lambda}\boldsymbol{\mu}_k\right). \tag{4.8}$$

4.2.2 Esperimenti

Pang et al. [68] confrontano le due varianti di *DSLDA* (matrice di covarianza statica e non) con l'algoritmo FT *iCaRL* [77] e un modello fully-trained. Di quest'ultimo vengono trattate due varianti. Nella prima l'intero modello viene allenato con tutti i dati disponibili. Invece, nella seconda solo l'ultimo layer viene allenato con tutti i dati disponibili. Tutti gli algoritmi utilizzano ResNet-18 come rete. I modelli vengono confrontati su due dataset:

- ImageNet-1K ILSVRC-2012. Le reti vengono inizializzate con 100 classi casuali e le performance vengono valutate ogni 100 classi imparate attraverso la top-5 accuracy.
- CORe50. Dato che il dataset consiste di sequenze video ordinate temporalmente, l'ordine in cui i dati vengono presentati influenza il risultato finale. Per tale motivo vengono utilizzati 4 differenti ordinamenti delle immagini per *DSLDA*:
  - IID: Tutti i frame sono mescolati casualmente.
  - Class IID: Tutti i frame sono mescolati casualmente all'interno di ogni classe.
  - Instance: I video e i relativi frame vengono ordinati secondo l'oggetto del video.

– Class Instance: I video e i relativi frame vengono ordinati secondo l'oggetto del video all'interno di ogni classe.

A prescindere dall'ordinamento, il modello viene valutato ogni 1200 campioni imparati utilizzando la top-1 accuracy.

La performance complessiva viene poi calcolata come:

$$\Omega = \frac{1}{T} \sum_{t=1}^{T} \frac{\alpha_t}{\alpha_{\text{offline},t}}, \tag{4.9}$$

dove α_t è la performance (top-1 o top-5 accuracy) dell'algoritmo ottenuta al tempo t e $\alpha_{\text{offline},t}$ è la performance del modello fully-trained ottenuta al tempo t.

Su ImageNet i modelli vengono inizializzati con 100 classi. Dunque le rimanenti 900 classi vengono imparate in modo incrementale. Su CORe50, F e G vengono inizializzati con i pesi ottenuti dall'addestramento su ImageNet. L'ultimo fully connected layer viene sostituito con un layer di 10 output. F e G vengono riallenati su 1200 esemplari, appartenenti a 2 classi, di CORe50 ottenuti secondo uno degli ordinamenti precedentemente discussi.

I risultati complessivi sono riportati nella Tabella 4.1. Da questa è chiaro che *DSLDA* ottiene risultati migliori di *iCaRL* in tutti i dataset e ordinamenti. La plasticità della matrice di covarianza ha un effetto leggermente positivo sulle performance su ImageNet, ma risulta essere un fattore molto importante su CORe50, contribuendo notevolmente al miglioramento delle performance. Infatti, ciò è dovuto al fatto che su ImageNet il modello è inizializzato con 100 classi, mentre su CORe50 F e G vengono riaddestrati su soli 1200 campioni. La matrice di covarianza risulta molto più significativa su ImageNet, dunque non è necessario aggiornarla. Al contrario, su CORe50 aggiornare la matrice è fondamentale per rappresentare più features possibili.

Tabella 4.1: Valori di Ω su ImageNet e CORe50. Il migliore modello per ogni dataset e ordinamento è evidenziato in **grassetto**.

Dataset	ImageNet	CORe50			
	CLS IID	IID	CLS IID	INST	CLS INST
SLDA (Fixed Σ)	0.748	0.967	0.916	0.943	0.913
SLDA (Plastic Σ)	**0.752**	**0.976**	**0.958**	**0.963**	**0.959**
iCaRL	0.692	-	0.839	-	0.845
Upper bounds approssimativi					
Fully-trained (Ultimo layer)	0.853	0.979	0.954	0.966	0.955
Fully-trained	1.000	1.000	1.000	1.000	1.000

Dai risultati dell'esperimento è evidente l'ottima accuratezza ottenuta da *DSLDA*. Dal momento che viene allenato solo l'output layer, l'algoritmo risulta molto più veloce di altri algoritmi. Inoltre, l'approccio deve la sua velocità anche allo streaming learning, dal momento che i metodi basati sul batch learning iterano su dataset più volte. Uno possibile sviluppo futuro per *DSLDA* è applicare della KD in contesti generosi di memoria e calcolo computazionale. In questo modo sarebbe possibile sfruttare momenti in cui il modello è in idle per allenare il feature extractor.

Capitolo 5

Fine-Tuning

Gli algoritmi FT sono caratterizzati dall'uso della KD per mitigare l'effetto del catastrophic forgetting [10]. Grazie a tale approccio, gli algoritmi *LwF* e la sua variante *iCaRL* rappresentano un punto di riferimento per tale paradigma [1, 8, 10, 38], sebbene metodi più recenti raggiungano facilmente accuratezze migliori [1, 7, 33, 78, 92, 95]. Tali algoritmi raggiungono un'accuratezza elevata quando viene fornita una quantità sufficiente di memoria per conservare gli esemplari di step incrementali passati. La tesi affronta i seguenti algoritmi FT:

- *BiC* [92] di Wu et al. (2019)
- *SS-IL* [1] di Ahn et al. (2021)

5.1 *BiC*: Bias Correction

Wu et al. [92] sottolineano che gli algoritmi FT con una bassa quantità di memoria soffrono di una scarsa scalabilità. Essi ritengono che ciò sia dovuto al *data imbalance*, fenomeno nel quale il numero di esemplari disponibili delle classi passate è inferiore rispetto a quello delle classi dell'attuale stato incrementale. Ciò rappresenta un problema poiché nella fase di addestramento si da minore importanza alle classi passate. Il data imbalance porta l'ultimo layer ad avere un forte bias verso le nuove classi, il che significa che il modello tende ad assegnare più facilmente una classe più recente a un'istanza. Come discusso nel Capitolo 1, questo fenomeno è del tutto analogo al CF. L'algoritmo proposto, denominato *Bias Correction* (*BiC*), cerca di correggere il bias presente nell'ultimo layer per contrastare l'imbilanciamento dei dati e il catastrophic forgetting.

5.1.1 Correzione del bias e addestramento

Bias correction layer

Il bias correction layer è un modello lineare posto dopo il FC layer. Esso prende in input i punteggi in uscita dal FC layer e calcola i nuovi punteggi del modello tramite l'equazione

$$q_t^k = \begin{cases} o_t^k, & \text{se } 1 \leq k \leq N_{t-1}, \\ \alpha o_t^k + \beta, & \text{se } N_{t-1}+1 \leq k \leq N_t. \end{cases}, \tag{5.1}$$

dove o_t^k è il punteggio predetto dal FC layer per la classe k-esima, definito nella Sezione 2.1, α e β sono i due parametri di correzione di bias. Infatti, solamente i punteggi delle nuove classi vengono modificati dal bias correction layer. Addestrando il modello sul validation set, i parametri di bias correction sono in grado di ridimensionare il punteggio finale associato a una nuova classe, moderando il bias verso di essa e gli effetti del CF. I due parametri sono condivisi fra tutte le nuove classi, in modo tale da poterli stimare attraverso un piccolo validation set.

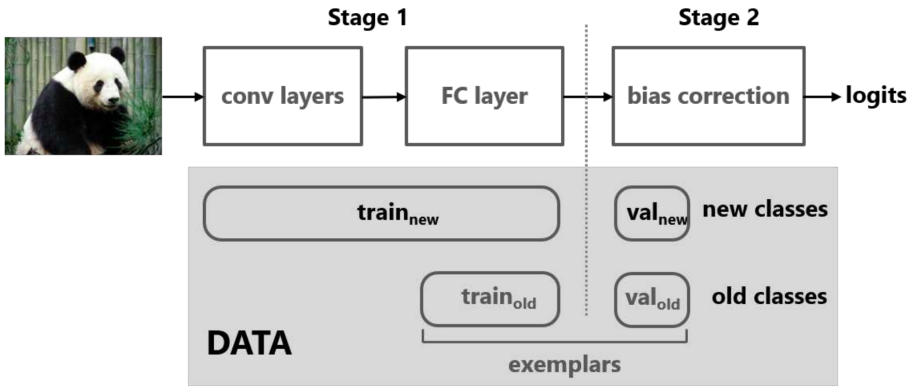

Figura 5.1: Panoramica di *BiC*. Gli esemplari delle classi passate e recenti vengono suddivisi in training set e validation set. Il training set viene utilizzato per addestrare il feature extractor (i convolutional layer e il fully connected layer) (nella fase 1). Il validation set viene utilizzato per la correzione del bias (nella fase 2). Tratto da [92].

Knowledge distillation

L'algoritmo fa uso della GKD [1]. Di conseguenza, il modello viene addestrato utilizzando la loss function definita nell'Equazione 2.17, dove il termine L_d relativo alla distillation loss è formulata come nell'Equazione 2.13. In particolare, si pone

$$\lambda_t = \frac{N_{t-1}}{N_{t-1} + P_t}, \tag{5.2}$$

dove λ_t è il valore dello scalare λ in S_t. Inoltre, si pone $\lambda_0 = 0$, dal momento che in S_0 non ci sono ancora classi passate, e dunque (2.13) non ha effetto nell'Equazione 2.17. È interessante notare che per $N_{t-1} \gg P_t$ si ha $\lambda_t \approx 1$ e $L \approx L^d$. Ciò significa che per stati incrementali elevati, mantenere conoscenza delle classi passate diventa estremamente importante poiché la loro quantità diventa sempre più grande.

Addestramento

BiC comprende due fasi di addestramento. Inizialmente, viene addestrato il feature extractor e il classifier usando la distillation loss nell'Equazione 2.17. Nella seconda fase, i parametri del feature extractor e del classifier vengono fissati e si stimano due parametri di bias utilizzando un set di validazione.

Gli esemplari di D_t e K vengono divisi in un training set e in un validation set. Il training set viene utilizzato per apprendere i parametri del feature extractor e del classifier. Invece, il validation set è impiegato per apprendere i parametri del bias correction layer. Il training set e il validation set ottenuti da K vengono chiamati rispettivamente $train_{old}$ e val_{old}. Al contrario, il training set e il validation set ottenuti da D_t vengono chiamati rispettivamente $train_{new}$ e val_{new}. In generale, si ha $|train_{new}| > |train_{old}|$ a causa della limitatezza della memoria. Invece, per i validation set vale $|val_{new}| = |val_{old}|$, il che ha l'obiettivo di bilanciare il numero di esemplari di classi passate con quello di classi recenti per il bias correction layer. Inoltre, i validation set sono, in generale, più piccoli dei training set [92]. Di conseguenza, gli autori di [92] ritengono che il bias correction layer debba avere un numero ristretto di parametri, dal momento i sottoinsiemi usati nella loro stima hanno una cardinalità ristretta. Uno schema della procedura di divisione appena descritta è riportato in Figura 5.1. L'ottimizzazione dei parametri di correzione di bias avviene congelando il feature extractor e il classifier. La loss function L_b utilizzata è la cross entropy loss dell'Equazione 2.11.

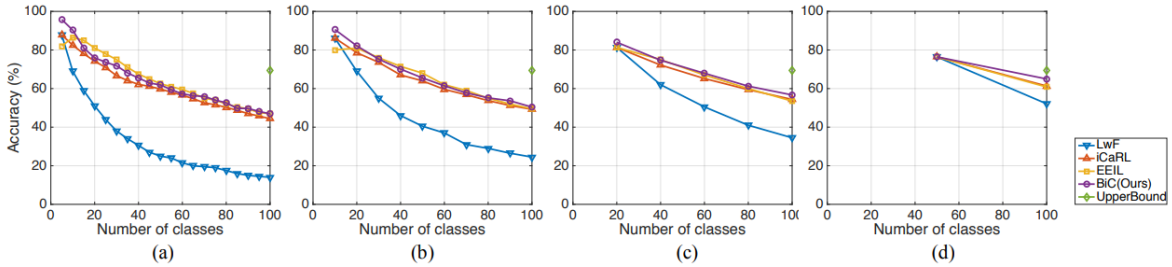

Figura 5.2: Top-1 accuracy su CIFAR-100 con $P_t = 5$ (a), $P_t = 10$ (b), $P_t = 20$ (c), $P_t = 50$ (d). In verde si raffigura l'accuratezza del modello fully-trained. Tratto da [92].

5.1.2 Esperimenti

Accuratezza

Wu et al. [92] testano il modello su un dataset grande (ImageNet-1K ILSVRC-2012) e su uno piccolo (CIFAR-100). L'implementazione del modello utilizza 18-layer ResNet [36] per ImageNet e una 32-layer ResNet per CIFAR-100. L'implementazione di ResNet proviene dai modelli di TensorFlow.

Su ImageNet-1K, ogni ogni step incrementale ha 100 epoch [64]. Ogni stato incrementale introduce 100 classi. Il learning rate [4, 64] è posto a 0.1 e viene diviso per 10 dopo 30, 60, 80 e 90 epoch. Il weight decay [4] è posto a 0.0001 e la grandezza di un batch è 256. Le immagini vengono pre-processate seguendo i passaggi di pre-processing di VGG [83], tra cui ritaglio casuale, ribaltamento orizzontale e ridimensionamento. La temperatura T della softmax con temperatura in Equazione 2.14 è posta a 2.

Su CIFAR-1000, ogni ogni step incrementale ha 250 epoch [64]. Vengono condotti gli esperimenti introducendo 5, 10, 20 e 50 classi a ogni stato incrementale. Il learning rate è posto a 0.1 e viene a 0.01, 0.001, 0.0001 dopo 100, 150 e 200 epoch. Il weight decay è posto a 0.0002 e la grandezza di un batch è 128. Le immagini vengono preprocessate seguendo i passaggi indicati in [36], tra cui ritaglio casuale e ribaltamento orizzontale. Come per ImageNet, la temperatura T della softmax con temperatura in Equazione 2.14 è posta a 2.

Il modello viene comparato con altri metodi FT: *LwF* [55], *iCaRL* [76] e *EEIL* [16]. Per questi vengono utilizzati gli stessi protocolli e parametri descritti precedentemente. *LwF* non usa esemplari dalle classi passate. *iCaRL*, *EEIL* e *BiC* usano lo stesso numero di esemplari dalle classi passate.

In Tabella 5.1 vengono riportati i risultati su ImageNet-1K. *BiC* ottiene le migliori performance da S_2 in poi. Ciò è dovuto al fatto che la data imbalance cresce assieme al numero di step incrementali [92], e *BiC* è stato progettato specificatamente per risolvere tale problema. In S_0 e S_1, invece, *EEIL* ottiene le migliori performance. Secondo gli autori, ciò è dovuto alla *enhanced data augmentation* (EDA) [16] utilizzata dall'algoritmo, che si rivela molto efficacie quanto la data imbalance non è troppo elevata. Per quanto riguarda *LwF* e *iCaRL*, invece, le performance degradano molto velocemente nel tempo.

In Figura 5.2 vengono riportati i risultati su CIFAR-100. I metodi *BiC*, *iCaRL* e *EEIL* ottengono performance simili. *BiC* risulta migliore negli stati incrementali con 50 e 20 classi, mentre è leggermente inferiore a *EEIL* in quelli con 10 e 5 classi.

Tabella 5.1: Top-1 accuracy su ImageNet-1K con $P_t = 100$. Il migliore risultato è evidenziato in **grassetto**. Tratto da [92].

	100	200	300	400	500	600	700	800	900	1000
LwF	0.900	0.770	0.680	0.595	0.525	0.495	0.465	0.430	0.405	0.390
iCaRL	0.900	0.830	0.775	0.705	0.630	0.575	0.535	0.500	0.480	0.440
EEIL	**0.950**	**0.955**	0.860	0.775	0.710	0.680	0.620	0.598	0.550	0.520
BiC	0.941	0.925	**0.896**	**0.891**	**0.857**	**0.832**	**0.802**	**0.775**	**0.750**	**0.732**

Bias correction layer

Gli autori hanno condotto un ulteriore esperimento per verificare l'efficacia del bias correction layer. A tale scopo, vengono costruite delle confusion matrix per 4 modelli:

- baseline-1, modello addestrato con la cross entropy loss dell'Equazione 2.11;
- baseline-2, modello addestrato con la distillation loss dell'Equazione 2.17;
- *BiC*;
- Fully-trained.

In Figura 5.3 vengono riportate le confusion matrix di tale esperimento. Valori bassi e omogenei in tutta la matrice al di fuori della diagonale indicano l'assenza di bias nel modello (c, d). Al contrario, valori elevati e concentrati nelle ultime colonne sono sintomo di bias verso le nuove classi (a, b). Infatti, dalla figura è evidente l'efficacia del bias correction layer di *BiC*, la cui matrice di confusione assomiglia a quella del modello fully-trained.

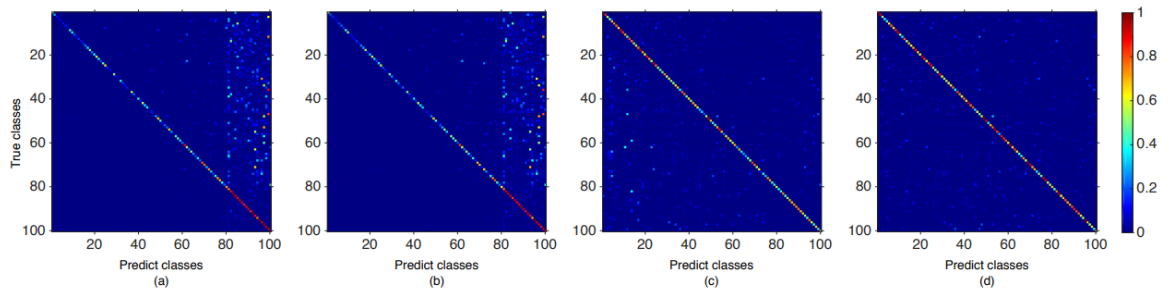

Figura 5.3: Confusion matrix per baseline-1 (a), baseline-2 (b), *BiC* (c) e il modello fully-trained (d). Tratto da [92].

Validation set

Vengono testate diverse suddivisioni di $train_{old}$ e val_{old} al fine di trovare un rapporto ottimale $train_{old}$: val_{old} che favorisca sia l'apprendimento del feature extractor che quello del bias correction layer [92], considerando il limitato numero di esemplari delle classi passate disponibili. La Tabella 5.2 riporta i risultati di tale esperimento, dimostrando la superiorità del rapporto 9 : 1.

Tabella 5.2: Top-1 accuracy per diverse suddivisioni di campioni di training e validation. L'esperimento è stato condotto su CIFAR-100 con $P_t = 20$. Il migliore risultato è evidenziato in **grassetto**. Tratto da [92].

trainold : *valold*	20	40	60	80	100
9 : 1	0.8400	**0.7469**	**0.6793**	**0.6125**	**0.5669**
8 : 2	0.8450	0.7319	0.6501	0.5868	0.5431
7 : 3	**0.8470**	0.7160	0.6368	0.5812	0.5374
6 : 4	0.8333	0.6884	0.6221	0.5600	0.5117

Efficacia e utilizzo di memoria

In [92] viene proposto *BiC*, il quale ha lo scopo di contrastare l'effetto del data imbalance sul modello. Dopo aver verificato che il classifier ha un forte bias verso le nuove classi, gli autori introducono nel modello un bias correction layer. Esso si rivela efficacie nel correggere tale bias e nel mantenere elevate le performance, soprattutto per un numero elevato di classi [92]. Però, come viene sottolineato in [10], il modello necessita una certa quantità di memoria per funzionare correttamente, atta a conservare abbastanza esemplari di training e validazione. Infine, si nota che *BiC* non può funzionare in assenza di memoria, al contrario di *LwF* e tutti gli algoritmi FR.

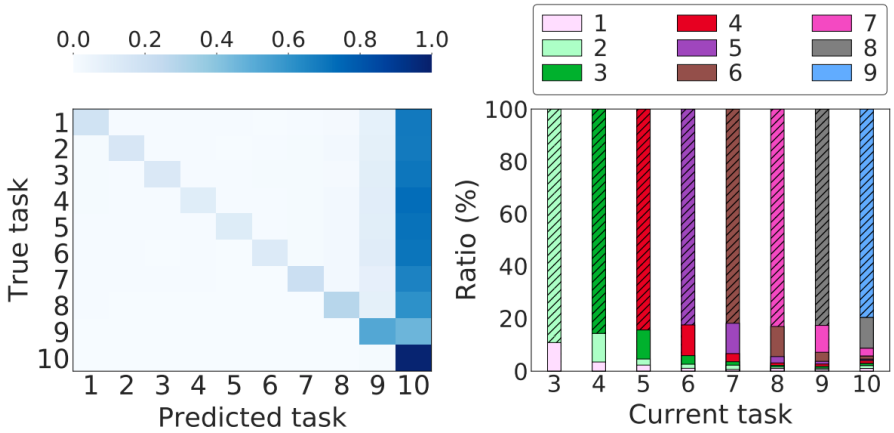

Figura 5.4: Risultati di un esperimento di un metodo che usa cross-entropy loss e GKD, effettuato su ImageNet-1K, $P_t = 100$ e $|K| = 10k$. A sinistra, la confusion matrix fra gli stati incrementali. A destra, rapporto del numero di predizioni attribuite a ogni task $1, \dots, t-1$, compiute in S_{t-1} su D_t. Tratto da [1].

5.2 *SS-IL*: Separated Softmax for Incremental Learning

Ahn et al. [1], analogamente a Wu et al. [92], ritengono necessario eliminare il bias verso le nuove classi dovuto al data imbalance. Inoltre, essi sostengono che la GKD, definita nell'Equazione 2.18, non sia in grado di risolvere tale bias. Di conseguenza, viene proposto un metodo che, a differenza di *BiC*, fa uso della TKD, definita nell'Equazione 2.19). Gli autori trattano due osservazioni: il bias è causato dalla cross-entropy loss e che il bias viene preservato dalla GKD.

5.2.1 Bias causato dalla cross-entropy loss

Data la presenza del data imbalance, si mostra la cross-entropy loss $L_{\text{CE},t}$ causa il bias verso le nuove classi. Per far ciò si calcola il gradiente della cross-entropy rispetto all'output o_t^c. Esso vale

$$\frac{\partial L_{\text{CE},t}((\mathbf{x}, y), \boldsymbol{\theta})}{\partial o_t^c} = \phi_t^{1:t,c}(\mathbf{x}, \boldsymbol{\theta}) - \mathbb{1}_{c=y}, \tag{5.3}$$

dove $\mathbf{x}$ è l'esemplare in input, y è la sua label e $\boldsymbol{\theta}$ sono i parametri del modello. Poiché la derivata parziale nell'Equazione 5.3 è sempre positiva per $c \neq y$, gli score delle classi passate decresceranno durante gli step di gradient descent eseguiti per gli esemplari in D_t, che sono più numerosi di quelli in K, sopratutto per t elevati.

5.2.2 Bias preservato dalla GKD

Gli autori ritengono che la GKD non sia in grado di risolvere il bias causato dalla cross-entropy loss. Nell'Equazione 2.18, $\boldsymbol{\phi}_{t-1}^{1:t-1}(\mathbf{x})$ è l'output del modello in S_{t-1}, usato per la KD. La Figura 5.4 a sinistra mostra la matrice di confusione di un modello addestrato con cross-entropy loss e GKD. Gli alti valori nell'ultima colonna indicano che la maggior parte delle istanze $\mathbf{x} \in D_t$ in input al modello vengono assegnate a delle classi appartenenti all'ultimo step incrementale appreso. La medesima Figura a destra mostra il rapporto del numero di ogni task $\{1, \dots, t-1\}$ predetto dal modello in S_{t-1} quando le istanze del nuovo task $\mathbf{x} \in D_t$ vengono fornite in input. Emerge che le predizioni hanno un forte bias verso le classi del task $t-1$. A partire dall'esperimento riportato, gli autori ipotizzano che $\boldsymbol{\phi}_{t-1}^{1:t-1}(\mathbf{x})$ sia *distorto* verso le classi più recenti in S_{t-1}, e che ciò preservi il bias quando utilizzato nella GKD, penalizzando di molto le probabilità in output delle classi meno recenti, e quindi causando il CF.

5.2.3 Prevenzione del bias

Le due osservazioni suggeriscono che il bias sia dovuto al fatto che le probabilità softmax vengono calcolate combinando i nuovi e i passati task. Infatti, come specificato nell'Equazione 2.11, l'addestra-

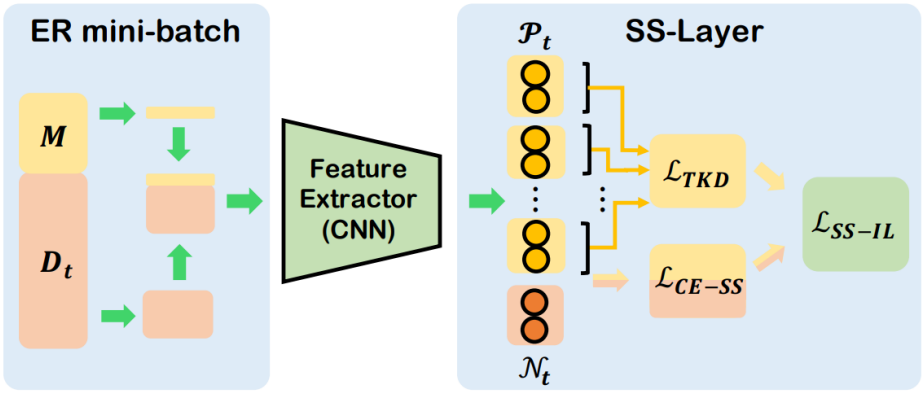

Figura 5.5: Illustrazione di *SS-IL*. Le regioni gialle rappresentano le classi passate, mentre le regioni arancioni rappresentano le nuove classi. Tratto da [1].

mento in S_t avviene prelevando campioni da $D_t \cup K$, senza distinguere l'appartenenza di un elemento alla memoria oppure al dataset. Inoltre, la distillation loss nella GKD in Equazione 2.18 non prende in considerazione il task di una istanza pescata dalla memoria. Tale informazione è importante, dal momento che in un modello biased l'accuratezza verso le classi passate è più bassa rispetto a quella verso le classi recenti. *SS-IL* è formato da due componenti: il layer di output *Separated-Softmax* (SS) e la TKD. Sia $\mathcal{P}_t$ insieme delle classi del task precedente, e $\mathcal{N}_t$ l'insieme delle classi del task attuale.

SS layer

Per ogni $(\mathbf{x}, y) \in D_t \cup K$ si definisce l'output layer SS modificando la cross-entropy loss nell'equazione

$$L_{\text{CE-SS},t}((\mathbf{x}, y), \boldsymbol{\theta}) = L_{\text{CE},t-1}((\mathbf{x}, y), \boldsymbol{\theta}) \cdot \mathbb{1}\{y \in \mathcal{P}_t\} + D_{KL}(\mathbf{y}_t || \mathbf{p}_t(\mathbf{x}, \boldsymbol{\theta})) \cdot \mathbb{1}\{y \in \mathcal{N}_t\}, \tag{5.4}$$

dove $y_t^{(k)} = 1$ se $\mathbf{x}$ appartiene alla classe k-esima, altrimenti $y_t^{(k)}$, $\mathbf{y}_t \in \mathbb{R}^{|\mathcal{N}_t|}$. In questo modo, se $(\mathbf{x}, y) \in \mathcal{P}_t$ l'output softmax viene calcolato solamente usando i punteggi delle classi in $\mathcal{P}_t$. Analogamente, se $(\mathbf{x}, y) \in \mathcal{N}_t$ l'output softmax viene calcolato solamente usando i punteggi delle classi in $\mathcal{N}_t$. Si noti che se $c \in \mathcal{P}_t$ e $(\mathbf{x}, y) \in D_t$ si ha che

$$\frac{\partial L_{\text{CE},t}((\mathbf{x}, y), \boldsymbol{\theta})}{\partial o_t^c} = 0. \tag{5.5}$$

Di conseguenza, il gradiente ottenuto dagli esemplari con classe in $\mathcal{N}_t$ (se $(\mathbf{x}, y) \in D_t$ allora $y \in \mathcal{N}_t$) non penalizza le classi in $\mathcal{P}_t$, poiché i parametri impiegati nella loro classificazione non vengono alterati.

TKD

L'algoritmo fa uso della TKD invece che della GKD. Data la presenza dell'output layer SS e poiché le probabilità $\{\mathbf{p}_s(\mathbf{x}, \boldsymbol{\theta})\}_{s=1}^{t-1}$ vengono calcolate all'interno dello stesso task, ci si aspetta che TKD sia resistente al bias verso le nuove classi, a differenza della GKD.

Loss function

La loss function di *SS-IL* è ottenuta come:

$$L_{\text{SS-IL},t}((\mathbf{x}, y), \boldsymbol{\theta}) = L_{\text{CE-SS},t}((\mathbf{x}, y), \boldsymbol{\theta}) + L_{\text{TKD},t}((\mathbf{x}, y), \boldsymbol{\theta}). \tag{5.6}$$

Il mini-batch SGD [51] viene eseguito con l'*Experience-Replay* (ER) [17], una tecnica che mantiene fissato il rapporto tra il numero di classi nuove e passate all'interno di un mini-batch per assicurarsi che vi sia una quantità minima di campioni dalla memoria K. Una illustrazione di *SS-IL* è riportata in Figura 5.5

5.2.4 Esperimenti

Dataset

SS-IL viene messo a confronto con gli altri metodi allo stato dell'arte su due dataset di grandi dimensioni: ImageNet ILSVRC 2012 e Landmark-v2. In ImageNet si utilizzano 1000 classi, ognuna delle quali con 1300 istanze. In Landmark-v2 si selezionano 1000 e 10 000 classi in base al numero più elevato di campioni per classe per ottenere due varianti, e si indica ciascun dataset come Landmark-v2-1K e Landmark-v2-10K, rispettivamente. Il numero di immagini per classe in Landmark varia da 300 a 500. Questa variazione, inevitabilmente, genera uno squilibrio nel quantitativo di dati destinati all'addestramento per ciascuna classe. Viene seguito il protocollo di valutazione definito in [77].

Protocollo di valutazione

Nei dataset da 1000 classi (ImageNet e Landmark-v2-1K) il numero di step incrementali varia come $T = 5, 10, 20$, il che corrisponde a $m = \{200, 100, 50\}$ nuove classi. Per tali dataset si ha $|K| = \{5k, 10k, 20k\}$. Invece, in Landmark-v2-10K, per i suddetti valori di T si ha $m = \{2k, 1k, 500\}$. In questo caso valo $|K| = \{20k, 40k, 60k\}$. Si utilizza la selezione casuale usata in [10] per la costruzione della memoria.

Risultati

I risultati degli esperimenti vengono riportati nelle Tabelle 5.3 e 5.4. *SS-IL* viene confrontato con *iCaRL* [77], *FT* [proposto in 8], *IL2M* [8], *BiC*, *LUCIR* [38], *PODNet* [24]. Per ogni metodo si utilizzano gli iperparametri definiti nei paper originali. Non è stato possibile valutare *PODNet* in Landmark-v2 a causa dei limiti di tempo e memoria. È evidente che *SS-IL* domina in tutti gli stati incrementali e tutte le configurazioni di memoria, a eccezione per $T = 20$ in Landmark-v2-10K, dove LUCIR ne esce vincitore.

Tabella 5.3: Top-1 accuracy e top-5 accuracy medie fra gli step incrementali con $T = 10$. Il migliore risultato è evidenziato in **grassetto**. Tratto da [1].

	$T = 10$		
Dataset	ImageNet-1K	Landmark-v2-1K	Landmark-v2-10K
$\|K\|$	$5k/10k/20k$	$5k/10k/20k$	$20k/40k/60k$
Average Top-1 accuracy			
iCaRL	0.470/0.505/0.531	0.374/0.411/0.440	0.231/0.272/0.292
FT	0.381/0.458/0.535	0.419/0.489/0.558	0.336/0.403/0.445
IL2M	0.419/0.484/0.553	0.424/0.492/0.559	0.342/0.407/0.448
EEIL	0.578/0.594/0.609	0.521/0.555/0.582	0.435/0.461/0.480
BiC	0.513/0.564/0.605	0.499/0.545/0.584	0.387/0.437/0.465
LUCIR	0.510/0.536/0.565	0.505/0.537/0.573	0.462/0.491/0.509
PODNet	0.522/0.575/0.604	-	-
SS-IL	**0.635/0.645/0.652**	**0.577/0.590/0.599**	**0.501/0.514/0.519**
Average Top-5 accuracy			
iCaRL	0.710/0.751/0.774	0.569/0.620/0.652	0.356/0.419/0.448
FT	0.667/0.733/0.788	0.621/0.685/0.740	0.494/0.567/0.606
IL2M	0.706/0.753/0.797	0.624/0.685/0.739	0.497/0.567/0.606
EEIL	0.812/0.820/0.830	0.726/0.749/0.767	0.604/0.626/0.641
BiC	0.744/0.789/0.818	0.692/0.731/0.761	0.555/0.607/0.633
LUCIR	0.724/0.756/0.787	0.687/0.722/0.753	0.622/0.652/0.667
PODNet	0.736/0.794/0.821	-	-
SS-IL	**0.860/0.864/0.867**	**0.781/0.788/0.793**	**0.678/0.686/0.691**

Le alte performance ottenute negli esperimenti su dataset a larga scala confermano il successo di *SS-IL* nel prevenire il bias verso le nuove classi all'aumentare degli step incrementali. Inoltre, date le osservazioni fatte in precedenza, si conclude che la scelta di utilizzare la TKD piuttosto che la GKD assieme al layer di output SS si rivela ottimale nel bilanciare gli score delle classi nuove e passate.

Tabella 5.4: Top-1 accuracy e top-5 accuracy medie fra gli step incrementali con $|M| = 10k$ (ImageNet e Landmark-v2-1K), $40k$ (Landmark-v2-10K). Il migliore risultato è evidenziato in **grassetto**. Tratto da [1].

	$\|K\| = 10k$ (1K), $40k$ (10K)		
Dataset	ImageNet-1K	Landmark-v2-1K	Landmark-v2-10K
Step	$T = 20/T = 5$	$T = 20/T = 5$	$T = 20/T = 5$
Average Top-1 accuracy			
iCaRL	0.448/0.562	0.381/0.451	0.238/0.312
FT	0.445/0.469	0.450/0.536	0.383/0.431
IL2M	0.456/0.524	0.452/0.542	0.384/0.440
EEIL	0.535/0.638	0.505/0.591	0.415/0.498
BiC	0.485/0.615	0.458/0.611	0.363/0.508
LUCIR	0.468/0.613	0.485/0.610	**0.442**/0.539
PODNet	0.488/0.655	-	-
SS-IL	**0.588**/**0.682**	**0.514**/**0.643**	0.430/**0.558**
Average Top-5 accuracy			
iCaRL	0.697/0.797	0.586/0.657	0.378/0.468
FT	0.713/0.730	0.646/0.725	0.546/0.589
IL2M	0.718/0.787	0.644/0.731	0.543/0.598
EEIL	0.770/0.853	0.703/0.777	0.578/0.660
BiC	0.694/0.842	0.638/0.793	0.524/0.676
LUCIR	0.692/0.827	0.677/0.780	0.607/0.693
PODNet	0.711/0.858	-	-
SS-IL	**0.829**/**0.884**	**0.733**/**0.818**	**0.618**/**0.724**

Capitolo 6

Approcci, proprietà e applicazioni

Nel Capitolo 2 sono stati introdotti i tre tipi principali di algoritmi di CIL: Model-Growth, Fixed-Representation e Fine-Tuning. Ognuno di essi segue un approccio differente nel modo in cui è in grado di ampliare il numero di classi apprese, pur mantenendo la conoscenza passata. Gli algoritmi MG espandono la propria struttura, aumentando il numero di parametri del modello per ampliare il numero di classi riconosciute. Tale crescita permette al modello di non sovrascrivere i parametri iniziali, utili per il riconoscimento di classi introdotte nei primi step incrementali. Gli algoritmi FR conservano la propria conoscenza passata fissando il feature extractor dopo S_0. Per accomodare nuove capacità classificative, il modello aggiorna il classificatore a ogni step incrementale. Infine, gli algoritmi FT adattano i parametri dell'intero modello minimizzando una loss con un termine di distillazione. In questo modo si tiene conto, in modo approssimato, delle istanze di classi passate durante l'addestramento nei successivi step incrementali.

Dopo aver trattato alcuni algoritmi di CIL, la tesi offre una discussione in cui si confrontano i vari metodi impiegati e si individuano i contesti in cui un particolare approccio è preferibile piuttosto che un altro. Per ogni algoritmo presentato, si trattano i riscontri teorici negli esperimenti riportati nella tesi.

6.1 Tipi di algoritmi e proprietà

Nella Sezione 2.6 sono state introdotte 6 proprietà ricercate in un algoritmo di CIL. Tali proprietà sono semplicità, indipendenza dalla memoria, accuratezza, tempestività e plasticità. Idealmente, un algoritmo di CIL mira ad avere tutte queste proprietà. Inoltre, alcuni contesti favoriscono algoritmi che godono di alcune proprietà piuttosto che altre. Nella Tabella 6.1 si discutono le suddette proprietà in relazione al tipo di algoritmo di CIL.

6.2 Discussione dei risultati sperimentali

Di seguito si analizzano e discutono i risultati ottenuti negli esperimenti riportati nei Capitoli 3, 4 e 5. In particolare, ci si chiede se i risultati verificano o meno le considerazioni teoriche sui tipi di algoritmi e le loro caratteristiche. Nel farlo, si utilizza la Tabella 6.1 come riferimento per le proprietà tipicamente attribuite a un tipo di algoritmo di CIL.

6.2.1 *Tree-CNN*

L'algoritmo appartiene al tipo MG, dunque ci si aspetta che esso ottenga un'alta accuratezza. Dai risultati riportati nelle Figure 3.3 e 3.4 e Tabella 3.1 si nota che *Tree-CNN* ottiene i più alti valori di accuratezza fra gli algoritmi testati. In particolare, tutte e tre le versioni di *Tree-CNN* sono più performanti degli algoritmi FT *iCaRL* e *LwF*. Ciò è spiegabile dal momento che *Tree-CNN* non impone limitazioni alla memoria impiegata dal modello. D'altro canto, *LwF* non fa uso di memoria, mentre *iCaRL* ne utilizza una quantità prefissata. Di conseguenza, *Tree-CNN* può ospitare i nuovi parametri in una memoria illimitata e dunque ampliare la propria conoscenza pur mantenendo quella passata. Le

considerazioni teoriche sull'influenza della memoria nell'accuratezza degli algoritmi MG e FT riportate nella teoria sono verificate dai risultati dell'esperimento.

Degli importanti risultati si ricavano dall'analisi delle tre varianti dell'algoritmo. Nelle Figure 3.3 e 3.4 e Tabella 3.1 vi è una correlazione fra grandezza del modello e accuratezza ottenuta dagli esperimenti. Infatti, si nota che maggiore è l'espansione del modello durante gli step incrementali maggiore è l'accuratezza ottenuta negli esperimenti. Tale osservazione conferma la capacità di un algoritmo MG di ampliare la propria conoscenza attraverso l'aggiunta di componenti e parametri.

6.2.2 *DER*

DER è un algoritmo MG caratterizzato da un'ottima accuratezza negli step incrementali. Nelle Tabelle 3.2 e 3.3 si nota che le performance rimangono elevate nel tempo, in accordo con quanto riportato in Tabella 6.1. In particolare, ottiene risultati migliori di *BiC*, algoritmo FT allo stato dell'arte, in tutte le configurazioni testate. Però, in questo caso non vi sono limitazioni sulla memoria occupabile da *DER* per l'espansione del modello, mentre per gli algoritmi FT è fissata a priori. Ciò è in accordo con le considerazioni teoriche sulla dipendenza dalla memoria dei due approcci: se questa è limitata, gli algoritmi MG e FT ne risentono in termini di accuratezza. In Tabella 3.2. Si può vedere che la versione di *DER* senza riduzione del feature extractor occuperebbe fino a 23 volte la memoria occupata dagli altri algoritmi in S_{50}. Invece, grazie alla riduzione del feature extractor, *DER* riesce a ottenere le migliori performance pur avendo meno parametri degli algoritmi FT confrontati. Dunque, *DER* è capace di espandere la propria dimensione minimizzando il numero di parametri addizionali, come auspicato per algoritmi MG in [10]. In aggiunta, la versione *DER* con riduzione ha un'accuratezza di poco superiore alla versione senza riduzione. In generale, la differenza abissale fra il numero di parametri delle due giustifica l'impiego della versione con riduzione del feature extractor, poiché meno dipendente dalla memoria.

Dai valori di accuratezza presenti nelle Tabelle 3.2 e 3.3, si può dedurre una maggiore plasticità del metodo *DER* rispetto agli altri algoritmi FT. Infatti, le classi di CIFAR-100 possono essere estremamente diverse tra loro, rendendo inutilizzabile la conoscenza appresa dal modello nei passaggi incrementali precedenti. L'accuratezza di *DER*, nel tempo, diminuisce leggermente, indicando che il modello è in grado di riconoscere correttamente anche le nuove classi. Questa plasticità è probabilmente dovuta ai nuovi feature extractor aggiunti durante l'espansione del modello. D'altra parte, negli algoritmi FT, la plasticità viene limitata dalla KD. Sebbene quest'ultima sia fondamentale per mantenere la conoscenza pregressa, essa può impedire al modello di apprendere correttamente le feature delle nuove classi. Tale risultato supporta le considerazioni teoriche sulla plasticità degli algoritmi MG. Sebbene anche gli algoritmi FT siano, in generale, plastici, gli algoritmi MG introducono nuovi parametri dedicati alle nuove classi introdotte, permettendo l'apprendimento di feature anche molto diverse da quelle conosciute. Queste considerazioni indicano che, in generale, gli algoritmi MG sono più plastici degli algoritmi FT. Ciò non viene espressamente riportato nella teoria, ma è deducibile dagli esperimenti descritti.

6.2.3 *DeeSIL*

Nei risultati degli esperimenti riportati in Figura 4.2 si ottiene che *DeeSIL* ottiene dei migliori risultati di *iCaRL* e *LwF*. Ciò è in disaccordo con quanto riportato nella Tabella 6.1: infatti, i metodi FT dovrebbero, in generale, ottenere migliori risultati dei metodi FR. Però, questi risultati hanno delle fondamenta teoriche. *LwF* è un algoritmo datato e non utilizza istanze delle classi passate durante l'addestramento per mantenere la conoscenza pregressa. Al contrario, *DeeSIL* utilizza tale memoria per allenare i classificatori. Di conseguenza, *DeeSIL* è in grado di contrastare il CF più efficacemente di *LwF*, come verificabile dai risultati ottenuti. D'altro canto, spiegare perché *DeeSIL* performa meglio di *iCaRL* non è immediato, il quale sfrutta la memoria delle classi precedenti. La motivazione più plausibile è l'addestramento del DFE con una grande quantità di classi nello step non incrementale. Infatti, come indicato nella Sezione 4.1.2, gli esperimenti si sono svolti addestrando il DFE con 100 e 1000 classi di ImageNet e 1000 classi da un dataset Flickr. In tutti i casi, le feature apprese in S_0 sono state riutilizzate negli step futuri, permettendo al modello di ottenere accuratezze più elevate di un modello addestrato con un algoritmo FT. Tale risultato è in accordo con la teoria, confermando l'importanza dell'addestramento base in un algoritmo FR al fine di ottenere alte accuratezze.

6.2.4 *DSLDA*

Essendo *DSLDA* un algoritmo FR, ci si aspetta che esso ottenga, in generale, performance peggiori di algoritmi FT. Infatti, il solo fully-connected layer F viene riaddestrato negli step incrementali successivi, mentre il feature extractor G viene fissato. Analogamente a quanto trattato nella Sezione 6.2.3, si ha che l'algoritmo FR *DSLDA* ottiene valori di accuratezza maggiori di un algoritmo FT *iCaRL*. Anche in questo caso, si può attribuire tale fenomeno al fatto che il modello ha riutilizzato correttamente delle feature apprese nello step non incrementale. Questo risultato supporta quanto descritto nella teoria.

6.2.5 *BiC*

Gli esperimenti riportati nella Sezione 5.1.2 mettono a confronto solamente algoritmi FT. Di conseguenza, tutti dovrebbero dovrebbero essere caratterizzati da alti valori di accuratezza. *BiC* ottiene i migliori punteggi, confermando l'efficacia del bias correction layer nel prevenire il bias verso le nuove classi.

Per quanto riguarda la scalabilità, l'esperimento su ImageNet-1K riportato in Tabella 5.1 mette in evidenza la capacità di *BiC* di apprendere un grande numero di classi nel tempo. Infatti, gli algoritmi FT godono, in generale, di un'ottima scalabilità se la memoria disponibile è sufficiente. Di conseguenza, tale osservazione è in accordo con la teoria.

Infine, va sottolineato che *BiC* non può funzionare senza l'ausilio di una memoria per conservare le immagini delle classi precedenti, le quali risultano indispensabili per la costruzione del validation set. In tale circostanza, il modello non è in grado di apprendere α e β, e di conseguenza, non può correggere il bias relativo alle nuove classi. Pertanto, si può affermare che *BiC* dipende completamente dalla memoria e risulta incapace di operare in sua assenza. Questa considerazione è in accordo con quanto riportato nella teoria.

6.2.6 *SS-IL*

Come è visibile dalle Tabelle 5.3 e 5.4, *SS-IL* ottiene le migliori performance nella quasi totalità dei casi. Anch'esso, come *BiC*, viene comparato con soli algoritmi FT. L'algoritmo ottiene ottimi valori di accuratezza e scalabilità. Infatti, l'algoritmo gode di alta accuratezza su Landmark-v2-10K, apprendendo fino a 10 000 classi. Inoltre, poiché l'algoritmo utilizza la TKD, esso non può funzionare correttamente senza memoria disponibile, rendendolo fortemente dipendente dalla memoria. Di conseguenza, tali risultati verificano quanto discusso nella teoria.

6.3 Applicazioni

Come introdotto nella Sezione 1.1, lo sviluppo di entità in grado di apprendere in modo incrementale ha dei notevoli risvolti pratici. Di seguito si trattano delle possibili applicazioni del CIL calati nel contesto del riconoscimento di immagini. Viene discusso, per ogni applicazione, quale tipo di algoritmo potrebbe essere utilizzato.

6.3.1 Riconoscimento facciale

Nel problema del riconoscimento facciale [52, 66] si vuole realizzare un agente in grado di attribuire l'immagine di un volto a una determinata persona (i.e. classe). Nel caso del CIL, tale apprendimento deve essere incrementale, cioè i volti devono essere appresi nel tempo. È chiaro che l'entità non deve scordare i volti appresi nel passato.

Nel caso in cui i volti da apprendere siano in grande quantità (e.g. dei dipendenti di una azienda) potrebbe essere opportuno impiegare degli algoritmi MG o FT. Infatti, essi sarebbero in grado di apprendere una grande varietà e quantità di volti, permettendo una elevata accuratezza e scalabilità. Inoltre, le alte performance rendono gli algoritmi MG e FT adatti in applicazioni di riconoscimento facciale legate alla autenticazione, nelle quali la sicurezza è la massima priorità. In tale contesto, *Tree-CNN* potrebbe ottenere ottime performance. Infatti, l'algoritmo potrebbe suddividere i volti in una struttura gerarchica legata all'età, all'etnia, al sesso, etc. Nel caso in cui si decidesse di utilizzare

tali algoritmi, si potrebbe inserire il modello in una macchina remota con elevate capacità di calcolo e disponibilità di memoria.

D'altro canto, se i volti da apprendere sono pochi (e.g. di una famiglia) e le disponibilità computazionali sono limitate, si potrebbe optare per degli algoritmi FR, in generale meno scalabili e plastici. Le feature apprese nello step non incrementale sarebbero facilmente riutilizzabili nel futuro (e.g. tratti somatici relativi al naso, occhi, etc) e il solo classificatore verrebbe riaddestrato per distinguere un volto dagli altri. Tuttavia, bisogna considerare che un tale modello potrebbe presentare prestazioni scadenti se, ad esempio, si inserisce il volto di una persona anziana dopo un addestramento iniziale su volti di persone giovani, poiché il primo potrebbe avere caratteristiche tipiche che non sono presenti nei secondi (e.g. rughe) e viceversa (e.g. acne). Una soluzione consisterebbe nell'addestrare lo step non incrementale su molti volti di diverso tipo, come riportato in Tabella 6.1. In tal caso, il modello sarebbe in grado di estrarre correttamente feature dalle immagini dei volti. Impiegando un algoritmo FR, il modello ottenuto sarebbe altamente tempestivo e fortemente indipendente dalla memoria.

6.3.2 Riconoscimento di pillole

Classificare le categorie di pillole da immagini del mondo reale è cruciale per diverse applicazioni intelligenti nel settore della salute [63]. In questo campo, le tecniche di riconoscimento di immagini consentono di identificare una pillola sconosciuta. Tuttavia, la corretta classificazione delle immagini di pillole è difficile a causa della grande varietà di forme, colori e fibra. Di conseguenza, un algoritmo FR potrebbe non apprendere feature sufficienti nello step non incrementale per riutilizzarle negli step futuri. D'altro canto, degli algoritmi FT e MG potrebbero performare meglio in tale contesto. Infatti, essi potrebbero acquisire le caratteristiche distintive delle nuove classi, poiché il loro feature extractor non è vincolato. Ad esempio, i nuovi feature extractor introdotti nell'espansione in *DER* potrebbero imparare nuove forme e tipi di pillole. D'altra parte, i risultati riportati nella Tabella 5.1 indicano che *BiC* è in grado di mantenere alte prestazioni pur acquisendo un gran numero di classi. In modo analogo, dalle Tabelle 5.3 e 5.4 emerge che *SS-IL* ottiene le migliori accuratezze nei dataset con 1000 e 10 000 classi. Considerando l'elevato numero di pillole classificabili, la scalabilità caratteristica di *BiC* e *SS-IL* risulterebbe estremamente utile in questo contesto.

6.3.3 Robot assistivi

Un robot assistivo deve essere in grado di classificare correttamente un oggetto in situazioni e ambienti dinamici [27] e dunque deve essere in grado di acquisire nuove capacità di riconoscimento nel corso del tempo. In particolare, essi devono essere in grado di classificare un oggetto in diversi contesti di illuminazione, angolazione, distanza e disordine dell'ambiente [27].

Se l'azione del robot è limitata a un solo ambiente (e.g. cucina di una casa) si potrebbe optare per un algoritmo FR. Questo robot non avrebbe bisogno di elevate risorse di calcolo o grandi quantità di memoria e potrebbe comunque ottenere elevate accuratezze nel riconoscimento degli oggetti all'interno di un contesto ristretto. In questo caso, sarebbe possibile impiegare l'algoritmo *DSLDA* grazie alla sua capacità di funzionare in modo indipendente dalla memoria disponibile. Inoltre, l'algoritmo consentirebbe una certa flessibilità del modello grazie all'aggiornamento della matrice di covarianza $\mathbf{\Sigma}$. Al contrario, *DeeSIL* non sarebbe una buona scelta per tale contesto. Infatti, esso necessita di memoria per conservare gli esemplari delle classi passate e non garantisce alcuna plasticità dopo lo step non incrementale.

D'altro canto, se il robot deve agire su un ambiente molto più ampio e soggetto a cambiamenti repentini, si dovrebbe impiegare un algoritmo MG o FT più plastico e scalabile. Analogamente a quanto riportato nella Sezione 6.3.2, *BiC* e *SS-IL* godono di una elevata scalabilità, e dunque sarebbero degli algoritmi adatti per apprendere un numero di classi elevato in un ambiente aperto. *Tree-CNN* potrebbe performare bene in tale contesto grazie alla sua capacità di organizzare le classi apprese in una struttura gerarchica. Infatti, come riportato nella Sezione 3.1.2, l'algoritmo ottiene ottimi risultati su CIFAR-100, il quale contiene immagini legate a molti ambienti distinti, come animali, veicoli, paesaggi, persone, etc.

6.4 Limitazioni dello studio

Nonostante gli esperimenti riportati abbiano un grande valore per comprendere un algoritmo e le sue caratteristiche, essi presentano delle lacune che impediscono una trattazione più completa. In particolare, molte pubblicazioni confrontano l'algoritmo proposto solo con pochi altri algoritmi di CIL, come *iCaRL* e *LwF* [7, 33, 78], che sono datati e non rappresentano più lo stato dell'arte. Poiché spesso vengono utilizzati criteri di accuratezza e dataset diversi, non è possibile veramente mettere a confronto due metodi che non fanno parte del medesimo esperimento. Ne consegue che spesso manca una comparazione diretta fra algoritmi recenti e allo stato dell'arte.

Inoltre, spesso manca un confronto sperimentale fra algoritmi di diverso tipo. Ad esempio, nell'esperimento in Sezione 3.1.2 non compare alcun algoritmo FR, mentre in quello in Sezione 4.1.4 non viene menzionato nemmeno un algoritmo MG. Inoltre, nei casi di esperimenti relativi ad algoritmi FT, si trascurano quasi del tutto gli algoritmi MG e FR, come si può verificare nelle Sezioni 5.1.2 e 5.2.4, oppure in altre pubblicazioni [10]. La scarsità di esperimenti che includano tutti i tipi di algoritmi di CIL in letteratura riduce l'ampiezza delle analisi e la comprensione dell'effettiva efficacia dei diversi approcci di apprendimento incrementale nel contesto del CIL.

Per approfondire la ricerca sul CIL, sarebbe opportuno condurre esperimenti con diversi algoritmi MG, FR e FT, confrontandoli direttamente utilizzando lo stesso framework di valutazione. In questo modo, sarà possibile evidenziare le analogie e le differenze tra di loro dal punto di vista sperimentale e individuare i contesti in cui un approccio risulta più adatto dell'altro.

6.5 Prospettive future

Di seguito si trattano alcune possibili prospettive future del Class Incremental Learning.

6.5.1 Sviluppo di approcci misti

Una possibile strada la ricerca potrebbe essere lo sviluppo di nuovi algoritmi ibridi che combinano le caratteristiche positive di Model-Growth, Fixed-Representation e Fine-Tuning, allo scopo di ottenere un approccio più flessibile e adattabile a un'ampia varietà di scenari.

Ad esempio, si potrebbe introdurre una componente di KD negli algoritmi Fixed-Representation, così da renderli più plastici e migliorarne le performance. Infatti, la deep representation verrebbe aggiornata tenendo conto delle classi nuove e passate congiuntamente. In tale contesto, si potrebbe limitare l'effetto della KD sul feature extractor, per mantenerlo molto simile a quello ottenuto nello step non incrementale e limitare il Catastrophic Forgetting.

Un ulteriore spunto nello sviluppo di approcci misti potrebbe riguardare l'utilizzo di componenti Fixed-Representation in algoritmi Model-Growth. Ad esempio, una volta aggiunta una nuova componente al modello, la si potrebbe fissare in stile Fixed-Representation. In questo modo, la conoscenza introdotta con una determinata espansione non verrebbe modificata da ulteriori step incrementali. Nuove classi non verrebbero apprese modificando componenti introdotte nel passato, ma solo grazie all'espansione del modello. In tal modo, l'algoritmo sarebbe molto più tempestivo di un algoritmo totalmente Model-Growth e potrebbe contrastare più efficacemente il Catastrophic Forgetting.

6.5.2 Tecniche di riduzione del bias e riformulazione della Knowledge Distillation

Si è visto come la cross-entropy loss crei nel modello un bias verso le nuove classi. Tale bias viene propagato dalla General Knowledge Distillation, senza prendere apposite misure per contrastarlo. La ricerca sul Class Incremental Learning potrebbe svilupparsi partendo da queste considerazioni. Per esempio, nel modello *BiC* è stata introdotta una soluzione lineare per ridurre il bias verso le nuove classi. Un'interessante prospettiva potrebbe consistere nello studio di approcci più sofisticati, come l'assegnazione di parametri α e β distinti per ogni classe o la modifica del modello del layer.

Inoltre, si potrebbero esaminare alternative definizioni della Knowledge Distillation, come la Task-wise Knowledge Distillation o quella proposta in [70], allo scopo di trovare soluzioni possibili al bias verso le nuove classi. Alcuni approcci Fine-Tuning potrebbero persino eliminare del tutto la Knowledge

Distillation [9], mentre comunque aggiornano l'intero modello a ogni step incrementale mediante altri meccanismi.

6.5.3 Limitazione dell'espansione

Nella tesi è stato osservato come gli algoritmi Model-Growth ottengano prestazioni eccellenti grazie all'espansione del modello. La ricerca su questi algoritmi dovrebbe ora concentrarsi sulla riduzione del numero di parametri pur mantenendo elevate le prestazioni. A questo proposito, potrebbe essere interessante trarre ispirazione da *DER* e sviluppare ulteriori tecniche di espansione dinamica. Altri algoritmi Model-Growth potrebbero adottare meccanismi per limitare il numero di nuovi parametri introdotti, migliorando la semplicità e la tempestività.

Per esempio, *Tree-CNN* potrebbe impiegare parametri apprendibili che controllano il numero massimo di figli consentiti per un dato nodo. Questo meccanismo potrebbe essere implementato addestrando tali parametri su un validation set, in modo simile a quanto accade in *BiC* durante il calcolo dei parametri α e β. In questo modo, si potrebbe ottenere una riduzione dei parametri senza compromettere le prestazioni del modello.

Tabella 6.1: Analisi dei tre tipi di algoritmi di CIL in riferimento alle proprietà ricercate.

	Model-Growth	Fixed-Representation	Fine-Tuning
Semplicità	La semplicità diminuisce nel tempo dal momento che nuovi parametri vengono aggiunti al modello [76].	Il modello è fissato in S_0. Nei casi più semplici [7] i soli parametri aggiunti sono quelli dei classifier. In contesti più avanzati [42] vengono aggiunti ulteriori parametri per migliorare le performance.	Il numero di parametri è marginalmente modificato fra gli stati incrementali [16, 92].
Indipendenza dalla memoria	La memoria viene utilizzata per inserire i nuovi parametri invece delle immagini di classi passate [3, 78].	Poiché il modello non viene aggiornato, gli algoritmi FR hanno una scarsa dipendenza dalla memoria di classi passate [7].	Tali algoritmi hanno una forte dipendenza dalla memoria [10] per memorizzare istanze di classi passate [16, 77, 92] o proprietà statistiche di esse [8].
Accuratezza	Se la crescita è limitata, tali algoritmi ottengono una accuratezza inferiore agli algoritmi FR e FT [20]. Se il modello può crescere di molto, allora l'accuratezza diventa simile a quella del modello fully-trained [78].	L'accuratezza è, in generale, inferiore a quella degli algoritmi FT [10] perché il modello non viene aggiornato fra gli stati incrementali. Se in S_0 il modello viene allenato su un dataset grande l'accuratezza è alta [7].	La knowledge distillation rende gli algoritmi FT molto performanti [1, 55, 77, 92]. Se la memoria disponibile è elevata, l'accuratezza si avvicina a quella del modello fully-trained [10].
Tempestività	È necessario ri-allenare il modello a ogni S_t [20], dunque tali algoritmi sono meno tempestivi di quelli FR.	Dal momento che in ogni S_t è necessario allenare solo il classifier, tali algoritmi hanno una alta tempestività e nuova conoscenza può essere integrata frequentemente [34].	È necessario ri-allenare il modello a ogni S_t [10], dunque tali algoritmi sono meno tempestivi di quelli FR.
Plasticità	Gli algoritmi MG sono plastici poiché, aumentando la grandezza del modello, è possibile apprendere feature anche molto diverse da quelle presenti in $S_0, \ldots, S_{t-1}$ [3]. Si vuole rendere gli algoritmi plastici minimizzando il numero di parametri addizionali [20].	La plasticità è limitata dal momento che le feature vengono apprese in S_0.	Gli algoritmi FT sono plastici dal momento che l'intero modello viene allenato sulle nuove istanze, favorendo l'apprendimento delle feature a esse relative.
Scalabilità	Tali algoritmi scalano bene a patto che il modello abbia la possibilità di crescere sufficientemente [10].	Se i parametri appresi in S_0 possono essere riutilizzati, gli algoritmi FR hanno una ottima scalabilità [10].	Gli algoritmi FT sono scalabili se la memoria impiegata per conservare istanze di $S_0, \ldots, S_{t-1}$ è sufficiente [16, 77].

Capitolo 7

Conclusione

Si è visto come numerose applicazioni richiedano di aggiornare le capacità di riconoscimento del modello con l'introduzione di classi inedite. Tale esigenza è dovuta all'impiego del sistema in ambienti dinamici, nei quali non è possibile definire a priori l'insieme di classi da dover riconoscere, né tantomeno la sua cardinalità. In particolare, nella classificazione di immagini, un sistema di apprendimento incrementale è imprescindibile in diversi contesti e applicazioni, come nel riconoscimento di volti, di pillole o di oggetti all'interno di un ambiente casalingo. In ciascuno di questi casi, è difficile prevedere le classi che il modello necessita di apprendere per tutta la durata del suo utilizzo. Rieffettuare l'addestramento sulle sole nuove classi provoca il Catastrophic Forgetting, mentre riallenare il modello su un dataset comprendente classi nuove e passate è oneroso in termini di tempo e costo computazionale. Data l'impraticità di tali soluzioni, nasce l'esigenza di sviluppare degli approcci che permettano a un sistema di riconoscimento visivo di apprendere ulteriori classi e, al contempo, mantenere la conoscenza su quelle incontrate in precedenza.

La tesi ha fornito una panoramica sulle soluzioni allo stato dell'arte al problema del Class Incremental Learning, applicato alla classificazione di immagini. Si è formalizzato il problema e sono stati presentati i tre principali tipi di approcci impiegati, ossia: Model-Growth, Fixed-Representation e Fine-Tuning. Sono state, inoltre, definite delle proprietà ricercate negli algoritmi di Class Incremental Learning, proponendo un framework di valutazione basato su di esse. In seguito, si sono analizzati i risultati degli esperimenti, che hanno confermato le caratteristiche attese. Infine, sono state presentate delle applicazioni reali degli approcci trattati. L'impiego di un algoritmo di Class Incremental Learning in queste applicazioni risulterebbe decisivo, in quanto queste soluzioni sono esenti dalle problematiche legate al riaddestramento parziale o totale del modello. Tali approcci permettono di ottenere tempi di aggiornamento del sistema ragionevoli e accuratezze elevate su tutte le classi introdotte, rendendoli necessari quando le capacità di riconoscimento richieste aumentano in modo imprevedibile. In particolare, si è visto che in ognuna delle applicazioni trattate alcuni approcci risulterebbero più adatti rispetto ad altri. Ad esempio, nei sistemi di riconoscimento facciale è necessario mantenere alte accuratezze per motivi legati alla sicurezza, mentre non è indispensabile aggiornare tempestivamente il modello all'introduzione di nuovi volti. D'altro canto, è fondamentale che un sistema di classificazione immerso in un ambiente dinamico e in costante evoluzione abbia un rapido aggiornamento delle classi riconoscibili. Inoltre, in entrambi i casi, l'importanza di altre proprietà, come scalabilità e plasticità, dipende dall'ordine di grandezza del numero di classi da apprendere e da quanto queste siano diverse fra loro.

Al netto delle considerazioni trattate finora, si conclude che la scelta dell'approccio migliore dipende dalle specifiche esigenze del caso d'uso. Ogni applicazione necessita, quindi, di un differente algoritmo di Class Incremental Learning, come approfondito nella Sezione 6.3. Inoltre, dal momento che esiste del margine di miglioramento fra gli algoritmi allo stato dell'arte e il modello fully-trained, il problema del Class Incremental Learning è ancora aperto. Di conseguenza, lo studio sull'argomento non è ancora concluso. Infatti, periodicamente vengono sviluppati nuovi algoritmi con l'obiettivo di ottenere soluzioni sempre più efficaci ed efficienti rispetto allo stato dell'arte.

Lo studio futuro potrebbe concentrarsi sugli approcci Fine-Tuning. Poiché questi sono i più diffusi in letteratura, ogni anno vengono proposti nuovi metodi e approcci che fanno uso della Knowledge Distillation. In particolare, potrebbe essere interessante studiare il modo in cui è possibile ridurre al minimo la dipendenza dalla memoria di tali algoritmi, pur mantenendo ottime performance. Inoltre, si

potrebbe esplorare il sottoinsieme più ristretto degli algoritmi Fine-Tuning che utilizzano la memoria per conservare le proprietà statistiche delle classi passate, anziché le istanze degli step incrementali precedenti.

Un'altra possibilità per lo studio futuro riguarda l'analisi degli *ablation study* in letteratura. Essi analizzano le performance di un algoritmo rimuovendo alcune componenti, con lo scopo di capirne meglio il contributo sull'intero sistema. Questa analisi sarebbe fondamentale per identificare le parti cruciali dell'algoritmo e valutarne l'importanza relativa.

Gli approfondimenti in tali ambiti sarebbero estremamente preziosi nel contesto della ricerca accademica futura, arricchendola con una prospettiva più completa e approfondita sul campo del Class Incremental Learning.

Bibliografia

[1] Hongjoon Ahn et al. "SS-IL: Separated Softmax for Incremental Learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Ott. 2021, pp. 844–853.

[2] Saad Albawi, Tareq Abed Mohammed e Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[3] Rahaf Aljundi et al. "Memory aware synapses: Learning what (not) to forget". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 139–154.

[4] Md Zahangir Alom et al. "The history began from alexnet: A comprehensive survey on deep learning approaches". In: *arXiv preprint arXiv:1803.01164* (2018).

[5] Brian Babcock et al. "Models and issues in data stream systems". In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2002, pp. 1–16.

[6] Kunal Banerjee et al. "Exploring alternatives to softmax function". In: *arXiv preprint arXiv:2011.11538* (2020).

[7] Eden Belouadah e Adrian Popescu. "DeeSIL: Deep-Shallow Incremental Learning." In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. Set. 2018.

[8] Eden Belouadah e Adrian Popescu. "IL2M: Class Incremental Learning With Dual Memory". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Ott. 2019.

[9] Eden Belouadah e Adrian Popescu. "Scail: Classifier weights scaling for class incremental learning". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2020, pp. 1266–1275.

[10] Eden Belouadah, Adrian Popescu e Ioannis Kanellos. "A comprehensive study of class incremental learning algorithms for visual tasks". In: *Neural Networks* 135 (2021), pp. 38–54. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608020304202.

[11] Yoshua Bengio, Ian Goodfellow e Aaron Courville. *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA, 2017.

[12] Monica Bianchini e Franco Scarselli. "On the complexity of neural network classifiers: A comparison between shallow and deep architectures". In: *IEEE transactions on neural networks and learning systems* 25.8 (2014), pp. 1553–1565.

[13] Tommaso Boccato et al. "In the Depths of Hyponymy: A Step towards Lifelong Learning". In: *Proceedings of the International Conference on Autonomic and Autonomous Systems*. 2020, pp. 103–109.

[14] Joost Broekens, Marcel Heerink, Henk Rosendal et al. "Assistive social robots in elderly care: a review". In: *Gerontechnology* 8.2 (2009), pp. 94–103.

[15] Olivier Caelen. "A Bayesian interpretation of the confusion matrix". In: *Annals of Mathematics and Artificial Intelligence* 81.3-4 (2017), pp. 429–450.

[16] Francisco M. Castro et al. "End-to-End Incremental Learning". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Set. 2018.

[17] Arslan Chaudhry et al. “Continual learning with tiny episodic memories”. In: *Workshop on Multi-Task and Lifelong Reinforcement Learning*. 2019.

[18] Zhi Chen e Pin-Han Ho. “Global-connected network with generalized ReLU activation”. In: *Pattern Recognition* 96 (2019), p. 106961.

[19] Xu Cheng et al. “Explaining knowledge distillation by quantifying the knowledge”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 12925–12935.

[20] Matthias De Lange et al. “Continual learning: A comparative study on how to defy forgetting in classification tasks”. In: *arXiv preprint arXiv:1909.08383* 2.6 (2019), p. 2.

[21] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[22] Xinyang Deng et al. “An improved method to construct basic probability assignment based on the confusion matrix for classification problem”. In: *Information Sciences* 340 (2016), pp. 250–261.

[23] Simone Disabato e Manuel Roveri. “Learning convolutional neural networks in presence of concept drift”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[24] Arthur Douillard et al. “Podnet: Pooled outputs distillation for small-tasks incremental learning”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*. Springer. 2020, pp. 86–102.

[25] Issam El Naqa e Martin J Murphy. *What is machine learning?* Springer, 2015.

[26] Andre Esteva et al. “Deep learning-enabled medical computer vision”. In: *NPJ digital medicine* 4.1 (2021), p. 5.

[27] Fan Feng et al. “Challenges in Task Incremental Learning for Assistive Robotics”. In: *IEEE Access* 8 (2020), pp. 3434–3441. DOI: 10.1109/ACCESS.2019.2955480.

[28] Alexander Gepperth e Barbara Hammer. “Incremental learning algorithms and applications”. In: *European Symposium on Artificial Neural Networks (ESANN)*. Bruges, Belgium, 2016. URL: https://hal.science/hal-01418129.

[29] Alexandru Lucian Ginsca et al. “Large-scale image mining with flickr groups”. In: *MultiMedia Modeling: 21st International Conference, MMM 2015, Sydney, NSW, Australia, January 5-7, 2015, Proceedings, Part I 21*. Springer. 2015, pp. 318–334.

[30] Ian J Goodfellow et al. “An empirical investigation of catastrophic forgetting in gradient-based neural networks”. In: *arXiv preprint arXiv:1312.6211* (2013).

[31] Daniel H Grollman e Odest Chadwicke Jenkins. “Sparse incremental learning for interactive robot control policy estimation”. In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 3315–3320. DOI: 10.1109/ROBOT.2008.4543716.

[32] Tyler L Hayes, Nathan D Cahill e Christopher Kanan. “Memory efficient experience replay for streaming learning”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9769–9776.

[33] Tyler L. Hayes e Christopher Kanan. “Lifelong Machine Learning With Deep Streaming Linear Discriminant Analysis”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. Giu. 2020.

[34] Tyler L. Hayes et al. “REMIND Your Neural Network to Prevent Catastrophic Forgetting”. In: *Computer Vision – ECCV 2020*. A cura di Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 466–483.

[35] Kaiming He e Jian Sun. “Convolutional Neural Networks at Constrained Time Cost”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2015.

[36] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2016.

[37] Yu-Lin He et al. “Determining the optimal temperature parameter for Softmax function in reinforcement learning”. In: *Applied Soft Computing* 70 (2018), pp. 80–85.

[38] Saihui Hou et al. “Learning a unified classifier incrementally via rebalancing”. In: *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*. 2019, pp. 831–839.

[39] Hidenori Ide e Takio Kurita. “Improvement of learning for CNN with ReLU activation by sparse regularization”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2684–2691. DOI: 10.1109/IJCNN.2017.7966185.

[40] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[41] Sagar V Kamarthi e Stefan Pittner. “Accelerating neural network training using weight extrapolations”. In: *Neural networks* 12.9 (1999), pp. 1285–1299.

[42] Ronald Kemker e Christopher Kanan. “Fearnet: Brain-inspired model for incremental learning”. In: *arXiv preprint arXiv:1711.10563* (2017).

[43] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[44] Igor Kononenko. “Machine learning for medical diagnosis: history, state of the art and perspective”. In: *Artificial Intelligence in medicine* 23.1 (2001), pp. 89–109.

[45] Alex Krizhevsky, Geoffrey Hinton et al. “Learning multiple layers of features from tiny images”. In: (2009).

[46] Alex Krizhevsky, Ilya Sutskever e Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. A cura di F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[47] Anders Krogh. “What are artificial neural networks?” In: *Nature biotechnology* 26.2 (2008), pp. 195–197.

[48] Yann LeCun, Yoshua Bengio e Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[49] Cecilia S Lee e Aaron Y Lee. “Clinical applications of continual learning machine learning”. In: *The Lancet Digital Health* 2.6 (2020), e279–e281.

[50] Kibok Lee et al. “Overcoming catastrophic forgetting with unlabeled data in the wild”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 312–321.

[51] Mu Li et al. “Efficient mini-batch training for stochastic optimization”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 661–670.

[52] Yinglong Li. “Research and application of deep learning in image recognition”. In: *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*. IEEE. 2022, pp. 994–999.

[53] Zewen Li et al. “A survey of convolutional neural networks: analysis, applications, and prospects”. In: *IEEE transactions on neural networks and learning systems* (2021).

[54] Zhizhong Li e Derek Hoiem. “Learning without Forgetting”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2018), pp. 2935–2947. DOI: 10.1109/TPAMI.2017.2773081.

[55] Zhizhong Li e Derek Hoiem. “Learning without forgetting”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[56] Yaoyao Liu, Bernt Schiele e Qianru Sun. “Adaptive aggregation networks for class-incremental learning”. In: *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*. 2021, pp. 2544–2553.

[57] Vincenzo Lomonaco e Davide Maltoni. “CORe50: a New Dataset and Benchmark for Continuous Object Recognition”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. A cura di Sergey Levine, Vincent Vanhoucke e Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, nov. 2017, pp. 17–26.

[58] Jie Lu et al. “Learning under concept drift: A review”. In: *IEEE transactions on knowledge and data engineering* 31.12 (2018), pp. 2346–2363.

[59] Yong Luo et al. “An appraisal of incremental learning methods”. In: *Entropy* 22.11 (2020), p. 1190.

[60] Warren S McCulloch e Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[61] Chen Meng et al. “Training deeper models by GPU memory optimization on TensorFlow”. In: *Proc. of ML Systems Workshop in NIPS*. Vol. 7. 2017.

[62] Dunja Mladenić et al. “Feature selection using linear classifier weights: interaction with classification models”. In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 2004, pp. 234–241.

[63] Tung-Trong Nguyen et al. “Multi-stream Fusion for Class Incremental Learning in Pill Image Classification”. In: *Proceedings of the Asian Conference on Computer Vision*. 2022, pp. 4565–4580.

[64] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[65] Keiron O’Shea e Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[66] Seiichi Ozawa et al. “Incremental learning of feature space and classifier for face recognition”. In: *Neural Networks* 18.5-6 (2005), pp. 575–584.

[67] Thomas Paine et al. “Gpu asynchronous stochastic gradient descent to speed up neural network training”. In: *arXiv preprint arXiv:1312.6186* (2013).

[68] Shaoning Pang, S. Ozawa e N. Kasabov. “Incremental linear discriminant analysis for classification of data streams”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 35.5 (2005), pp. 905–914. DOI: 10.1109/TSMCB.2005.847744.

[69] German I Parisi et al. “Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization”. In: *Frontiers in neurorobotics* (2018), p. 78.

[70] Wonpyo Park et al. “Relational knowledge distillation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3967–3976.

[71] Kitsuchart Pasupa e Wisuwat Sunhem. “A comparison between shallow and deep architecture classifiers on small dataset”. In: *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. IEEE. 2016, pp. 1–6.

[72] Mary Phuong e Christoph H Lampert. “Distillation-based training for multi-exit architectures”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1355–1364.

[73] R. Polikar et al. “Learn++: an incremental learning algorithm for supervised neural networks”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.4 (2001), pp. 497–508. DOI: 10.1109/5326.983933.

[74] Vinay Venkatesh Ramasesh, Aitor Lewkowycz e Ethan Dyer. “Effect of scale on catastrophic forgetting in neural networks”. In: *International Conference on Learning Representations*. 2021.

[75] Jesse Read et al. “Batch-incremental versus instance-incremental learning in dynamic and evolving data”. In: *Advances in Intelligent Data Analysis XI: 11th International Symposium, IDA 2012, Helsinki, Finland, October 25-27, 2012. Proceedings 11*. Springer. 2012, pp. 313–323.

[76] Sylvestre-Alvise Rebuffi, Hakan Bilen e Andrea Vedaldi. “Efficient parametrization of multi-domain deep neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8119–8127.

[77] Sylvestre-Alvise Rebuffi et al. “iCaRL: Incremental Classifier and Representation Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Lug. 2017.

[78] Deboleena Roy, Priyadarshini Panda e Kaushik Roy. "Tree-CNN: A hierarchical Deep Convolutional Neural Network for incremental learning". In: *Neural Networks* 121 (2020), pp. 148–160. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.09.010. URL: https://www.sciencedirect.com/science/article/pii/S0893608019302710.

[79] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[80] Paul Sajda. "Machine learning for detection and diagnosis of disease". In: *Annu. Rev. Biomed. Eng.* 8 (2006), pp. 537–565.

[81] Joan Serra et al. "Overcoming catastrophic forgetting with hard attention to the task". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4548–4557.

[82] Shai Shalev-Shwartz e Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[83] Karen Simonyan e Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[84] Daniel Strigl, Klaus Kofler e Stefan Podlipnig. "Performance and Scalability of GPU-Based Convolutional Neural Networks". In: *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. 2010, pp. 317–324. DOI: 10.1109/PDP.2010.43.

[85] Le Sun et al. "Few-Shot Class-Incremental Learning for Medical Time Series Classification". In: *IEEE Journal of Biomedical and Health Informatics* (2023).

[86] Bart Thomee et al. "The New Data and New Challenges in Multimedia Research". In: *CoRR* abs/1503.01817 (2015). arXiv: 1503.01817. URL: http://arxiv.org/abs/1503.01817.

[87] Dimitris Tsipras et al. "From imagenet to image classification: Contextualizing progress on benchmarks". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9625–9635.

[88] Athanasios Voulodimos et al. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* 2018 (2018).

[89] Karl Weiss, Taghi M Khoshgoftaar e DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[90] Max Welling. "Herding dynamical weights to learn". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 1121–1128.

[91] Tobias Weyand et al. "Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2575–2584.

[92] Yue Wu et al. "Large Scale Incremental Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2019.

[93] Tianjun Xiao et al. "Error-Driven Incremental Learning in Deep Convolutional Neural Network for Large-Scale Image Classification". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 177–186. ISBN: 9781450330633. DOI: 10.1145/2647868.2654926. URL: https://doi.org/10.1145/2647868.2654926.

[94] Rikiya Yamashita et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into imaging* 9 (2018), pp. 611–629.

[95] Shipeng Yan, Jiangwei Xie e Xuming He. "DER: Dynamically Expandable Representation for Class Incremental Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2021, pp. 3014–3023.

[96] Peng Zhang, Xingquan Zhu e Yong Shi. "Categorizing and mining concept drifting data streams". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 812–820.

[97] Bowen Zhao et al. "Maintaining discrimination and fairness in class incremental learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 13208–13217.

[98] Fuzhen Zhuang et al. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.