Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Progetto Finale

Classificazione di immagini su un dataset con molte classi simili tra di loro

Autori:
Lorenzo Mammana - 807391- l.mammana@campus.unimib.it
Eric Nisoli - 807147- e.nisoli1@campus.unimib.it

16 Gennaio 2020

Sommario

Il presente lavoro illustra un approccio supervisionato alla classificazione di immagini su un dataset di piccole dimensioni, ma con un grande numero di classi. In particolare viene mostrato come la combinazione di moderne tecniche di addestramento e i recenti sviluppi nel campo della ricerca di architetture per reti neurali, siano in grado di ottenere delle performance comparabili a quelle ottenuti da modelli molto più complessi presenti in letteratura.

1 Introduzione

L'obiettivo del progetto è quello di costruire un modello di machine learning per la classificazione di immagini in grado di operare bene su un dataset di piccole dimensioni comprendente un alto numero di classi simili tra di loro. Come è facilmente intuibile, il problema principale di operare su un dataset di questo tipo è l'overfitting del modello sui dati di training con conseguente fallimento del modello su dati nuovi. Per risolvere questo problema ci siamo concentrati sulla ricerca di modelli, iperparametri e tecniche di generalizzazione che permettessero al modello di essere in grado di operare su un dataset di test molto più ampio di quello utilizzato per l'addestramento. Il modello utilizzato per questo tipo di analisi è una rete neurale convoluzionale (CNN); questo tipo di modello si è dimostrato negli ultimi anni il migliore nel classificare immagini in moltissimi campi di applicazione. In particolare viene fatto uso di un modello preaddestrato su immagini simili, ci si aspetta quindi che la classificazione finale sia buona nonostante i problemi iniziali derivati dal dataset.

2 Dataset

Il dataset utilizzato è il *102 Category Flower Dataset* [1] creato dai ricercatori del *Visual Geometry Group* di *Oxford*. Il dataset è composto da 8189 immagini RGB di dimensione variabile, ogni immagine contiene uno o più fiori su sfondo neutro ed è etichettata con una singola categoria estratta da un insieme di centodue possibili categorie. Il dataset originale contiene anche i fiori segmentati dallo sfondo (Figura 1), queste immagini possono essere usate ad esempio come ulteriore input per la rete neurale. In questo lavoro non sono state utilizzate principalmente per forzare la rete ad essere più elastica

rispetto allo sfondo delle immagini. E' stata mantenuta la suddivisione del dataset definita nella pubblicazione originale, in particolare si hanno:

- 1020 immagini nel training set
- 1020 immagini nel validation set
- 6149 immagini nel test set

Ogni categoria è rappresentata da 10 immagini nel training set e nel validation set, mentre la proporzione di immagini per ogni categoria varia nel test set. La difficoltà di operare su un dataset di questo tipo è evidente, il numero di immagini è limitato mentre il test set è ampio.

Figura 1: Immagini di training e relativa segmentazione [1]

3 Approccio metodologico

Per classificare le immagini nelle classi corrette si è deciso di utilizzare una rete neurale convoluzionale. Come è noto addestrare questo tipo di modello richiede un numero di immagini dell'ordine di milioni di elementi, non avendo a disposizione così tante immagini è necessario eseguire la tecnica nota come *finetuning* in cui la CNN viene inizializzata con pesi calcolati su un dataset di milioni di immagini e modificati in accordo con i propri dati disponibili. Si è deciso quindi di utilizzare reti preaddestrate sul dataset *Imagenet* [2] contenente più di 10 milioni di immagini di cui più di trecentomila rappresentanti fiori.

3.1 Esperimento baseline

Considerando il fatto che il numero di immagini di training è limitato, è stato inizialmente deciso di definire un modello baseline non particolarmente complesso. Il modello scelto è una *Resnet18* [3] preaddestrata su *Imagenet*. Inizialmente si è provato ad utilizzare la rete come estrattore di features

per valutare quanto il preaddestramento influisca sulla classificazione finale. Le features estratte vengono classificate utilizzando un semplice percettrone multistrato:

Tabella 1: Architettura del percettrone

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 256)	6422784
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 102)	13158

L'architettura è stata scelta testando diverse configurazioni e prendendo la migliore in termini di accuratezza sul validation set.

3.1.1 Preprocessing e Data augmentation

Prima di procedere con l'estrazione delle features le immagini vengono normalizzate per avere media nulla e varianza unitaria utilizzando la formula:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{1}$$

Dove $\overline{x}$ e σ rappresentano rispettivamente la media e la varianza calcolate su *Imagenet*. Questo permette di passare alla rete delle immagini più simili a quelle su cui è stata addestrata ottenendo quindi dei risultati più attendibili.

3.1.2 Scelta degli iperparametri

Per addestrare il percettrone sulle features estratte si è deciso di utilizzare il mini-batch stochastic gradient descent (SGD) [4], con dimensione del batch pari a 64 e momentum pari a 0.9. Essendo il task una classificazione multiclasse e singola label viene utilizzata una cross-entropy categorica come funzione obiettivo da minimizzare. Il learning rate viene calcolato in accordo con l'algoritmo descritto in "Cyclical Learning Rates for Training Neural Networks" [5]. Questo algoritmo prevede che il learning rate oscilli avanti ed indietro in un range prefissato di valori ad ogni iterazione nell'addestramento, questo ha un duplice scopo:

- Permette all'ottimizzatore di uscire da eventuali punti di sella o da minimi locali che potrebbero bloccarne la corretta convergenza.

- Riduce il bias introdotto dalla scelta iniziale di un learning rate scorretto.

Per calcolare il range su cui far variare il learning rate viene utilizzata una seconda tecnica automatica descritta nello stesso articolo:

1. Si definiscono un estremo inferiore piccolo ed un estremo superiore grande su cui far variare il learning rate, ad esempio [1e-10, 1e-1].

2. Si addestra la rete per un numero ridotto di epoche partendo dall'estremo inferiore ed incrementando esponenzialmente il learning rate ad ogni batch.

3. Il training continua fino a quando il learning rate non raggiunge l'estremo superiore.

4. Viene plottato un grafico che mostra quando il learning rate è troppo basso o troppo alto.

L'algoritmo sul percettrone multistrato produce il seguente grafico:

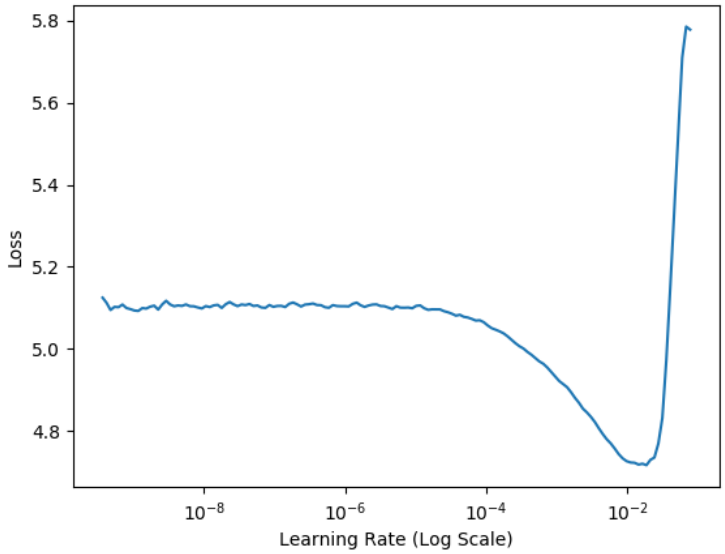


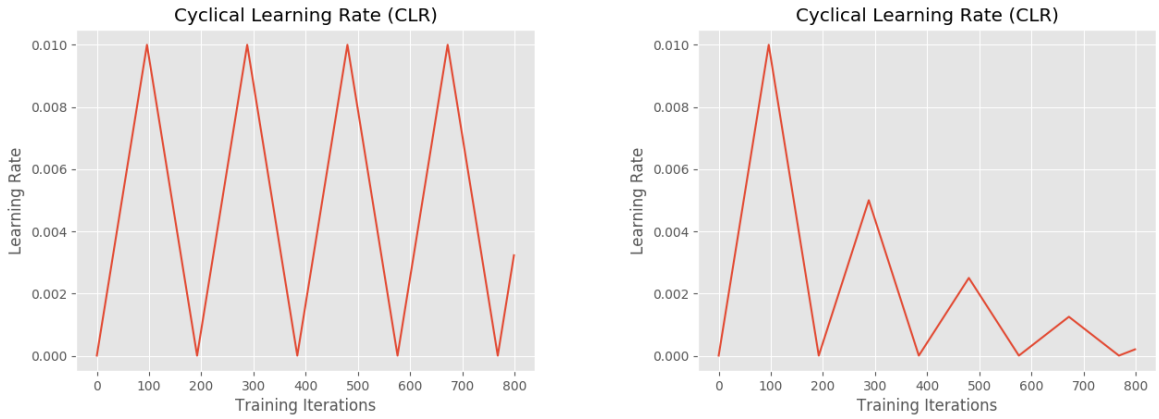
Figura 2: Output dell'algoritmo di ricerca del learning rate

Dal grafico si vede come la rete cominci ad apprendere circa a partire da 1e-5 e diverga una volta superato 1e-2, questi due valori verranno quindi utilizzati come estremi per l'algoritmo di cyclical learning rate. Questo algoritmo ha due ulteriori iperparametri, il primo è definito *Step size* ed indica il numero di iterazioni richiesto per passare dal minimo learning rate al massimo, il secondo è il metodo con cui viene modificato il learning rate. La rete viene addestrata per 50 epoche con diverse coppie di iperparametri; i risultati sono riportati in tabella.

Tabella 2: Risultati dei diversi iperparametri sul test set

Metodo	Step size	Test accuracy
Triangular	2	0.550
Triangular	4	0.550
Triangular	6	0.558
Triangular	8	0.547
Triangular2	2	0.510
Triangular2	4	0.527
Triangular2	6	0.550
Triangular2	8	0.533
Non cyclical	-	0.478

L'algoritmo non cyclical utilizza SGD con learning rate iniziale pari a 1e-2 e decay del learning rate di un fattore 0.1 in caso di non decrescita della loss sul set di validazione. E' evidente come l'algoritmo ciclico converga ad una soluzione decisamente migliore e soprattuto lo riesca a fare riducendo al massimo il tempo necessario per cercare il miglior learning rate per la rete. In tabella vengono mostrati due metodi uno definito *Triangular* e l'altro definito *Triangular2*, la differenza è che il secondo metodo dimezza l'ampiezza dell'intervallo di ricerca del learning rate ad ogni completamento del ciclo, questo è ben visibile plottando la variazione del learning rate nel tempo, come mostrato in figura 3.



(a) Triangular (b) Triangular2

Figura 3: Andamento del learning rate con step size 6

Da questa prima valutazione si è in grado di notare come il preaddestramento influisca decisamente sulla classificazione del dataset.

3.2 Esperimento 1 - Finetuning Resnet18

La sola estrazione delle feature non è ovviamente abbastanza per ottenere delle performance soddisfacenti sul test set, si è proceduto quindi ad eseguire il finetuning dell'intera rete in modo tale da aggiornare i pesi per renderli più compatibili con il nuovo dataset. Adesso invece di accodare un percettrone multistrato all'ultimo layer della rete, viene applicato un layer di *average pooling* seguito da un layer di *dropout* ($\rho = 0.5$) ed un layer completamente connesso contenente 102 neuroni.

3.2.1 Preprocessing e Data augmentation

Per massimizzare la generalizzazione del modello si è deciso di "aumentare" le immagini di training applicando i seguenti effetti:

- Flip orizzontale random
- Aumento/riduzione della luminosità ($\gamma \in [0.7, 1.3]$)
- Rotazione in un angolo di $\pm$ 10°

Il modello viene inoltre addestrato su crop random delle immagini a cui viene applicata la stessa data augmentation per verificare se ciò aumenti la generalizzazione. Le immagini vengono poi normalizzate in accordo con la media e la varianza di *Imagenet* come descritto nella sezione precedente.

3.2.2 Scelta degli iperparametri

Per addestrare la rete vengono utilizzate le stesse tecniche descritte nella sezione relativa al modello baseline. Il finetuning viene però effettuato utilizzando sia SGD che Adam [6] per verificare le eventuali differenze in termini di accuratezza. Per ognuno dei due ottimizzatori viene calcolato il range da utilizzare per l'addestramento tramite learning rate ciclico. Il modello viene addestrato per 50 epoche, ad ogni epoca vengono salvati i pesi nel caso in cui la loss sul validation set sia minore del miglior modello precedente e nel caso l'accuratezza sul validation set sia maggiore.

3.2.3 Risultati ottenuti

In tabella 6 vengono mostrati i risultati ottenuti, si vede una netta discrepanza tra i due algoritmi. La motivazione potrebbe essere relativa al fatto che, essendo la rete già addestrata, Adam sia in grado di pesare automaticamente in maniera minore i primi layer della rete rispetto agli ultimi garantendo una convergenza ottimale dell'algoritmo. Uno dei problemi riscontrati con il package *Keras*, utilizzato per addestrare il modello, è proprio l'assenza di un modo semplice per settare il moltiplicatore del learning rate per i layer iniziali, questa opzione è generalmente presente in tutti gli altri software di ML. Dalla tabella vediamo anche come il modello addestrato su crop random delle immagini non sia in grado di ottenere performance a livello del modello addestrato sulle immagini non tagliate, anche questo è facilmente giustificabile mostrando come sono fatte le immagini di training e di test.

(a) Training

(b) Test

Figura 4: Confronto immagini di training e test

Gran parte delle immagini sono su sfondo neutro, generalmente foglie, questo rende probabilmente più facile per la rete concentrarsi sul fiore ignorando lo sfondo, cosa che non succede se si utilizzando pezzi di immagine.

3.3 Esperimento 2 - Finetuning Densenet121

La Densenet-121 [7] è un tipo di architettura considerato un'evoluzione della precedente Resnet, la rete è estremamente più profonda, aumenta il rischio di overfitting sui dati, ma dovrebbe essere in grado di ottenere risultati superiori al modello addestrato precedentemente. Le medesime tecniche di data augmentation e addestramento descritte nella sezione precedente sono utilizzate per addestrare il modello. Anche in questo caso, come visibile in tabella 8 Adam ottiene risultati leggermente superiori rispetto ad SGD, vi è inoltre un netto aumento del 5% rispetto ai risultati prodotti con il modello precedente.

3.4 Esperimento 3 - Finetuning EfficientnetB4

Infine è stato testata una delle architetture più interessanti proposte in letteratura negli ultimi anni, la cosiddetta Efficientnet [8]. Questo tipo di architettura è stato costruito tramite tecniche automatiche di ricerca e riscalamento di architetture per reti neurali in modo tale da massimizzare contemporaneamente l'accuratezza della rete ed il numero di operazioni eseguite al secondo. Come si vede dal grafico, la Efficientnet è in grado di ottenere performance comparabili a modelli con dieci volte il numero di parametri da addestrare.

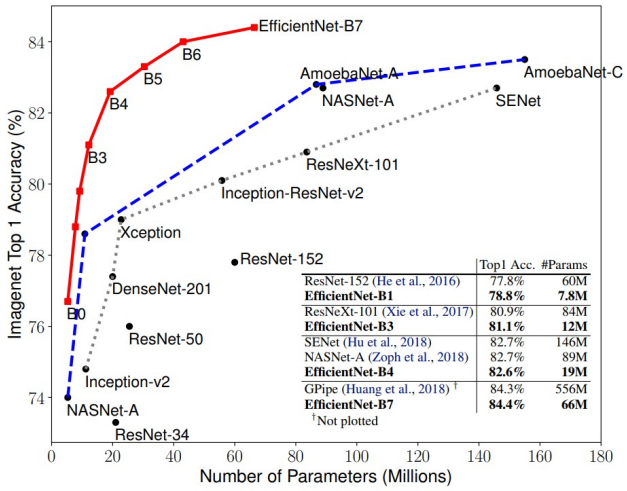


Figura 5: Confronto accuratezza rispetto al numero di parametri su imagenet

La scelta di utilizzare il modello B4 è relativa al fatto di essere il modello migliore abbastanza leggero da essere contenuto sulla GPU utilizzata.

L'input di questa rete è costituito da immagini di dimensione 380x380, il numero di immagini per batch è stato ridotto a 8 per non incappare in overflow della memoria. Utilizzando le stesse tecniche di addestramento già descritte si ottengono i risultati in tabella 10, anche in questo caso Adam funziona leggermente meglio di SGD e otteniamo un ulteriore aumento di accuratezza rispetto alla Densenet di circa il 5%.

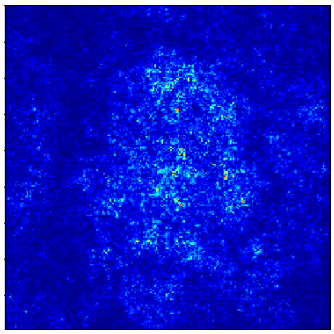
3.5 Verifica dell'apprendimento della rete

Per verificare su quali parti dell'immagine la rete fa maggiormente riferimento per eseguire la classificazione, utilizziamo due metodi ben noti in letteratura, le *Saliency Maps* [9] e il grad-CAM [10].

3.5.1 Saliency maps

L'idea delle *Saliency maps* è molto semplice, si visualizzano i gradienti di un'immagine in ingresso rispetto alla sua label calcolati sul layer di classificazione. Questa mappa dei gradienti indica quali sono le zone che più influiscono sulla classificazione corretta dell'immagine.

(a) Input



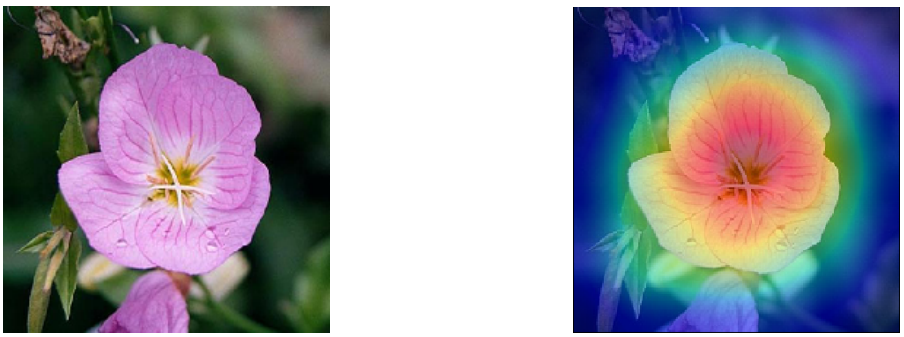
(b) Saliency map

Figura 6: Esempio di saliency map estratto dalla Desnenet121

Come si vede in figura 6 la maggior parte dei pixel che influiscono sulla classificazione è contenuta all'interno del fiore, questo è un buon indicatore della bontà del modello.

3.5.2 grad-CAM

Simile alle *saliency maps* il grad-CAM (Class Activation Map) utilizza il gradiente dell'ultimo layer convoluzionale invece che quello dell'output, questo permette di utilizzare le informazioni spaziali perse nel passaggio da convoluzionale a completamente connesso.



(a) Input (b) Saliency map

Figura 7: Esempio di grad-CAM estratto dalla Desnenet121

In questo caso, come si vede chiaramente in figura 7, è ancora più evidente come il fiore sia la componente principale nell'apprendimento della rete, si nota inoltre che l'area centrale è particolarmente discriminativa.

4 Risultati ottenuti

In questa sezione vengono riportari i risultati ottenuti sul test set quando si utilizzano i migliori modelli addestrati oltre che sul training anche sul validation set. I risultati sono mostrati in tabella 3.

Tabella 3: Risultati ottenuti

architettura	**metodo**	**step**	**lr range**	**batch size**	**crop**	**test acc**
Resnet18	Triangular	2	[1e-6,5e-4]	64	No	0.9187
Densenet121	Triangular	4	[1e-5,1e-3]	64	No	0.9455
EfficientnetB4	Triangular2	4	[1e-5,1e-3]	8	No	**0.9719**

Viene poi mostrato il confronto con gli algoritmi proposti in letteratura, alcuni di questi ([1], [11], [12]) fanno uso anche dei fiori segmentati per la classificazione.

Tabella 4: Confronto con risultati in letteratura

metodo	**test acc**
EfficientnetB4	**0.9719**
Nilsback and Zisserman [1]	0.7280
Yaqub et. al [11]	0.9710
CNN-RI-Deep [13]	0.9401
Zheng et al. [14]	0.9560
Inception-v3 [15]	0.9400
LG-CNN [12]	0.9660

5 Discussione

Gli obiettivi discussi inizialmente sono stati raggiunti, grazie alla combinazione di moderne architetture e tecniche di addestramento si è in grado di ottenere risultati comparabili a tecniche molto più sofisticate sviluppate da ricercatori esperti. Ci si aspettava dei risultati migliori per quanto riguarda l'addestramento delle reti su ritagli di immagini, questo tipo di modelli potrebbero funzionare meglio di quelli addestrati sulle intere immagini se testati su un dataset più complesso. Per ottenere un ulteriore aumento delle performance potrebbe essere possibile integrare le segmentazioni delle immagini come canale di input per la rete, questo dovrebbe migliorare sicuramente le performance sul dataset in oggetto, ma richiederebbe la conoscenza di queste immagini anche nel caso si volesse testare la rete su dati nuovi, limitandone l'utilizzo reale.

6 Conclusioni

Classificare un dataset con molte classi simili tra di loro è ormai possibile grazie agli incredibili avanzamenti in materia di ottimizzazione e architetture di reti neurali. Come mostrato in questo lavoro il semplice utilizzo di tecniche moderne permette di ottenere performance estremamente elevate, senza dover necessariamente andare ad inventare architetture o algoritmi molto più complessi. In un mondo sempre più indirizzato verso l'autoML la scelta di utilizzare tecniche di ottimizzazione e architetture automatizzate sembra la più sensata, nonostante ciò l'intervento umano richiesto è comunque ancora parecchio evidente. Come già citato in precedenza ci sono alcune tecniche testabili in futuro per aumentare ulteriormente le performance, sarebbe inoltre interessante costruire un nuovo dataset con le stesse classi, ma molto più grande e complesso, per verificare ulteriormente quali tecniche funzionino realmente meglio. Come si è visto anche una semplice Resnet18, architettura proposta cinque anni fa e non particolarmente profonda ottiene delle performance superiori al 90%. Il dataset in oggetto, proposto ormai nel lontano 2008 è quasi vicino al punto in cui una rete neurale è perfettamente in grado di classificarlo!

Ringraziamenti

Gli autori ringraziano il supporto del Consortium GARR per aver permesso l'utilizzo di una Nvidia Tesla V100 su cui sono stati eseguiti tutti gli esperimenti.

Riferimenti bibliografici

[1] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[2] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[4] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Ann. Math. Statist.*, vol. 23, no. 3, pp. 462–466, 09 1952. [Online]. Available: https://doi.org/10.1214/aoms/1177729392

[5] L. N. Smith, "Cyclical learning rates for training neural networks," 2015.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[7] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016.

[8] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019.

[9] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013.

[10] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, Oct 2019. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7

[11] H. Hiary, H. Saadeh, M. Saadeh, and M. Yaqub, "Flower classification using deep convolutional neural networks," *IET Computer Vision*, vol. 12, 04 2018.

[12] G.-S. Xie, X.-Y. Zhang, W. Yang, M. Xu, S. Yan, and C.-L. Liu, "Lg-cnn: From local parts to global discrimination for fine-grained recognition," *Pattern Recognition*, vol. 71, pp. 118 – 131, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320317302248

[13] L. Xie, J. Wang, W. Lin, B. Zhang, and Q. Tian, "Towards reversal-invariant image representation," *International Journal of Computer Vision*, vol. 123, pp. 226–250, 2016.

[14] L. Zheng, Y. Zhao, S. Wang, J. Wang, and Q. Tian, "Good practice in cnn feature transfer," 2016.

[15] Xiaoling Xia, Cui Xu, and Bing Nan, "Inception-v3 for flower classification," in *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*, June 2017, pp. 783–787.

A Risultati sperimentali

A.1 Resnet18

Tabella 5: Dettagli addestramento

ott	**range lr**	**epoche**	**batch size**	**tempo training medio**
SGD	[1e-5, 1e-3]	50	64	300s
Adam	[1e-6, 5e-4]	50	64	300s

Tabella 6: Risultati finetuning resnet18

tipo	**ott**	**crop**	**metodo**	**step**	**test acc**	**tipo**	**ott**	**crop**	**method**	**step**	**test acc**
acc	sgd	base	triangular	2	0.7627	acc	adam	base	triangular	2	0.8619
loss	sgd	base	triangular	2	0.7627	loss	adam	No	triangular	2	**0.8632**
acc	sgd	No	triangular	4	0.7528	acc	adam	No	triangular	4	0.8554
loss	sgd	No	triangular	4	0.7583	loss	adam	No	triangular	4	0.8578
acc	sgd	No	triangular	6	0.7436	acc	adam	No	triangular	6	0.8510
loss	sgd	No	triangular	6	0.7436	loss	adam	No	triangular	6	0.8537
acc	sgd	No	triangular	8	0.7604	acc	adam	No	triangular	8	0.8565
loss	sgd	No	triangular	8	0.7604	loss	adam	No	triangular	8	0.8603
acc	sgd	No	triangular2	2	0.2663	acc	adam	No	triangular2	2	0.8248
loss	sgd	No	triangular2	2	0.2663	loss	adam	No	triangular2	2	0.8246
acc	sgd	No	triangular2	4	0.5077	acc	adam	No	triangular2	4	0.8388
loss	sgd	No	triangular2	4	0.5077	loss	adam	No	triangular2	4	0.8485
acc	sgd	No	triangular2	6	0.6111	acc	adam	No	triangular2	6	0.8414
loss	sgd	No	triangular2	6	0.6111	loss	adam	No	triangular2	6	0.8497
acc	sgd	No	triangular2	8	0.6638	acc	adam	No	triangular2	8	0.8472
loss	sgd	No	triangular2	8	0.6640	loss	adam	No	triangular2	8	0.8484
acc	sgd	random	triangular	2	0.7015	acc	adam	random	triangular	2	0.8388
loss	sgd	random	triangular	2	0.7036	loss	adam	random	triangular	2	0.8412
acc	sgd	random	triangular	4	0.7127	acc	adam	random	triangular	4	0.8266
loss	sgd	random	triangular	4	0.7127	loss	adam	random	triangular	4	0.8256
acc	sgd	random	triangular	6	0.7017	acc	adam	random	triangular	6	0.8326
loss	sgd	random	triangular	6	0.7017	loss	adam	random	triangular	6	0.8376
acc	sgd	random	triangular	8	0.7150	acc	adam	random	triangular	8	0.8248
loss	sgd	random	triangular	8	0.7150	loss	adam	random	triangular	8	0.8274
acc	sgd	random	triangular2	2	0.2263	acc	adam	random	triangular2	2	0.7906
loss	sgd	random	triangular2	2	0.2267	loss	adam	random	triangular2	2	0.7906
acc	sgd	random	triangular2	4	0.4338	acc	adam	random	triangular2	4	0.8243
loss	sgd	random	triangular2	4	0.4351	loss	adam	random	triangular2	4	0.8256
acc	sgd	random	triangular2	6	0.5594	acc	adam	random	triangular2	6	0.8268
loss	sgd	random	triangular2	6	0.5607	loss	adam	random	triangular2	6	0.8272
acc	sgd	random	triangular2	8	0.6152	acc	adam	random	triangular2	8	0.8323
loss	sgd	random	triangular2	8	0.6158	loss	adam	random	triangular2	8	0.8297

A.2 Densenet121

Tabella 7: Dettagli addestramento

ott	range lr	epoche	batch size	tempo training medio
SGD	[1e-5, 1e-2]	50	64	450s
Adam	[1e-5, 1e-3]	50	64	450s

Tabella 8: Risultati finetuning Densenet121

tipo	ott	crop	metodo	step	test acc	tipo	ott	crop	metodod	step	test acc
acc	sgd	No	triangular	2	0.8934	loss	sgd	random	triangular2	8	0.8866
loss	sgd	No	triangular	2	0.8955	acc	adam	No	triangular	2	0.9094
acc	sgd	No	triangular	4	0.8910	loss	adam	No	triangular	2	0.9094
loss	sgd	No	triangular	4	0.8939	acc	adam	No	triangular	4	**0.9108**
acc	sgd	No	triangular	6	0.9009	loss	adam	No	triangular	4	0.9103
loss	sgd	No	triangular	6	0.9019	acc	adam	No	triangular	6	0.9056
acc	sgd	No	triangular	8	0.8981	loss	adam	No	triangular	6	0.9081
loss	sgd	No	triangular	8	0.9020	acc	adam	No	triangular	8	0.9063
acc	sgd	No	triangular2	2	0.8765	loss	adam	No	triangular	8	0.9063
loss	sgd	No	triangular2	2	0.8733	acc	adam	No	triangular2	2	0.8973
acc	sgd	No	triangular2	4	0.8885	loss	adam	No	triangular2	2	0.8975
loss	sgd	No	triangular2	4	0.8884	acc	adam	No	triangular2	4	0.9073
acc	sgd	No	triangular2	6	0.8957	loss	adam	No	triangular2	4	0.9074
loss	sgd	No	triangular2	6	0.8786	acc	adam	No	triangular2	6	0.9076
acc	sgd	No	triangular2	8	0.8965	loss	adam	No	triangular2	6	0.9097
loss	sgd	No	triangular2	8	0.8967	acc	adam	No	triangular2	8	0.9029
acc	sgd	random	triangular	2	0.8816	loss	adam	No	triangular2	8	0.9056
loss	sgd	random	triangular	2	0.8827	acc	adam	random	triangular	2	0.8921
acc	sgd	random	triangular	4	0.8874	loss	adam	random	triangular	2	0.8918
loss	sgd	random	triangular	4	0.8874	acc	adam	random	triangular	4	0.9030
acc	sgd	random	triangular	6	0.8835	loss	adam	random	triangular	4	0.9030
loss	sgd	random	triangular	6	0.8851	acc	adam	random	triangular	6	0.8986
acc	sgd	random	triangular	8	0.8816	loss	adam	random	triangular	6	0.8986
loss	sgd	random	triangular	8	0.8876	acc	adam	random	triangular	8	0.8928
acc	sgd	random	triangular2	2	0.8474	loss	adam	random	triangular	8	0.8951
loss	sgd	random	triangular2	2	0.8474	acc	adam	random	triangular2	2	0.8804
acc	sgd	random	triangular2	4	0.8770	loss	adam	random	triangular2	2	0.8817
loss	sgd	random	triangular2	4	0.8780	acc	adam	random	triangular2	4	0.8881
acc	sgd	random	triangular2	6	0.8833	loss	adam	random	triangular2	4	0.8879
loss	sgd	random	triangular2	6	0.8858	acc	adam	random	triangular2	6	0.8990
acc	sgd	random	triangular2	8	0.8814	loss	adam	random	triangular2	6	0.8991
loss	sgd	random	triangular2	8	0.8837	acc	adam	random	triangular2	8	0.8894

A.3 EfficientnetB4

Tabella 9: Dettagli addestramento

ott	range lr	epoche	batch size	tempo training medio
SGD	[1e-5, 1e-3]	50	8	2000s
Adam	[1e-5, 1e-3]	50	68	2000s

Tabella 10: Risultati finetuning EfficientnetB4

tipo	ott	crop	metodo	step	test acc	tipo	ott	crop	metodo	step	test acc
acc	sgd	No	triangular	2	0.9198	acc	adam	No	triangular	2	0.9105
loss	sgd	No	triangular	2	0.9198	loss	adam	No	triangular	2	0.9105
acc	sgd	No	triangular	4	0.9173	acc	adam	No	triangular	4	0.9386
loss	sgd	No	triangular	4	0.9173	loss	adam	No	triangular	4	0.9386
acc	sgd	No	triangular	6	0.9164	acc	adam	No	triangular	6	0.9261
loss	sgd	No	triangular	6	0.9183	loss	adam	No	triangular	6	0.9261
acc	sgd	No	triangular	8	0.9164	acc	adam	No	triangular	8	0.9302
loss	sgd	No	triangular	8	0.9164	loss	adam	No	triangular	8	0.9329
acc	sgd	No	triangular2	2	0.5316	acc	adam	No	triangular2	2	0.9492
loss	sgd	No	triangular2	2	0.5394	loss	adam	No	triangular2	2	0.9490
acc	sgd	No	triangular2	4	0.7371	acc	adam	No	triangular2	4	**0.9567**
loss	sgd	No	triangular2	4	0.7375	loss	adam	No	triangular2	4	0.9559
acc	sgd	No	triangular2	6	0.8211	acc	adam	No	triangular2	6	0.9411
loss	sgd	No	triangular2	6	0.8250	loss	adam	No	triangular2	6	0.9447
acc	sgd	No	triangular2	8	0.8692	acc	adam	No	triangular2	8	0.9393
loss	sgd	No	triangular2	8	0.8697	loss	adam	No	triangular2	8	0.9393
acc	sgd	random	triangular	2	0.9092	acc	adam	random	triangular	2	0.8923
loss	sgd	random	triangular	2	0.9092	loss	adam	random	triangular	2	0.8923
acc	sgd	random	triangular	4	0.9116	acc	adam	random	triangular	4	0.9141
loss	sgd	random	triangular	4	0.9116	loss	adam	random	triangular	4	0.9141
acc	sgd	random	triangular	6	0.9079	acc	adam	random	triangular	6	0.9128
loss	sgd	random	triangular	6	0.9121	loss	adam	random	triangular	6	0.9128
acc	sgd	random	triangular	8	0.9094	acc	adam	random	triangular	8	0.9168
loss	sgd	random	triangular	8	0.9110	loss	adam	random	triangular	8	0.9208
acc	sgd	random	triangular2	2	0.4844	acc	adam	random	triangular2	2	0.9386
loss	sgd	random	triangular2	2	0.4815	loss	adam	random	triangular2	2	0.9395
acc	sgd	random	triangular2	4	0.7119	acc	adam	random	triangular2	4	0.9355
loss	sgd	random	triangular2	4	0.7118	loss	adam	random	triangular2	4	0.9370
acc	sgd	random	triangular2	6	0.7978	acc	adam	random	triangular2	6	0.9305
loss	sgd	random	triangular2	6	0.7978	loss	adam	random	triangular2	6	0.9331
acc	sgd	random	triangular2	8	0.8492	acc	adam	random	triangular2	8	0.9284
loss	sgd	random	triangular2	8	0.8502	loss	adam	random	triangular2	8	0.9284