Image classification di razze di cani con tecniche di Deep Learning

Di Cosmo Giuseppe
0258587

Nedia Salvatore
0264925

***Abstract*—Nell'ambito del corso di Algoritmi per il Web, anno accademico 2018-2019, è stato realizzato il progetto, il cui scopo è quello di creare un classificatore in grado di determinare la razza di un cane da una foto, utilizzando il framework di deep learning Google TensorFlow.**

I. Introduzione

Le CNN forniscono un'architettura ottimale per il riconoscimento di immagini e la rilevazione dei pattern, oltre al fatto che il loro utilizzo nel deep learning è diventato sempre più comune grazie a tre fattori importanti:

- elimininazione della necessità dell'estrazione manuale delle feature poiché queste vengono apprese direttamente dalla CNN.
- produzione di risultati di riconoscimento all'avanguardia.
- possibilità di essere ri-addestrate per nuove attività di riconoscimento, consentendo di sfruttare reti pre-esistenti

Nel nostro caso l'obiettivo era quello di utilizzarle per identificare la razza del cane presente nell'immagine. Per la realizzazione di tale task è stata definita un'architettura ad hoc e sono state sfruttate diverse tecniche che sono presentate nel seguito.

II. Architettura della rete

La rete adottata è stata quella in figura 1, in cui viene adottata una rete NasNetLarge [1], a cui segue uno strato di GlobalMaxPooling2D,Flatten e la rete Dense con un numero di neuroni pari al numero di razze da classificare.

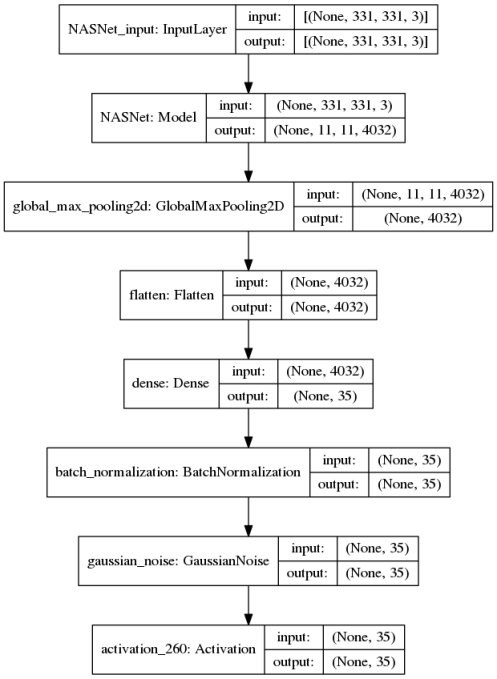

Figure 1. Architettura della rete

III. Preparazione del dataset

L'operazione preliminare è stata quella di suddividere il dataset in training set e validation set con una politica 80/20, seguita da un operazione di shuffle del dataset per evitare situazioni sfavorevoli, come tutti gli esempi di una classe in un determinato set.
Si è passato poi a definire la pipeline di esecuzione. Il pipelining si sovrappone alla pre-elaborazione e all'esecuzione del modello di una fase di addestramento. Mentre la GPU sta eseguendo il passo di addestramento N, la CPU permette di preparare i dati per il passo N + 1. In questo modo si riduce il tempo necessario per estrarre e trasformare i dati. E' stata quindi effettuata una fase di preprocessing in cui si è eseguita la conversione one hot delle etichette, la normalizzazione dei dati ed il resizing delle immagini. Per farlo sono stati provati diversi valori:

- (195,195)
- (350,350)
- (370,370)
- (400,400)
- (331,331)

Il passaggio da (195,195) a (350,350) è stato fondamentale, poichè ha permesso di passare da un'accuracy di 0.8328 a 0.93. Questo è dovuto dal fatto che la grandezza fissata rendeva le immagini troppo piccole per poter ottenere dettagli rilevanti per la classificazione. Si sono scartati valori maggiori di 350, dati i peggioramenti ottenuti con 370 e 400, per poi passare a (331,331), valori richiesti per la rete NASNetLarge.

Sono stati definiti successivamente i valori per la batch size e il numero di epoche(fissato a 150 ma l'early stopping permette di fermarsi intorno alla venticinquesima epoca). Per quanto riguarda la dimensione dei batch si è notato che questo risulta essere uno degli iperparametri che più influenzano le prestazioni della rete. Infatti è dipendente molto dal tipo di rete utilizzata per il pretraining, ad esempio con una

Xception si avevano buone prestazioni, intorno al 93% con una batch size abbastanza elevata(180), ma nel momento dell'adozione della NasNet, batch size più piccole hanno permesso di ottenere un valore di accuracy maggiore. Si è adottata una batch size pari a 64,notando che valori più alti o più bassi degradavano le prestazioni. Ad esempio utilizzando una dimensione di 96, si è ottenuto un degrado del 2/3%.

IV. Data augmentation

Il metodo migliore di stimare un modello in grado di offrire buone performance, in termini di generalizzazione, è senza dubbio quello di addestrarlo su un dataset il più ampio possibile; purtroppo però nel nostro caso la quantità di dati disponibili non è molto elevata(2205 esempi). Un modo per aggirare questo problema è quello di generare delle nuove immagini da aggiungere al training set, a partire da versioni leggermente modificate dei dati originali. Questa tecnica è particolarmente efficace in problemi di classificazione di immagini, in quanto quest'ultime sono influenzate dalla presenza di una vasta gamma di fattori di variazione, molti dei quali possono essere facilmente simulati.
Le tecniche di data augmentation adottate sono state:

- *random_flip_left_right*, in modo da ottenere immagini dove il cane venga spostato da destra a sinistra o viceversa
- *random_saturation*, con fattore di saturazione minimo pari a 0.8 e massimo pari a 1.8
- *random_brightness*, con fattore di illuminazione pari a 0.08
- *random_constrast*, con fattore di contrasto da un minimo di 0.8 a un massimo di 1.5

E' stata utilizzata, per un breve periodo, anche la tecnica *random_jpeg_quality*, per aggiungere immagini con una qualità più bassa o più alta. Tale tecnica è stata però scartata poichè produceva un peggioramento dell'ordine del 3/5% sull'accuracy del modello.

V. Transfer Learning

La tecnica che ha permesso di ottenere un incremento notevole dell'accuracy è stata sicuramente il transfer learning. Attraverso questa tecnica è possibile ottenere i pesi di una rete addestrata con *ImageNet*, in modo da poterli riutilizzare per l'addestramento della nostra rete. Questo permette di sfruttare il training effettuato su un dataset molto più grande di quello a disposizione. Avendo una dimensione del dataset abbastanza piccola, si è effettuato il freezing di tutti i layer eccetto l'ultimo. L'adozione di questa tecnica, per la prima volta, ha permesso di passare da un accuracy iniziale del 10% ad una del 51% con una rete Xception, confermando quindi le potenzialità elencate precedentemente. Tra le varie reti a disposizione si è utilizzata la NASNetLarge perchè ha permesso di ottenere l'accuracy maggiore. Ne sono state provate anche altre, che seguono in ordine di risultati migliori, con percentuali di peggioramento rispetto quella scelta :

1) InceptionResNet(-3%)
2) InceptionV2(-2%)
3) Xception(-5%)
4) DenseNet201(-10%)

VI. Tecniche di regolarizzazione

Uno dei principali problemi da affrontare è quello dell'overfitting. Le reti neurali, durante il loro processo di apprendimento sfruttano la funzione di costo per decidere come aggiornare i propri parametri, ovvero pesi e bias. In letteratura esistono numerose tecniche di regolarizzazione, tra le più utilizzate ci sono L1, L2 e L1L2 regularization, dropout, batch normalization, early stopping e gaussian noise.

A. L2 Regularization

La regolarizzazione L2 consiste nell'inserire un termine aggiuntivo alla funzione costo, penalizzante per alcune configurazioni. Si è utilizzata la regolarizzazione L2, rispetto alla L1 e la L1L2, poichè ha permesso di ottenere valori migliori e inoltre non era impattata da un aumento della loss(+12%) causata dalle altre due configurazioni. Inoltre il parametro di regolarizzazione è stato settato a 0.01, essendo il valore che ha permesso di ottenere le prestazioni migliori.

B. Batch Normalization

La batch normalization è una tecnica per l'addestramento di reti deep che permette di standardizzare gli input in uno strato per ciascun mini-batch. Ciò ha l'effetto di stabilizzare il processo di apprendimento e ridurre drasticamente il numero di epoche. Inoltre, aiuta a rimuovere rumore dai dati sottraendo la media, in modo da avere un effetto di regolarizzazione, questo è utile poichè una delle maggiori cause dell'overfitting si ha quando il modello impara diversi parametri "rumorosi". Nel nostro caso è stato adottato un solo layer, inserito dopo uno strato dense, permettendo di raggiungere prestazioni più alte rispetto al caso in cui questo non veniva usato.

C. Dropout

Il dropout è una tecnica in cui vengono selezionati neuroni in modo casuale che vengono ignorati durante l'addestramento. Ciò significa che il loro contributo all'attivazione dei neuroni a valle viene temporaneamente rimosso sul passo di forward e eventuali aggiornamenti di peso non vengono applicati al neurone sul passo di backward. Non vi è stata la necessità dell'uso di tale tecnica poiché si è notato che l'aggiunta di questo layer, rispetto a L2 regularization, degradava le prestazioni della rete. Si è puntato quindi all'uso della regolarizzazione classica. Questa scelta è stata dettata anche dal fatto che il dropout insieme a batch normalization può portare a un calo delle performance, come spiegato in [2].

D. Early stopping

Attraverso la tecnica dell'early stopping si interrompe prematuramente la minimizzazione della funzione di errore. All'inizio l'errore sul validation set diminuisce al migliorare dell'errore sul training,quindi se si ha il fenomeno dell'overfitting, l'errore sul validation set sale. A questo punto si interrompe il processo di addestramento. Tale tecnica è stata quindi adottata nell'ambito del progetto.

E. Gaussian Noise

Il rumore gaussiano è utile a mitigate overfitting,agendo quindi come tecnica di regolarizzazione. Inoltre può anche essere visto come una forma di data augmentation randomica. Nell'architettura è stato adottato questo layer con rumore gaussiano pari a 0.5, dato che valori minori o peggiori di questo portavano peggioramenti nelle prestazioni, anche se non molto elevati(1/2%).

VII. Diminuzione learning rate

Il learning rate è uno degli iperparametri più importanti nel deep learning, poiché decide la quantità di gradiente da propagare nuovamente. Questo a sua volta decide di quanto ci muoviamo verso i minimi. Un learning rate piccolo fa convergere lentamente il modello, mentre uno troppo grande lo fa divergere. E' quindi necessario configurare in modo corretto tale valore. Si è scelto un learning rate pari a 0.01 adottando inoltre una politica di learning rate decay, che nel nostro caso è step decay, cioè dopo un numero fissato di epoche(nel nostro caso 10, essendo il parametro che ci ha dato prestazioni migliori), si ha una diminuzione del passo pari a drop, che nel nostro caso risulta pari a 0.5. La funzione di decadimento è:

$$f(x) = (learningrate * drop^{\lfloor \frac{t}{10} \rfloor})$$

con t epoca corrente.
Si è provato ad utilizzare altre funzioni di decadimento, come cosine_decay,linear_decay e inverse_sqrt decay, ottenendo però un calo delle prestazioni. Un ulteriore funzione di decadimento utilizzata è stata la step_decay introducendo warmup, permettendo quindi per le prime 5 epoche un incremento del passo, ma è stata scartata dati i risultati ottenuti, più bassi della prima del 3/4%.

VIII. Ottimizzatori

Per quanto riguarda l'algoritmo di ottimizzazione è stato scelto Adagrad, poiché con gli altri algoritmi avevamo prestazioni simili o più basse. La scelta dell'ottimizzatore dipende molto anche dalla rete di pretraining utilizzata, si è notato infatti come un ottimizzatore potesse andare meglio per una rete piuttosto che per un'altra.

IX. Scelte di modello scartate

Sono state effettuate diverse scelte di modello prima di arrivare all'architettura finale. La rete utilizzata maggiormente è stata quella avente due layer dense prima di quello finale,la quale è stata successivamente scartata poiché si è notata la scarsa capacità di generalizzazione, dato che sono stati notati i tipici fenomeni di overfitting. Si è passato quindi a dimunire layer fino ad ottenere capacità di generalizzazione migliori. Nel caso in cui venisse aggiunto un solo layer dense prima dell'ultimo si è ottenuta un accuracy molto vicina a quella con il solo layer di classificazione, ma si è portata avanti la scelta di utilizzare quest'ultimo. Questo è giustificato dal fatto che il dataset è piccolo, quindi avendo pochi dati a disposizione non è necessario avere una rete molto complessa, motivo per cui si è deciso di semplificare l'architettura, ottenendo, come aspettato, risultati migliori.

X. Salvataggio del modello migliore

Si è pensato di poter salvare il modello che aveva ottenuto accuracy maggiore e loss minore durante tutto il training, in modo da poter sfruttare i pesi migliori forniti dalla rete. Questa tecnica è stata utile poiché ha permesso di passare da uno score di 0.9116 ad uno score finale di 0.9666.

XI. Conclusioni

La rete adottata ha permesso di raggiungere un accuracy sul test set pari a 0.96666 ,permettendo quindi di ottenere buoni risultati sfruttando tutte le tecniche elencate in precedenza.

References

[1] [Online]. Available: https://keras.io/applications/nasnet

[2] X. J. XiangLi, ShuoChen. (2018) Understandingthedisharmonybetweendropoutandbatchnormalizationby varianceshift. [Online]. Available: https://arxiv.org/pdf/1801.05134.pdf