Reusing Pretrained Network for Food Recognition

Davide Massuda

Giugno 2021

1 Abstract

Al giorno d'oggi l'uso sempre più intenso di reti neurali per l'apprendimento automatico ha portato l'uomo verso la possibilità di creare modelli per predizioni automatiche sempre più precisi. Nel corso degli ultimi venti anni infatti sono nate numerose architetture ognuna con dei propri punti di forza atte a superare le debolezze delle "generazioni" precedenti. Grazie a questo sviluppo, è stato possibile utilizzare il Machine Learning (ML) nei campi più variegati; dal riconoscimento automatico degli oggetti, alla guida autonoma per una macchina, dalla previsione di un prestito in ambito bancario, alla previsione di malattie in campo medico e molto altro ancora. In quest'articolo, ci occuperemo di come impiegare ML per riconoscere in modo automatico varie categorie di cibo. In particolare attraverso l'uso di varie reti Convolutional Neural Network (CNN), utilizzando come dataset, Food-101 [BGV14], che raccoglie 101 categorie, ognuna delle quali avente 1000 immagini uniche.

2 Introduzione

Nonostante la tecnologia abbia fatto passi da giganti negli ultimi venti anni, c'è un campo in cui questa rimane ancora indietro: quello culinario. Il cibo risulta essere un tassello essenziale nella vita di tutti i giorni e nonostante questo, c'è ancora molta ignoranza rispetto a quali cibi possano essere considerati sani e quali invece dannosi [Knu]. In questo articolo si cercherà di costruire un sistema di Food Recognition che attraverso un processamento dei dati in input, riesca a classificare il cibo presente nell'immagine in una delle 101 categorie a disposizione. Verranno addestrate a tal fine due architetture, una ResNet e una DenseNet, attraverso la tecnica del *Transfer learning from pre-trained models*, in quanto entrambe utilizzate per il riconoscimento delle immagini e, implementate attraverso l'utilizzo di due librerie diverse: Keras [Ker] per ResNet e PyTorch [PyT] per DenseNet. Questo può essere considerato un primo passo verso la creazione di modelli che riescano in un futuro a riconoscere pietanze ipocaloriche o ipercaloriche, *healthy* o *unhealthy* o addirittura cibi con particolari caratteristiche o particolari ingredienti dannosi o necessari per il nostro corpo, così da favorire un'alimentazione più equilibrata.

2.1 Il punto di partenza

Per la stesura di questo elaborato e la creazione dei modelli, il punto di partenza è stato l'articolo di Stanford *Deep Learning Based Food Recognition* di Qian Yu et al [Don]. in cui grazie alla creazione di una CNN ResNet e un pre-processamento delle immagini in input (bilanciamento dei colori), sono stati in grado di raggiungere una Top-1 Accuracy del 72.55% ed una Top-5 Accuracy del 91.31%. Si cercherà quindi di ricreare una rete che permetta di raggiungere prestazioni simili, inizialmente attraverso l'uso di una ResNet come suggerito nell'articolo di Stenford e poi attraverso l'utilizzo di una DenseNet. Per la prima rete si utilizzerà la libreria Keras, mentre la seconda invece verrà implementata attraverso la libreria PyTorch.

2.2 Food-101 dataset

Il dataset utilizzato per la creazione del modello è un dataset presente online che riposta 101 categorie di cibi ognuna avente 1000 immagini. La dimensione totale di questo è di circa 5GB e possiede in totale 101000 immagini. Per ogni categoria sono presenti immagini univoche che fotografano piatti di cibi differenti.

Figure 1: food-101

3 Modelli a confronto

i modelli usati per la creazione di questo elaborato sono modelli pre-addestrati per il riconoscimento delle immagini. In particolare si utilizzeranno reti addestrate attraverso l'utilizzo di un dataset chiamato *ImageNet* [Ima]. Si cercherà quindi di confrontare l'andamento delle reti prima riutilizzando tutti i parametri della rete e valutando le performance (Fixed weights), poi provando a "scongelare" gli ultimi layers in modo da riaddestrarli sullo specifico task, valutando eventuali miglioramenti.

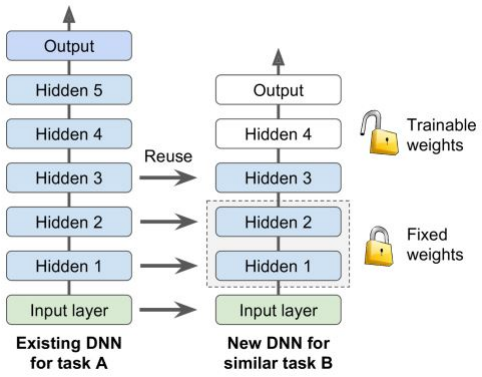

Figure 2: Reusing pretrained layers [Gér19]

Per quanto riguarda gli *Optimizer* utilizzati, si farà uso di RMSProp e Adam, poichè consigliati nell'articolo di Stenford, in quanto il primo tende a convergere più rapidamente nelle fasi iniziali, mentre il secondo risulta essere più accurato e stabile .

3.1 Convolutional Neural Network

Le architetture Convolutional Neural Network (CNN) sono architetture sviluppate intorno agli anni 80 ispirate a degli studi della zona della corteccia deputata al riconoscimento visivo. Hanno avuto grande impatto grazie allo sviluppo e utilizzo delle GPU che hanno permesso il processamento dei dati in modo più rapido. Queste tecniche si basano sull'utilizzo di più Layer sequenziali che prendono in input features estratte dal layer precedente, ed unite insieme, formano features sempre più complesse.

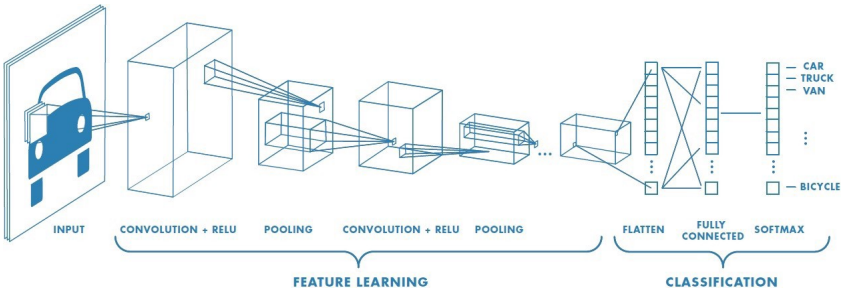

Figure 3: Convolutional Neural Network [CNN]

3.1.1 ResNet

Residual Network o come viene comunemente definita, Resnet, è una rete neurale convoluzionale sviluppata da *Kaiming He et al.* che vinse nel 2015 il ILSVRC. Questa architettura costituita da 152 livelli, utilizza skip connection per evitare problemi come exploding gradient o vanishing gradient. Risulta es-

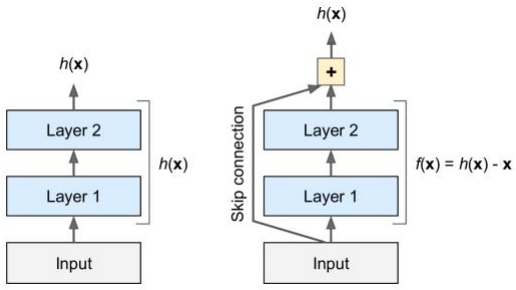

Figure 4: Resnet [Gér19]

sere un architettura piuttosto semplice, simile negli strati iniziali ad una GoogleLeNet, ma senza dropout layer e con uno strato deep composto da semplici *residual units* nel mezzo. Ognuna di queste è composta da due strati convoluzionali, con Batch Normalization e ReLu activation, attraverso l'utilizzo di kernel 3x3 con stride 1 e padding SAME.

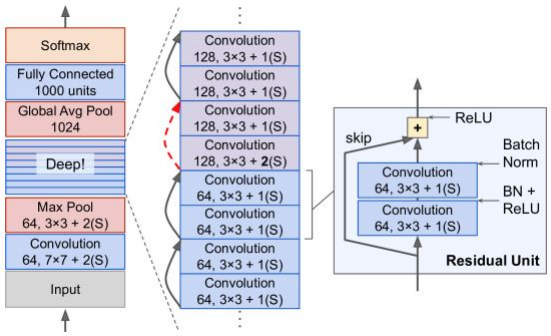

Figure 5: Resnet Architecture [Gér19]

3.1.2 DenseNet

Densely Connected CNN o DenseNet [Res] è un'architettura proposta da *Huang et al.* nel 2016. Come per la ResNet, anche la DenseNet sfrutta le *shortcut connections*, ma in questo caso, i tutti i layer sono collegati l'un l'altro. In questa architettura, l'input di ogni livello è costituito dalle *features maps* dei livelli precedenti e il suo output viene passato a ogni livello successivo. Oltre

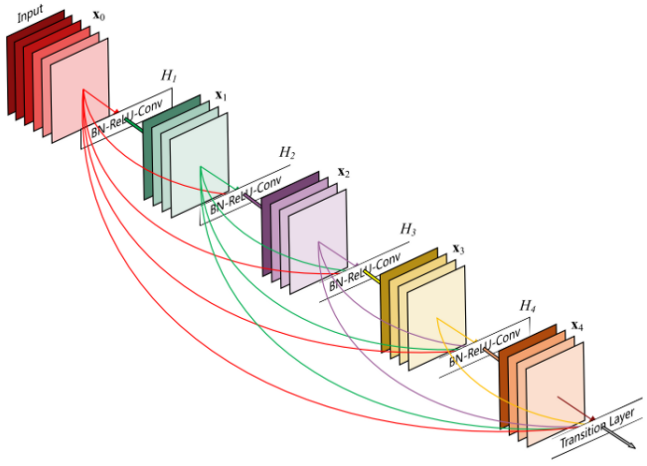

Figure 6: DenseNet [Gér19]

ad affrontare il problema del *vanishing problem*, quest'architettura incoraggia il riutilizzo delle features, rendendo la rete altamente efficiente in termini di parametri. Inoltre i layer intermedi chiamati *Transition Layers* permettono di applicare un *downsampling* attraverso l'utilizzo di una batch normalization, un layer convoluzionale 1x1 e un *pooling layer* 2x2.

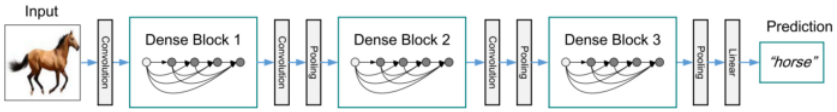

Figure 7: DenseNet Architecture

3.2 Optimizer

3.2.1 RMSProp

RMSProp (Root Mean Square Propagation) è un metodo con tasso di apprendimento adattivo che si prefissa come obiettivo il superamento del problema della cancellazione del tasso di apprendimento presente in AdaGrad. Infatti AdaGrad può rallentare la discesa del gradiente interrompendo anticipatamente il training. RMSProp risulta essere quindi una variazione di AdaGrad in cui il vettore accumulatore considera maggiormente gli ultimi gradienti calcolati. A tal fine viene introdotto un fattore di decay β che nonostante rappresenti un iperparametro compreso tra 0 ed 1, risulta lavorare bene per valori intorno allo 0.9.

$$\mathbf{s} \leftarrow \beta \mathbf{s} + (1 - \beta) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s} + \epsilon}$$

Figure 8: RMSProp [Gér19]

3.2.2 Adam

Adam (adaptive moment estimation) è un optimizer che cerca di combinare insieme l'idea di Momenum optimization e RMSProp. Come per Momentum optimization tiene traccia della media dei decadimenti esponenziali dei gradienti passati, ma in più grazie all'introduzione di RMSProp, tiene traccia del quadrato della media dei decadimenti esponenziali dei gradienti. Guardando

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
3. $\widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1 - \beta_1^{\,t}}$
4. $\widehat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - \beta_2^{\,t}}$
5. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \epsilon}$

Figure 9: Adam [Gér19]

alle formule 1, 2 e 5 si noterà che queste sono molto simili a quelle del Momentum optimization e RMSProp. L'unica differenza è che lo step 1 computa una media del decadimento esponenziale invece che una somma. Le formule 3 e 4 invece riguardano aspetti tecnici; infatti m e s sono inizialmente impostati a 0 e queste due formule servono ad incrementare i loro valori all'inizio del training. Per quanto riguarda invece gli iperparametri, solitamente β_1 è inizializzato a 0.9, mentre β_2 a 0.999. Poichè Adam è un algoritmo di learning adattivo, il valore iniziale di η può essere impostato ad un valore tipico di 0.001 senza ulteriore tuning.

4 Creazione del Modello

4.1 Specifiche Hardware

Per la creazione dei modelli si è utilizzato un computer ASUS VivoBook S15 con 16GB di RAM, processore Intel Core i7-8565U e scheda grafica GeForce MX150. Quest'ultima però non possedeva abbastanza RAM da permettere di

eseguire i calcoli e quindi i modelli sono stati addestrati tramite l'utilizzo della CPU.

4.2 Divisione dei dati

Il dataset di partenza è stato diviso in tre parti, un train set, un validation set e un test set. Ovviamente il modello è stato addestrato sul primo insieme e per ogni epoca, si è utlizzato il secondo per la verifica ed infine, l'accuracy finale è stata calcolata attraverso l'utilizzo del terzo. Questo è stato possibile grazie all'enorme quantità di immagini presenti, 1000 per ogni categoria. Si è deciso quindi di dividere i dati secondo la seguente percentuale: 60% per il train set, 20% per il test set e 20% per il validation set.

4.3 Pre-processamento delle immagini

Per quanto riguarda il pre-processamento dei dati, alcune volte le immagini risultavano essere troppo scure o avevano uno sfondo poco distinguibile dal soggetto. Avevano quindi problemi di colore, saturazione, luminanza e così via. Quindi come suggerito nell'articolo di Stenford si sono attuate due modifiche: la prima un *Grey World method* per un bilanciamento dei bianchi per far in modo che i valori RGB siano tutti simili ad un valore grigio ottenuto come media di questi; la seconda invece *Histogram equalization*, per aumentare contrasti e luminanza.

Figure 10: Processamento immagini

Nella figura sopra riportata viene mostrata inizialmente l'immagine originale, poi la stessa processata attraverso *Grey World method*, ed infine, l'ultima mostra il processamento finale, quello ottenuto dalla seconda con l'applicazione di *Histogram equalization*.

4.4 Esperimenti Vari

Si sono voluti creare più modelli ognuno con differenti caratteristiche, inizialmente testando le immagini senza pre-processamento, attraverso una rete ResNet e una DenseNet, lasciando bloccati (Fixed Wights) i layers, poi modificando i pesi degli ultimi; infine le stesse operazioni sono state eseguite con le immagini pre-processate. Il tutto è stato fatto utilizzando soltanto 10 epoche per modello,

ma i risultati possono considerarsi utili per un elaborato che tenga conto di più epoche.
Come primo esperimento si propone un modello addestrato tramite ottimizatore Adam, con rete ResNet, attraverso 10 epoche. I risultati ottenuti risultano essere non molto soddisfacenti, infatti, la top-1 accuracy di attesta intorno 53%, mentre per quanto riguarda la top-5, l'accuracy è del 78%.
È stato quindi rieseguito il modello questa volta con gli ultimi layers "sconge-lati" ottenendo così prestazioni molto più elevate. Di seguito viene riportata l'immagine delle ultime 5 epoche di addestramento. Si è poi provato, come de-

```
Epoch 1/5
1894/1894 [==============================] - 12687s 7s/step - loss: 1.9790 - accuracy: 0.4997 - val_loss: 1.6026 -
val_accuracy: 0.5947
Epoch 2/5
1894/1894 [==============================] - 12669s 7s/step - loss: 1.3658 - accuracy: 0.6363 - val_loss: 1.4204 -
val_accuracy: 0.6311
Epoch 3/5
1894/1894 [==============================] - 12639s 7s/step - loss: 1.0543 - accuracy: 0.7101 - val_loss: 1.3642 -
val_accuracy: 0.6558
Epoch 4/5
1894/1894 [==============================] - 12791s 7s/step - loss: 0.8073 - accuracy: 0.7693 - val_loss: 1.4104 -
val_accuracy: 0.6571
Epoch 5/5
1894/1894 [==============================] - 12673s 7s/step - loss: 0.5767 - accuracy: 0.8291 - val_loss: 1.3776 -
val_accuracy: 0.6750
632/632 [==============================] - 2347s 4s/step - loss: 1.3873 - accuracy: 0.6730
```

Figure 11: ResNet con Adam Optimizer

scritto nell'articolo di Stenford, ad utilizzare RmsProp come ottimizzatore nelle prime epoche e poi Adam per le altre, ma i risultati ottenuti non risultano essere molto diversi da quelli precedenti. Le stesse operazioni sono state eseguite per rete DenseNet ottenendo prestazioni simili e un overfitting all'aumentare delle epoche.
Utilizzando invece modelli che prendevano in input immagini pre-processate come descritto nel paragrafo *Pre-processamento delle immagini*, i risultati risultavano essere simili in termini di prestazioni, ma con accuracy sempre crescenti. Nonostante l'accuracy top-1 non sia aumentata, quella top-5 invece si attestava intorno ad un 84% contro un 78% ottenuto precedentemente.
Di seguito viene riportata una tabella contenente i risultati dei vari addestramenti su 10 epoche.

5 Conclusioni

In questo elaborato si sono mostrati vari modi e metodi per addestrare una rete neurale, sia tenendo conto dei parametri in input, sia delle immagini che la rete avrebbe processato. Nonostante le epoche prese in esame risultino essere poche per un addestramento ottimale, queste sono un primo passo verso la creazione di un modello che riesca a predire in modo accurato la categoria di cibo alla quale l'immagine appartiene. Anche se le immagini presenti nel dataset possono sembrare tante, vi sono presenti anche foto che purtroppo risultano essere sfogate, inquadrano altro rispetto al cibo o risultano essere forvianti.

Method	Top-1 Accuracy	Top-5 Accuracy
ResNet (Full Layer Fixed Weights)	52.97%	78.04%
ResNet (RMSProp + Adam)	66.54%	75.94%
ResNet (Adam)	67.30%	74.78%
ResNet(Preprocessed Images, RMSProp + Adam)	65.03%	84.32%
ResNet(Preprocessed Images, Adam)	66.27%	84.37%
DenseNet (RMSProp + Adam)	61.23%	-
DenseNet(Preprocessed Images, RMSProp + Adam)	65.42%	-

Figure 12: Top Accuracy dei modelli

References

[BGV14] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. "Food-101 – Mining Discriminative Components with Random Forests". In: *European Conference on Computer Vision*. 2014.

[Gér19] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., 2019. ISBN: 978-1492032649. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.

[CNN] CNN. "A Comprehensive Guide to Convolutional Neural Networks". In: URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[Don] Jingfan Wang Dongyuan Mao Qian Yu. "Deep Learning Based Food Recognition". In: URL: http://cs229.stanford.edu.

[Ima] ImageNet. "ImageNet". In: URL: https://www.image-net.org/index.php.

[Ker] Keras. "Keras". In: URL: https://keras.io/.

[Knu] Donald Knuth. *Exploiting Food Choice Biases for Healthier Recipe Recommendation*. URL: https://www.christophtrattner.info/pubs/SIGIR2017.pdf.

[PyT] PyTorch. "PyTorch". In: URL: https://pytorch.org/.

[Res] ResNet. "An Overview of ResNet and its Variants". In: URL: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.