Classificatore d'immagini: addestramento di una CNN basato su CIFAR-10 e CIFAR-100

334568 - Andrea Baldinelli - andrea.baldinelli@studenti.unipg.it

***Sommario*—Quello che un decennio fa sembrava un futuro lontanissimo, invece è diventato già un solido presente. Con l'avvento di calcolatori con capacità di calcolo via via sempre maggiori, è divenuto possibile svolgere compiti molto complessi; basti osservare la repentina crescita di applicazioni nell'ambito del *machine learning* e della *computer vision*. Dallo sviluppo di questi due mondi, soprattutto dalla loro intersezione che dà vita al *deep learning*, molti compiti sono diventati sempre meno impossibili da svolgere: uno di questi è sicuramente quello della *classificazione di immagini* per mezzo di reti neurali *convoluzionali*.**

In questo report si propone una possibile architettura di rete convoluzionale (addestrata su due dataset molto famosi: *CIFAR-10* e *CIFAR-100*) in grado di adempire al problema della classificazione di immagini, andando a descriverne struttura e risultati ottenuti ma anche possibili sviluppi futuri

I. Introduzione

Prima di affrontare un qualsivoglia discorso tecnico e applicativo, è giusto definire un po' di scenario in cui si sta andando ad operare. Classificare un'immagine in funzione del contenuto per un essere umano può sembrare un compito molto semplice; questo perché implicitamente una persona, difronte ad un'immagine, fa molte considerazioni di tipo semantico "in automatico". Facendo un esempio: per capire se l'immagine contiene un'automobile, l'essere umano inizia a cercare nell'immagine se sono presenti le *caratteristiche* che definiscono un'auto, le quali possono essere la dimensione, il numero di ruote e di porte e così via. Il punto è che in realtà, dietro a questa operazione, si celano molti automatismi che il cervello umano permette di fare. Due esempi possono essere la valutazione della dimensione dell'oggetto in funzione del contesto circostante oppure la valutazione della "profondità" degli elementi nell'immagine. Tante cose che per un essere umano sono scontate, che invece per un calcolatore non lo sono.

Come prima soluzione al problema, si potrebbe pensare all'utilizzo di una rete neurale "classica" (chiamata anche *densa* o *fully connected*) che in input riceve un'immagine, matematicamente parlando è un *tensore*, che viene vettorizzata e poi fatta passare attraverso un certo numero di *hidden layer* per poi ottenere in uscita la classe di appartenenza predetta. In questo modo, grazie ai neuroni negli hidden layer, la rete "apprende" concetti Anche se teoricamente il modello potrebbe pure funzionare, in realtà cela alcune problematiche. Per definizione di rete neurale densa, ogni neurone (considerando l'ingresso come *input layer*) di un layer della rete deve essere connesso con tutti i neuroni del layer successivo; questo significa, visto l'iniziale vettorizzazione dell'immagine, ad un numero molto elevato di parametri da dover ottimizzare ad ogni step. Una possibile soluzione potrebbe essere quello di non utilizzare un numero di hidden layer alto e con un numero sufficiente di neuroni; in realtà questa soluzione cela ancora altri problemi di ottimizzazione. Infatti i parametri della rete neurale vengono ottenuti per mezzo della risoluzione di un problema di ottimizzazione: si è dimostrato sperimentalmente che la probabilità che l'ottimizzazione termini in un minimo locale è alta; il problema è che c'è sensibile discrepanza tra un minimo locale ed uno globale.

In questo scenario hanno trovato la loro consacrazione le reti neurali convoluzionali (brevemente chiamate *CNN*) che sfruttano gli stessi concetti alla base delle reti neurali classiche ma presentano delle novità che ne hanno decretato il successo:

- *sparse interaction:* ogni neurone di un layer intermedio è connesso solo ad *alcuni* neuroni del layer precedente. Il numero di interazioni è determinato dalla definizione del *kernel*;
- *parameter sharing:* se nelle reti classiche ogni neurone aveva il proprio vettore di parametri, con le CNN i neuroni di un layer condividono lo stesso set di parametri. Questo garantisce che un'eventuale punto di interesse viene "trovato" indipendentemente dalla posizione dello stesso nell'immagine;
- *equivariant representation:* il risultato dell'uscita di un neurone cambia al cambiare dei parametri fotometrici dell'immagine.

Queste novità permettono di realizzare CNN più profonde in quanto il numero di parametri si è nettamente ridotto. Questo significa che, concettualmente, la rete nei primi layer si concentra ad estrapolare informazioni a basso livello (ovvero spigoli e bordi degli oggetti) che si vanno via via componendo andando in profondità, creando così informazioni di più alto livello. Lo strutturare reti con un numero elevato di livelli risolve implicitamente il problema dell'ottimizzazione dei parametri: sperimentalmente si è osservato che la *funzione obiettivo* di reti neurali profonde presenta minimi locali che differiscono di molto poco dal minimo globale!

Nonostante quanto viene scritto un paio di righe sopra, l'architettura di rete convoluzionale che viene proposta in questo report non è molto profonda (rispetto a quelle già presenti); per completezza vengono successivamente riportate alcune tra le architetture che rappresentano lo *stato dell'arte* nel contesto della classificazione d'immagine.

Come base d'addestramento, la medesima rete viene addestrata su due dataset molto famosi nel settore: CIFAR-10 e la sua estensione CIFAR-100 [1]. Sono dataset resi Nel seguito vengono descritte le prestazioni ottenute dalla rete su questi due dataset.

II. Altri Studi

L'*image classification* è un tema molto ricco di pubblicazioni, visto i suoi innumerevoli impieghi in molte altre applicazioni. Sembra pertanto doveroso dover effettuare un breve resoconto delle pubblicazioni più importanti (nota bene: tutte le pubblicazioni riguardano reti addestrate su CIFAR-10 e CIFAR-100).

Una delle prime reti convoluzionali proposte fu quella di Goodfellow et al. [2] (2013) che presentava una nuova funzione di attivazione, chiamata *maxout*, che garantiva che il gradiente "fluisse all'indietro" durante l'aggiornamento e non andava a deteriorarsi tendendo a scomparire. Questo quindi permetteva la realizzazione di reti più profonde. La rete presentata ha mostrato su CIFAR-10 un'accuratezza del 90.65% mentre su CIFAR-100 del 61.43%.

Un altra pubblicazione degna di nota è quella di Graham [3] (2015) che proponeva l'utilizzo del *fractional max-pooling*: rispetto al max-pooling classico, questo tipo di trasformazione va a definire delle zone di pooling all'interno dell'immagine in modo aleatorio o pseudo-aleatorio, che possono pure sovrapporsi fra di loro. In più il ridimensionamento dell'input che permette di effettuare può essere determinato in un intervallo compreso tra 1,2, ove 2 si intende il sotto campionamento dell'immagine ad un quarto della dimensione originale. L'accuratezza si assesta al 96.53% su CIFAR-10 e al 73.61% su CIFAR-100.

Un notevole passo in avanti in termini di architettura e profondità della stessa sono stati portati da Huang et al. [4] (2017) con l'introduzione delle *DenseNet*. Già Goodfellow [2] proponeva la sua soluzione al problema del gradiente che tende a scomparire in reti con un elevato numero di layer; quello che viene introdotto con DenseNet è il *DenseBlock*: un blocco di layer convoluzionali che prende in input, oltre alle *feature maps* del layer precedente, anche le feature maps di tutti gli altri layer precedenti ma non adiacenti. Questo tipo di "filosofia" è il caso degenere visto anche in *ResNet* a lezione. Il vantaggio di questo tipo di applicazione è un gradiente che in fase di *backpropagation* flusice all'indietro senza problemi (senza "sparire"), visto le connessioni "identità" con cui vengono connesse le feature map di layer precedenti. Un altro vantaggio sono le prestazioni che DenseNet riesce a garantire, grazie al propagare in avanti delle feature map di più basso livello; in termini di accuratezza fa registrare un 96.54% su CIFAR-10 e un molto interessante 82.82% su CIFAR-100. DenseNet è solo una delle molteplici varianti di ResNet, tutte che sfruttano il concetto di "residuo" all'interno di un blocco dell'architettura; ove Resnet prevede un *residual block* composto da due layer convoluzionali, in DenseNet invece la dimensione di un blocco è scelta dal progettista.

Un'architettura pubblicata recentemente (marzo 2022), chiamata *RegNet*, sfrutta il concetto alla base di ResNet (il *Residuo*), andando però a sfruttare un contributo informativo tralasciato: l'informazione intermedia fra i layer convoluzionali. In che modo viene sfruttata? Nel report di Xu et al. [5] si propone di modellare l'interazione tra i due layer con un *regulator module*, un blocco che si occupa di gestire questa interazione riutilizzando il suo "stato" per le interazioni intermedie successive; grazie a ciò il successivo blocco di ResNet in input ha anche features map ottenute da tale modulo. Visto l'introduzione del concetto di "stato", non deve sorprendere se tale modulo non è altro che una rete neurale ricorrente (il modulo è implementabile sia con LSTM che con GRU). In sostanza l'architettura, ad alto livello, è possibile riorganizzarla come due reti neurali parallelizzate; una ResNet ed una rete neurale ricorrente correlata, come come mostrato in figura 1

Le prestazioni di RegNet su CIFAR riportate indicano un un error rate del 7.28% su CIFAR-10 per la versione con 20 residual block e il regulator module implementato con LSTM, e del 29.69% su CIFAR-100 per la stessa rete ma con regulator module implementato con GRU.

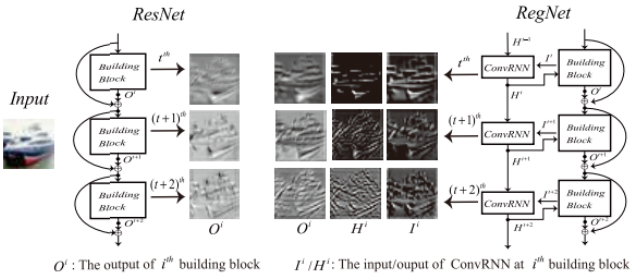

Figura 1: ResNet e RegNet a confronto. Immagine presa da [5]

III. Implementazione

L'architettura che viene proposta in questo report è dipendente dal tipo di dataset utilizzato per l'addestramento che, come ripetutamente detto, è CIFAR-10 assieme alla sua estensione CIFAR-100. CIFAR-10 contiene 60,000 immagini rgb di dimensione 32x32 pixel suddivise in 10 categorie di oggetti (aeroplani, auto, uccelli, gatti, cervi, cani, rane, cavalli, barche e camion)(un esempio in figura 2, con il risultato che per ogni classe sono presenti 6000 immagini; CIFAR-100 contiene lo stesso numero di immagini a stessa dimensione, aumentando il numero di categorie si riduce il numero di immagini per classe, che scende a 600.

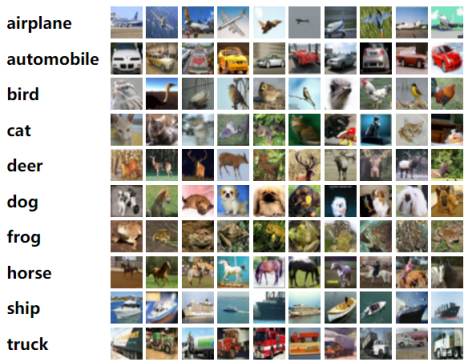

Figura 2: Esempio di immagini presenti in CIFAR-10.

L'architettura proposta verrà utilizzata su entrambi i dataset, andando solo a modificare le uscite della rete, per permettere di essere coerente con il numero di classi presenti nei dati.

La libreria utilizzata per la costruzione dell'architettura è PyTorch [6], in quanto risulta avere un ottimo compromesso

tra semplicità di utilizzo e potenza in termini di capacità espressiva.

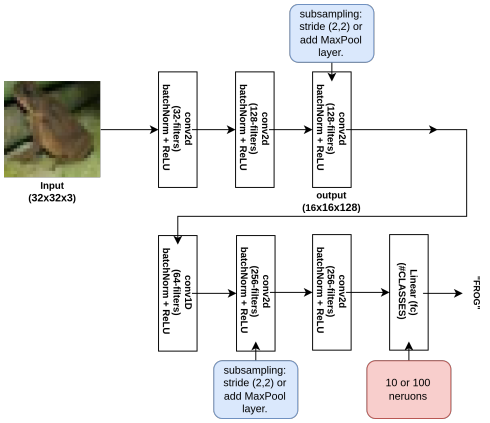

Figura 3: Struttura della rete neurale proposta.

L'architettura proposta, visivamente raffigurata in figura 3, si compone di: 6 macro blocchi "convoluzionali" che si occupano di estrarre informazioni dall'immagine data in input; un layer "lineare" che riceve in input la vettorizzazione delle features map in uscita dall'ultimo blocco convoluzionale e in uscita ha il numero di classi (per CIFAR-10 si hanno 10 neuroni di uscita e per CIFAR-100 invece 100).

Ogni macro blocco convoluzionale si compone di 3 componenti sempre presenti: il layer convoluzionale, il layer di *batch normalization* e la funzione di attivazione che è sempre ReLU. La componente dalla natura opzionale, a scelta dell'utilizzatore, è il layer di *Max Pooling*, usato per sotto campionare le features map da un blocco ad un altro; è opzionale perchè si può scegliere se effettuare quest'operazione sfruttando direttamente il layer convoluzionale, andando a modificare lo *stride* della convoluzione da (1,1) a (2,2). Successivamente vengono mostrate le differenze prestazionali fra questi due approcci. Per cercare di tenere sotto controllo il numero di parametri da ottimizzare, visto le capacità di calcolo limitate a disposizione per i test, uno dei blocchi convoluzionali utilizzati in realtà va ad effettuare non una "classica" convoluzione 2d, bensì una convoluzione 1d: la particolarità risiede nel fatto che può essere utilizzata per ridurre il numero di features map, condensando il contenuto informativo in un numero minore di mappe.

La fase di addestramento della rete è stata realizzata andando a sfruttare due metodi: Adam[7] e AdamW[8]; entrambi sono algoritmi che iterativamente adattano l'aggiornamento dei parametri ma AdamW si distingue per l'implementazione della regolarizzazione; i ricercatori asseriscono che questa differenza permette di avere un'ottimizzazione migliore per reti "classificatrici d'immagine". Così come per il sotto campionamento, vengono mostrate successivamente le eventuali differenze prestazionali.

Al fine di migliorare le capacità generalizzanti della rete, si è deciso di effettuare della *data agumentation* in fase di addestramento, su entrambi i dataset. Le trasformazioni definite vanno a cambiare in maniera aleatoria i parametri fotometrici dell'immagine (come luminosità, contrasto ed il tono) ma vanno anche a ruotare orizzontalmente e verticalmente le immagini. Nella tabella I vengono riassunti tutti i parametri utilizzati per l'addestramento della rete, dai parametri della *data gumentation* a quelli per l'ottimizzazione della rete, al netto di quelli fissi presenti all'interno dell'algoritmo di ottimizzazione:

Lr	Wd	P	Dim. Batch	luminosità	contrasto	tono
5*10^-4	10^-4	0.5	16	0.5	0.5	0.3

Tabella I: Parametri.

IV. Risultati

Sono stati condotti esperimenti su alcuni parametri della rete e componenti per l'addestramento della rete precedentemente descritta. Questi riguardano: la modalità utilizzata per sotto campionare la dimensione planare delle risposte dei filtri convoluzionali; l'algoritmo di ottimizzazione; il numero di epoche (20 o 40) usate per il training. Tutti i risultati mostrati sono il risultato di una media di addestramenti per ogni configurazione da valutare.

A. Addestramento su CIFAR-100

L'addestramento della rete su CIFAR-100, con opportuna modifica del numero di uscite della rete neurale, ha riportato i risultati presenti nelle tabelle II III suddivisi in funzione del numero di epoche di addestramento:

	MaxPooling	**stride (2,2)**
Adam	52.93%	49.25%
AdamW	53.89%	48.72%

Tabella II: Accuratezza classificazione test set CIFAR-100 su 20 epoche.

	MaxPooling	**stride (2,2)**
Adam	52.83%	50.18%
AdamW	53.21%	48.93%

Tabella III: Accuratezza classificazione test set CIFAR-100 su 40 epoche.

Inanzitutto si nota come, a parità di architettura di rete, le prestazioni in accuratezza scendano rispetto all'addestramento su CIFAR-100. Questo però è oltremodo ragionevole, visto l'aumentare della complessità di un fattore 10 del task di classificazione. É altresì vero che un classificatore aleatorio ha un'accuratezza del 1% quindi le prestazioni ottenute sono abbastanza soddisfacenti.

Osservando nel dettaglio i risultati di accuratezza media, si può decretare che le differenze di prestazioni usando due algoritmi di ottimizzazione differenti si attestano al più attorno al 2%, decretando quindi uno scarto poco significativo.

Osservando i risultati in funzione della politica di sotto campionamento si ottiene come usare il layer deterministico aggiuntivo di MaxPooling porta mediamente ad un'accuratezza sul test migliore, uno scarto di più del 3%; più evidente nel caso di addestramento su 20 epoche, meno evidente nello scenario di addestramento su 40.

A questo punto è corretto anche mostrare le criticità di questa architettura addestrata su un dataset con un'elevata complessità intrinseca: si evince infatti che le prestazioni di classificazione della rete sono pressoché identiche, a parità di

”configurazione”, al variare del numero di epoche ”utilizzate” in fase di addestramento. Questo fatto suona un po’ come un campanello d’allarme, perché significa che all’aumentare delle epoche ”la rete” non è stata più capace di generalizzare ed estrarre il contributo essenziale dall’immagine data in input. In altre parole, è un primo segnale della presenza di *overfitting*. Questa analisi è confermata dai grafici 4 e 5 (al fine di evitare un report troppo prolisso e,visto che il comportamento è analogo per tutte le configuarazioni, vengono riportati solo due grafici su otto) che evidenziano come l’accuratezza sul test venga raggiunta dopo poche epoche, mentre continua ad aumentare l’accuratezza sui dati di training. Questo fenomeno è dovuto dalla definizione di una architettura non sufficientemente profonda e che ha troppi *gradi di libertà* in funzione alla profondità della stessa.

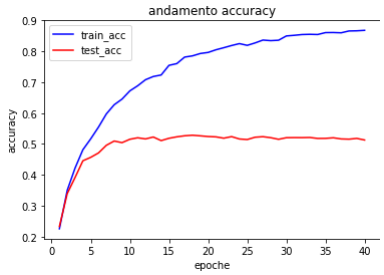

Figura 4: Andamento accuratezza su train e test (40 epoche-Adam-MaxPool).

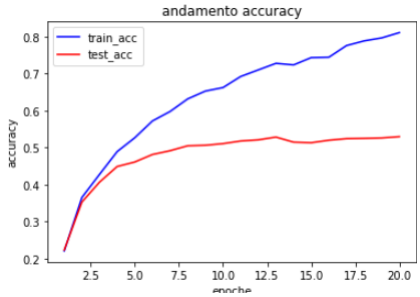

Figura 5: Andamento accuratezza su train e test (20 epoche-Adam-MaxPool).

B. *Addestramento su CIFAR-10*

L’addestramento della rete su CIFAR-10, con il numero corretto di classi di uscita dal layer terminale, ha riportato i dati suddivisi in funzione del numero di epoche usate per l’addestramento nelle tabelle IV e V:

	MaxPooling	**stride (2,2)**
Adam	83.93%	78.91%
AdamW	82.47%	80.01%

Tabella IV: Accuratezza classificazione test set CIFAR-10 su 20 epoche.

	MaxPooling	**stride (2,2)**
Adam	85.11%	79.83%
AdamW	82.21%	80.38%

Tabella V: Accuratezza classificazione test set CIFAR-10 su 40 epoche.

Rispetto alle prestazioni ottenute su CIFAR-100, su un dataset con meno classi di uscita e più immagini per singola classe, otteniamo prestazioni ben più ”soddisfacenti” a parità di architettura di rete: infatti in qualunque scenario ci si assesta sempre attorno al 80% di accuratezza nella classificazione.

Un altro aspetto da evidenziare è come la differenza di prestazioni medie ottenute utilizzando Adam e con AdamW sia sensibilmente più marcata, soprattutto nel caso di utilizzo di layer MaxPool, anche se rimane una differenza che raggiunge il massimo di 3 punti percentuali nel caso di addestramento su 40 epoche.

Continuando con l’analisi, le prestazioni sono ancora a vantaggio di un’architettura di rete che vede l’utilizzo di layer di MaxPool, rispetto ad una convoluzione con stride (2,2); nel caso di addestramento su 40 epoche si arriva ad una differenza media di quasi 6 punti percentuali, nel caso di addestramento effettuato su 40 epoche.

In termini di capacità di generalizzare, la rete ha mostrato un comportamento simile in tutte le configurazioni, per cui vengono riportati nelle figure 6 e 7 soltanto le curve di accuratezza di 2 configurazioni sulle 8 possibili.

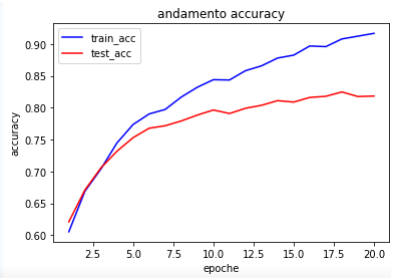

Figura 6: Andamento accuratezza su train e test (20 epoche-AdamW-stride).

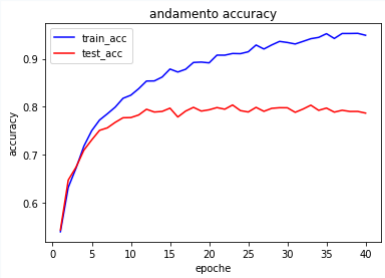

Figura 7: Andamento accuratezza su train e test (40 epoche-Adam-stride).

Anche se in una maniera meno marcata, si può notare una netta discrepanza tra le curve di addestramento e di test; infatti si nota come le 40 epoche di addestramento risultino eccessive per l’architettura proposta, in quanto gli stessi risultati sul test possono essere ottenuti con meno epoche (e quindi meno costo computazionale).

V. Conclusioni e Sviluppi Futuri

L’architettura di rete neurale proposta nonostante non sfrutti né concetti avanzati come quelli visti nella sezione II e né una struttura di rete ”profonda”, ha portato comunque a buone prestazioni in test su CIFAR-10; soddisfacenti su CIFAR-100. A risultare problematica è la presenza di un non trascurabile overfitting, in entrambi gli addestramenti ma più marcatamente problematico nel caso di addestramento su CIFAR-100. È vero

che si ottiene un'accuratezza nella classificazione "di tutto rispetto", ma la netta discrepanza della metrica tra training e test implica una poca capacità di generalizzare da parte della rete.

Un possibile scenario di aggiornamento dell'architettura può essere quella di aumentare la profondità della rete introducendo anche elementi che possano migliorare l'ottimizzazione dei parametri della stessa, come ad esempio: i blocchi residuali introdotti con ResNet o l'*inception module* di GoogLeNet. Altra soluzione possibile è sfruttare un concetto proveniente dal mondo del machine learning, ovvero l'introduzione della regolarizzazione nell'ottimizzazione dei parametri: si introduce nel problema di ottimizzazione un vincolo sulla norma del vettore dei parametri, il peso di questo contributo dipenderà da un iperparametro scelto dall'utilizzatore della rete.

RIFERIMENTI BIBLIOGRAFICI

[1] "The cifar-10 dataset." http://www.cs.toronto.edu/~kriz/cifar.html, accessed: 2022-06-17.

[2] W.-F. D. M. M. C. A. . B. Y. Goodfellow, I., "Maxout networks." *International Conference on Machine Learning*, vol. 1, no. 1, p. 1319–1327, 2013.

[3] B. Graham, "Fractional max-pooling," *ArXiv Preprint ArXiv:1412.6071*, vol. 1, no. 1, 2015.

[4] "Densely connected convolutional networks." vol. 1, no. 1, pp. 4700–4708, 2017.

[5] J. Xu, Y. Pan, X. Pan, S. Hoi, Z. Yi, and Z. Xu, "Regnet: Self-regulated network for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 1, no. 1, pp. 1–6, 2022.

[6] "Pytorch," https://pytorch.org/, accessed: 2022-06-17.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[8] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017. [Online]. Available: https://arxiv.org/abs/1711.05101