Relazione del Progetto
di
Metodi Quantitativi per l'Informatica

Prof.ssa: Pirri Fiora

Tutor: Puja Francesco

di Andrea Tantucci, Andrea Wrona, Dario Zurlo

4 aprile 2018

Classificazione di immagini tramite Neural Networks

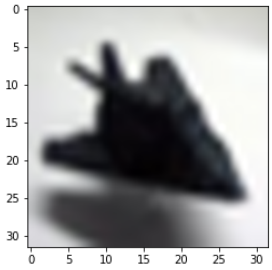
Definizione del problema

Il problema consiste nell'utilizzo di reti neurali supervisionate per la classificazione di immagini in *TensorFlow*. Si è scelto di mostrare le potenzialità di una *Convolutional Neural Network* rispetto a una *Fully Connected Neural Network*. Una *CNN* ha numerose proprietà, utili nel riconoscimento delle "features" di un'immagine. In particolare, uno strato convoluzionale è più appropriato per l'elaborazione di un input RGB, in quanto i suoi neuroni sono tridimensionali (height, width, channels) e i suoi filtri consentono di trovare caratteristiche comuni, aumentando la profondità delle immagini. Lo strato di *pooling* segue quello convoluzionale, riducendo altezza e larghezza dell'immagine a seconda della dimensione della finestra: questo strato accomuna i pixel contenuti in una finestra, calcolandone il valore massimo o, alternativamente, la media aritmetica. Infine lo strato *fully connected* consente di ottenere l'output della dimensione voluta (in genere un vettore che ha tante componenti quante sono le classi delle immagini). In questa zona della rete ogni neurone è collegato con tutti quelli dello strato successivo, e il loro obiettivo, dopo aver ricevuto in input un'immagine sotto forma di vettore unidimensionale (l'output "appiattito" dell'ultimo strato di pooling), è quello di modificarne le dimensioni tramite la ben nota formula

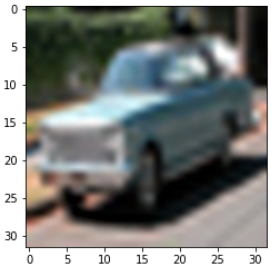
$$y = wx + b$$

ove y rappresenta l'uscita dello strato, x l'input, w i pesi (tensori bi-

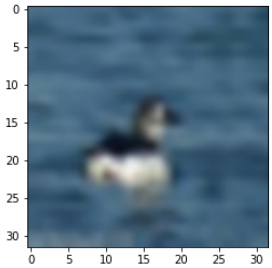
dimensionali), b i biases (tensori unidimensionali). Si ottiene infine un vettore unidimensionale con tante entries quante sono le diverse classi alle quali afferiscono le immagini. Ogni classe avrà associata una probabilità (ottenuta grazie all'applicazione della funzione di *softmax* all'output della rete: si tratta di modificare le componenti del vettore in modo tale che sommino a 1, rappresentando, appunto, una probabilità). L'obiettivo della rete è quello di computare l'errore (calcolato sulla base della *label* data in input) e minimizzarlo tramite funzioni di ottimizzazione che in maniera iterativa aggiornano i pesi della rete.
Il dataset utilizzato è il CIFAR-10: `https://www.cs.toronto.edu/~kriz/cifar.html`. Esso consiste di 60.000 immagini a colori di dimensione 32x32 divise in due blocchi: 50.000 immagini per il training set e 10.000 per il test set.
Nel dataset sono presenti dieci classi completamente diverse fra loro (aeroplano, automobile, uccello, gatto, cane, cervo, rana, cavallo, nave, camion). Gli oggetti delle stesse classi sono inoltre caratterizzati da sfondo, orientamento nello spazio, colore, luminosità diversi fra loro. Questo rende ancora più difficile per la rete accomunare gli oggetti della stessa classe: ad esempio, ogni oggetto della classe *automobile* ha forma e colore diversi da un altro. In figura 1 si riporta un esempio di 10 immagini, ognuna afferente alla propria categoria.
L'obiettivo del progetto è quello di evidenziare le differenze fra vari tipi di architetture di rete applicate al dataset di cui sopra. Sono state implementate tre architetture di rete di base, su cui poi sono state effettuate delle modifiche per studiare e interpretare le differenze nei risultati ottenuti. Ogni rete è stata usata tramite funzioni e strutture di *TensorFlow* diverse fra loro, in modo tale da evidenziare la versatilità e la potenzialità di questo Framework. La fase di train è stata effettuata tramite scheda grafica *NVIDIA GeForce 940MX 4GB*.



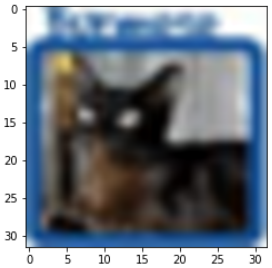
(1) Airplane



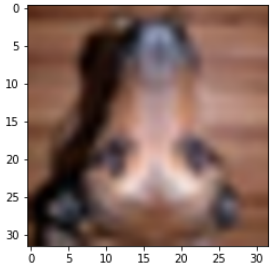
(2) Car



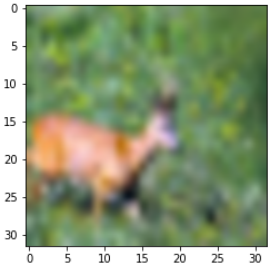
(3) Bird



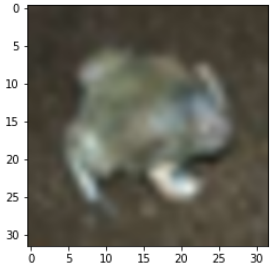
(4) Cat



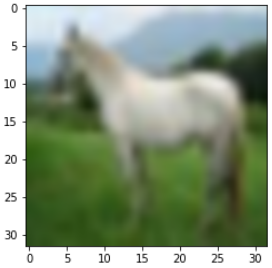
(5) Dog



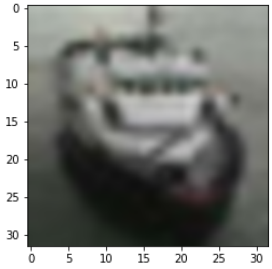
(6) Deer



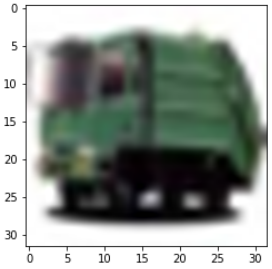
(7) Frog



(8) Horse



(9) Ship



(10) Truck

Figura 1: Images from Dataset

Sviluppo del problema

Prima Rete - *Fully Connected Neural Network*

La prima rete è stata costruita per evidenziare le problematiche legate all'uso di neuroni completamente connessi per elaborare le immagini. Essa è costituita da cinque strati completamente connessi, con un numero di neuroni iniziale di 500 e decrescente di 100 con l'aumentare della profondità. Una *dropout* segue gli ultimi due layers: questa è una tecnica secondo la quale non vengono considerati alcuni neuroni della rete. Questo significa che durante ogni passo di training ogni nodo della rete viene mantenuto con probabilità p: ciò consente di prevenire l'*overfitting* prematuro, ovvero quel fenomeno che si verifica quando la rete si allena in maniera eccessiva sul training set, non essendo poi in grado di predire nella maniera corretta le immagini del test set. La probabilità p di mantenimento scelta è del 75%. I pesi e i bias sono stati inizializzati con una distribuzione normale con media nulla e deviazione standard 1. La funzione di attivazione prescelta è la *ReLu function*: essa è definita nella seguente maniera:

$$f(x) = \begin{cases} 0 & \text{se } x \leq 0 \\ x & \text{altrove} \end{cases}$$

ove x rappresenta ogni pixel dell'immagine. La ReLu ha due vantaggi fondamentali rispetto a funzioni di attivazione non lineari, come ad esempio la *sigmoide*:

1. ridotta probabilità che il gradiente si annulli: infatti, per $x > 0$ il gradiente della ReLu è costante, mentre quello della sigmoide diminuisce all'aumentare di $|x|$;

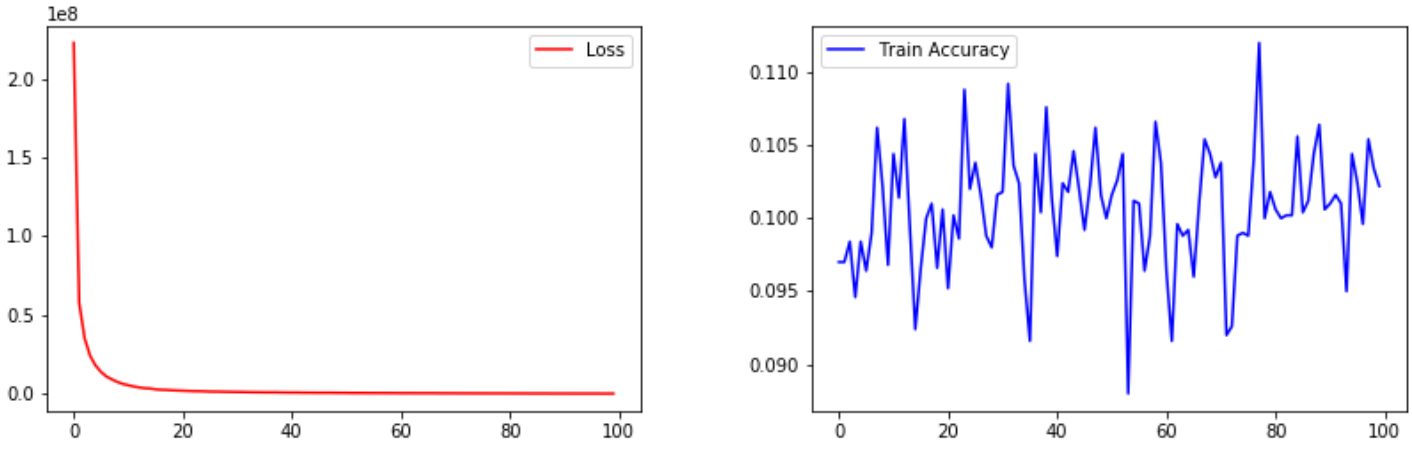
2. sparsità: essa si presenta chiaramente per $x < 0$. La sparsità semplifica in maniera sostanziale l'apprendimento della rete, a differenza di quanto avviene tramite la densità della sigmoide, che ha pochissimi valori nulli.

L'algoritmo utilizzato per aggiornare i pesi e, quindi, minimizzare l'errore, è *Adam Optimizer* con learning rate di default 1e-3.
L'algoritmo "Adam" prende il nome da *adaptive moment estimation* e fu presentato da alcuni ricercatori dell'Università di Toronto nel 2015. Esso è facile da implementare ed è molto efficiente dal punto di vista computazionale poiché richiede un basso utilizzo di risorse del computer (RAM, CPU) e quindi è particolarmente indicato per la risoluzione di problemi di minimizzazione con migliaia o milioni di dati e/o parametri. Si basa su *SGD* (Stochastic Gradient Descent), ovvero un'approssimazione stocastica dell'algoritmo di Newton (Gradient Descent) che presuppone che la funzione da minimizzare si possa scrivere come somma di funzioni differenziabili. Quindi, secondo SGD, i pesi si aggiornano computando il gradiente soltanto su una di queste funzioni differenziabili, presa in maniera casuale. Adam tiene traccia sia della media dei gradienti, sia della media del quadrato dei gradienti, provvedendo a farle decrescere esponenzialmente (i cosiddetti *momenti*). Indicando con α il learning rate di aggiornamento dei parametri finale, con β_1 e β_2 i rates di aggiornamento delle media, con g_t il gradiente al passo t-esimo, con m_t il primo momento, con v_t il secondo momento, con θ_t i parametri e con ε un valore numerico arbitrariamente piccolo per evitare la divisione

per zero, l'algoritmo è dunque definito nel modo seguente:

$$
\begin{aligned}
t &= t + 1 \\
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (\beta_2) \cdot g_t^2 \\
\tilde{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
\tilde{v}_t &= \frac{v_t}{1 - \beta_2^t} \\
\theta_t &= \theta_{t-1} - \alpha \cdot \frac{\tilde{m}_t}{\sqrt{\tilde{v}_t} + \varepsilon}
\end{aligned}
$$

In input alla rete viene fornito in maniera deterministica e statica il dataset, diviso in mini batch di 1000 elementi, non modificato in alcun modo, sotto forma di immagini in riga (*dimensione* = *height* x *width* x *channels*). Come si può osservare in figura 2, l'errore parte da un valore spropositato (circa 1e+8) e dopo 100 epoche, nonostante sia rapidamente sceso, è ancora molto elevato (circa 1.2e+5). L'accuratezza del train invece è piuttosta casuale e non è crescente.

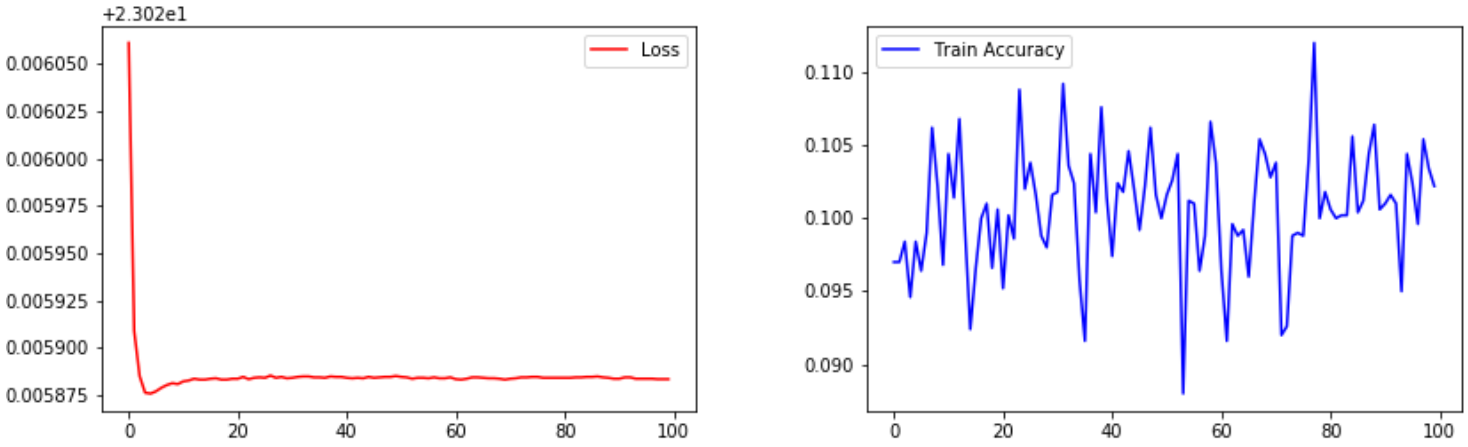


(a) Loss

(b) Train Accuracy

Figura 2: First Net Random Normal Weights

Come ci si aspettava, questo significa che la rete non è allenata bene sul dataset: infatti la percentuale di successo della train accuracy oscilla fra valori piuttosto casuali e non crescenti, come dovrebbero essere in un allenamento ottimale. Questo è dovuto al fatto che una semplice neural network con back propagation performa bene con immagini della stessa categoria che differiscono l'una dall'altra per pochi pixel (si veda ad esempio *MNIST*).
Inizializzando invece i pesi e i bias con una distribuzione normale troncata con media nulla e deviazione standard 1e-4 (si tratta di una derivata della distribuzione gaussiana, ottenuta riducendo gli estremi destro e sinistro da $+\infty$ e $-\infty$ a valori finiti simmetrici), si incontra il problema opposto: l'errore complessivo è troppo piccolo, come si vede in figura 3, e l'algoritmo di ottimizzazione non riesce a minimizzarlo nel modo corretto. Ne segue, come prima, un allenamento casuale da parte della rete, che non riesce ad apprendere nella maniera corretta.



(a) Loss

(b) Train Accuracy

Figura 3: First Net Truncated Normal Weights

Come visto, questa architettura di rete non è idonea per risolvere il problema di classificazione del CIFAR-10 dataset, quindi è necessario il passaggio ad una CNN.

Seconda Rete - *Convolutional Neural Network*

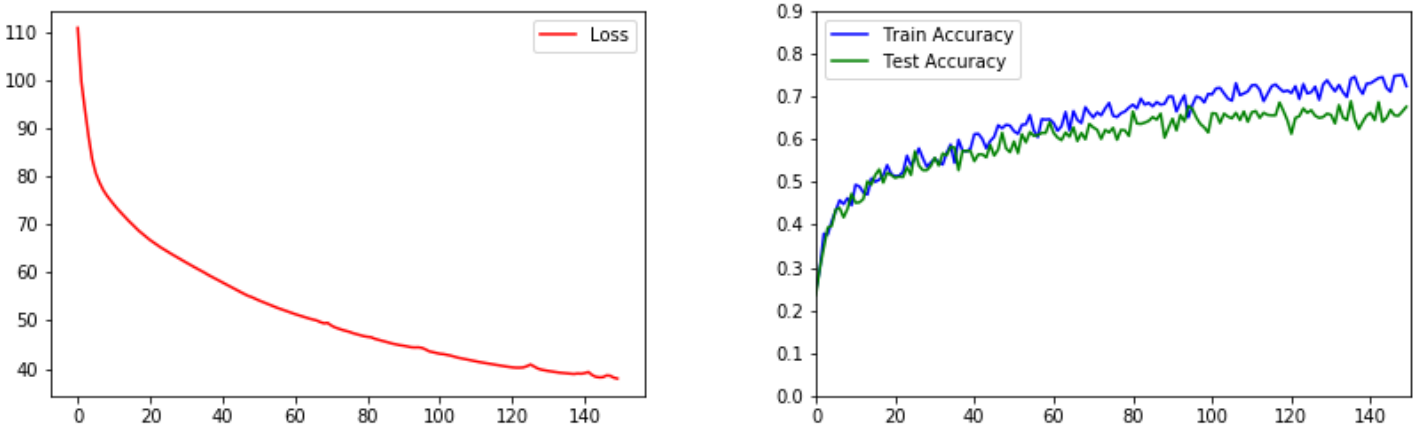
La seconda rete mostra come l'utilizzo di strati convoluzionali permetta un miglioramento sostanziale nell'apprendimento delle immagini da parte della rete. Da un punto di vista prettamente tecnico, a differenza della prima rete, essa è stata implementata facendo un uso ordinato del grafico di rete di TensorFlow: ad ogni strato e ad ogni operazione è stata assegnata una cosiddetta "variable name" tramite la funzione *tf.name_scope*. Questo è utile solo per migliorare l'eleganza e la comodità di visualizzazione del codice, ma non comporta differenza dal punto di vista algoritmico rispetto agli altri approcci tecnici, infatti le operazioni eseguite dipendono dalle funzioni di *TensorFlow* interne ai blocchi e non da come sono definiti i blocchi stessi. La rete è composta da due strati convoluzionali seguiti da due strati di pooling. Il primo strato convoluzionale dispone di 64 filtri, di dimensione 5x5, con stride 1, padding settato a *SAME*, in modo da non ridurre la dimensione dell'immagine in output. Il primo strato di pooling segue la politica del *max pooling*, cioè riduce la dimensione dell'immagine calcolando di volta in volta il valore massimo fra i pixel presenti nella *sliding window*. Indicando con F la dimensione della finestra, con S lo stride, con W_{in}, H_{in}, C_{in} le dimensioni di

ingresso, le dimensioni in uscita saranno le seguenti:

$$\begin{cases} W_o = \frac{(W_{in}-F)}{S} + 1 \\ H_o = \frac{(H_{in}-F)}{S} + 1 \\ C_o = C_{in} \end{cases}$$

Si è scelto di porre $F = 2$ e $S = 2$, ottenendo quindi in output un dimezzamento di altezza e larghezza dell'immagine. Il secondo strato convoluzionale è uguale al primo, salvo il fatto che presenta una riduzione dei filtri da 64 a 32. Il secondo strato di pooling è identico al primo. Seguono due strati completamente connessi, che ricevono in input l'output del secondo strato di pooling *flattened*, cioè "schiacciato" in riga. Il numero di neuroni è rispettivamente di 64 e 32: la scelta di tali parametri è dovuta al fatto che, più è alto il numero di neuroni, minore è l'accuratezza computata sia sul train che sul test set. I pesi relativi all'intera rete sono stati inizializzati in maniera casuale seguendo una distribuzione gaussiana troncata con media nulla e deviazione standard = 0.03; per i biases invece si è preferita una deviazione standard di 0.01 per la rete convoluzionale e di 1 per la fully connected network. Tali scelte sono dovute alla necessità di mantenere l'errore iniziale al di sotto di una soglia delle migliaia, per allenare la rete nella maniera migliore possibile. Dopo ogni strato completamente connesso, è stata eseguita una *dropout*. La funzione di attivazione scelta è la *ReLu function*, mentre l'algoritmo di ottimizzazione è *Adam Optimizer*, con learning rate 1e-3.
La rete è stata inizialmente allenata con l'intero train set di 50.000 immagini, che sono state normalizzate per limitare ogni pixel nel bound $[0 : 1]$: questo contribuisce a produrre un errore di partenza ragionevolmente basso. Le labels sono state riorganizzate sotto forma di vettori *one hot encoded*, ovvero vettori di zeri che hanno un

unico “1” in corrispondenza dell’indice associato alla categoria (ad esempio, poiché la prima categoria nel dataset è *airplane*, una label scalare con valore 0 corrisponderà proprio ad un aereo, e trasformata diverrà $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$). I risultati ottenuti sono mostrati in figura 4: L’errore parte da circa 110 e decresce gradualmente col pas-



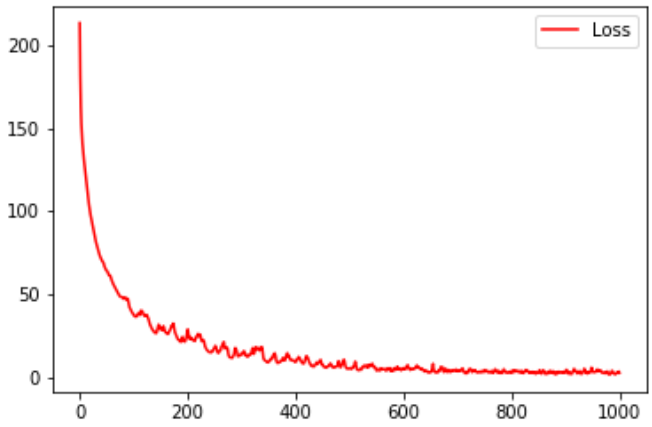
(a) Loss

(b) Train&Test Accuracy

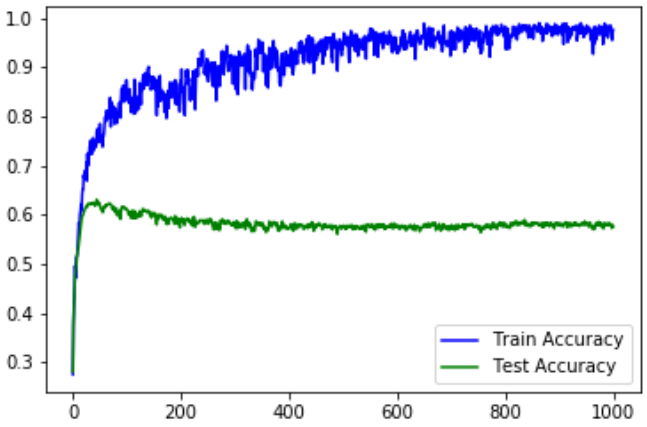
Figura 4: Second Net

sare delle epoche; l’accuratezza sul train set cresce, così come quella sul test set, chiaramente più bassa, vista la disomogeneità delle immagini (si ha un picco del 68%).
Successivamente sono state apportate varie modifiche su alcuni parametri della rete e/o sull’input stesso:

- aumento delle epoche da 150 a 1000. Tale modifica è stata fatta per mostrare un esempio di overfitting: incrementando a dismisura il numero delle epoche, la rete apprende in maniera eccellente solo le immagini del train set, e questo causa un decremento sostanziale dell’accuratezza sul test set. Ciò è mostrato in maniera chiara in figura 5;



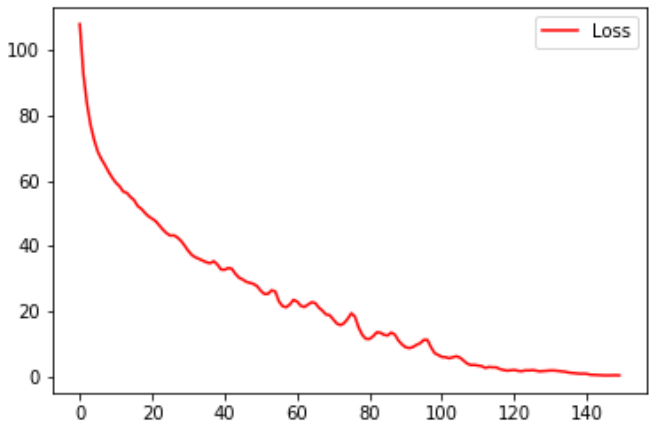
(a) Loss



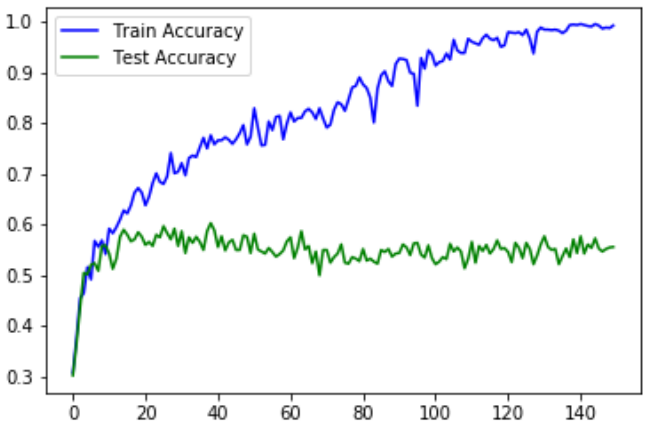
(b) Train&Test Accuracy

Figura 5: Second Net Overfitting

- aumento dei neuroni della fully connected network rispettivamente da 32 a 300 e da 64 a 500. I risultati sono riportati in figura 6. Dopo appena circa 140 epoche si è verificato overfitting:



(a) Loss



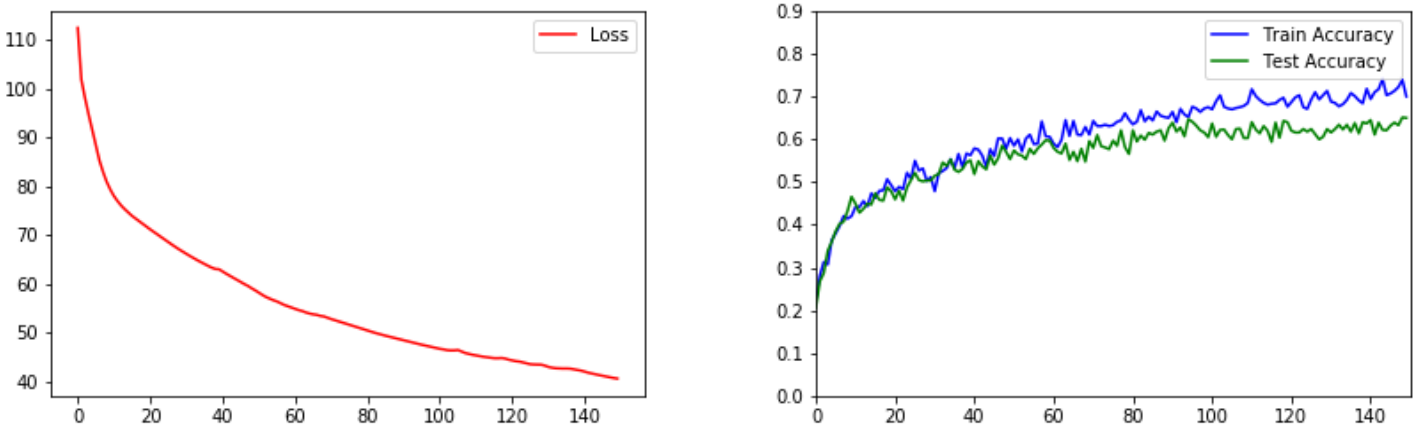
(b) Train&Test Accuracy

Figura 6: Second Net (More Neurons)

l'accuratezza del train set è circa del 95%, mentre l'efficacia sul

test supera a fatica la soglia del 60%;

- passaggio da *max pooling* a *AVG pooling*: quest'ultimo approccio, a differenza del primo, calcola come uscita la media aritmetica dei pixels presenti nella sliding window, e non il valore massimo. Questo significa che la rete dà un'importanza minore alle zone scure dell'immagine. Aumento dei filtri del secondo strato convoluzionale da 32 a 64 sul secondo strato convoluzionale, per catturare più informazioni nascoste dopo la prima applicazione del pooling. I risultati ottenuti, mostrati in figura 7, non si discostano molto da quelli ottenuti con il primo allenamento (l'accuratezza sul test raggiunge circa il 65%).
  Questo indica che modifiche sui parametri degli strati di convoluzione non sono determinanti per miglioramenti consistenti dell'efficienza di apprendimento della rete.



(a) Loss

(b) Train&Test Accuracy

Figura 7: Cifar_Second_Parameters_Changed

Come visto, una CNN performa decisamente meglio di una rete completamente connessa. Inoltre, l'accuratezza può dipendere da

diversi fattori, che rendono più o meno efficiente l'apprendimento. Tuttavia, i risultati di predizione sul test set non vanno oltre il 68%, quindi è necessario compiere uno step in più. Fino ad ora l'allenamento è stato fatto in maniera statica e deterministica, cioè ad ogni epoca alla rete veniva sottoposta sempre la stessa sequenza di immagini, uguali fra loro. Nella terza rete questo tipo di approccio verrà mutato, per favorire un apprendimento più casuale, ma soprattutto più completo.

Terza Rete - *Convolutional Neural Network* - Preprocessing delle immagini e training/testing aleatori

La terza rete costituisce la metodologia più efficace per affrontare e risolvere il problema della classificazione delle immagini del CIFAR-10 dataset nell'ambito delle Convolutional Neural Networks. Essa è stata implementata sfruttando le potenti librerie di un Framework derivato di TensorFlow, ovvero *PrettyTensor*. Tali librerie consentono la creazione di reti complesse in poche righe di codice: ciò è possibile poiché PrettyTensor assegna in maniera automatica i pesi e i biases ai vari strati in modo da rendere l'errore di partenza il più basso possibile (dell'ordine di poche unità per batch di 64 immagini), e discerne tra fase di training e fase di testing grazie ad una speciale variabile globale denominata "phase": ciò consente in maniera molto comoda di riutilizzare la medesima rete per le due fasi di lavoro. Per quanto riguarda l'architettura, essa è molto simile a quella della seconda rete, salvo presentare 64 filtri anche nel secondo strato convoluzionale, e una *batch normalization* dopo il primo strato di pooling: la batch normalization consiste nel ripetere la stessa operazione che viene fatta in input alla rete anche negli hidden

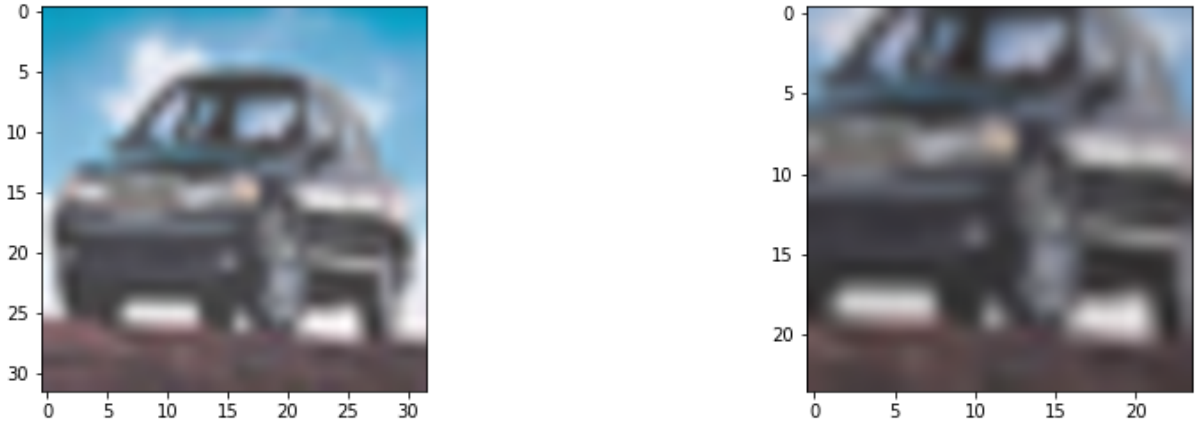
layers. L'unica differenza è che, mentre all'inizio ogni pixel dell'immagine viene diviso per 255, durante la fase di batch normalization ogni pixel viene "standardizzato", ovvero viene eseguita la seguente operazione:

$$new\_value = \frac{old\_value - \mu}{\sigma}$$

ove μ è la media di tutti i pixel dell'immagine, mentre σ è la deviazione standard degli stessi. In questo modo si riduce in maniera considerevole il "rumore" contenuto nell'immagine che viene elaborata negli hidden layers, e questo favorisce la velocità di apprendimento da parte della rete. Il numero dei neuroni dei due strati completamente connessi è rispettivamente di 256 e 128. L'algoritmo utilizzato è *Adam Optimizer*, con learning rate 1e-4: questo valore basso previene l'overfitting prematuro.
Come accennato prima, la novità di questo metodo consiste soprattutto nel *preprocessing* delle immagini. Queste, oltre ad essere preventivamente normalizzate, sono state modificate attraverso le librerie di TensorFlow. Per quanto concerne le immagini del train set:

- sono state ridotte in modo casuale ritagliandole fino a portarle alla dimensione 24x24x3: questo favorisce la scomparsa di parte dello sfondo che causava la presenza di un forte rumore nelle immagini;
- sono state girate "a specchio" (il cosiddetto *flipping*) con probabilità 0.5: in tal modo viene risolto, almeno parzialmente, il problema dei diversi orientamenti nello spazio;
- sono stati cambiati, ancora in maniera casuale, i valori di colore, contrasto, luminosità e saturazione: in particolare si è prestata la dovuta attenzione ad alzare il valore di contrasto, per favorire la distinzione fra oggetto e sfondo.

Di seguito un esempio di modifica: in figura 8 si può vedere come cambia l'immagine di un'automobile dopo il preprocessing. Lo sfondo, che è causa di confusione durante l'allenamento, viene eliminato, mentre colore, contrasto, saturazione e luminosità rimangono all'occhio umano fondamentalmente gli stessi (questo è dovuto al fatto che le modifiche sugli effetti visivi, essendo casuali, possono anche produrre in uscita pressoché la stessa immagine). Per quanto riguar-



(a) Before Preprocessing

(b) After Preprocessing

Figura 8: Modifica Automobile

da, invece, le immagini del test set, sono state semplicemente ridotte a 24x24, senza ulteriori modifiche ai valori dei pixel. Questa scelta è stata dettata perché, in fase di riconoscimento, il software deve essere in grado di distinguere fra le varie classi attenendosi alle immagini reali che gli vengono fornite in ingresso: eventuali modifiche sul test set in fase di riconoscimento post train potrebbero portare la rete a predizioni sbagliate. Ne segue che ogni alterazione va effettuata solo sul train set, per favorire un apprendimento più generalizzato possibile.

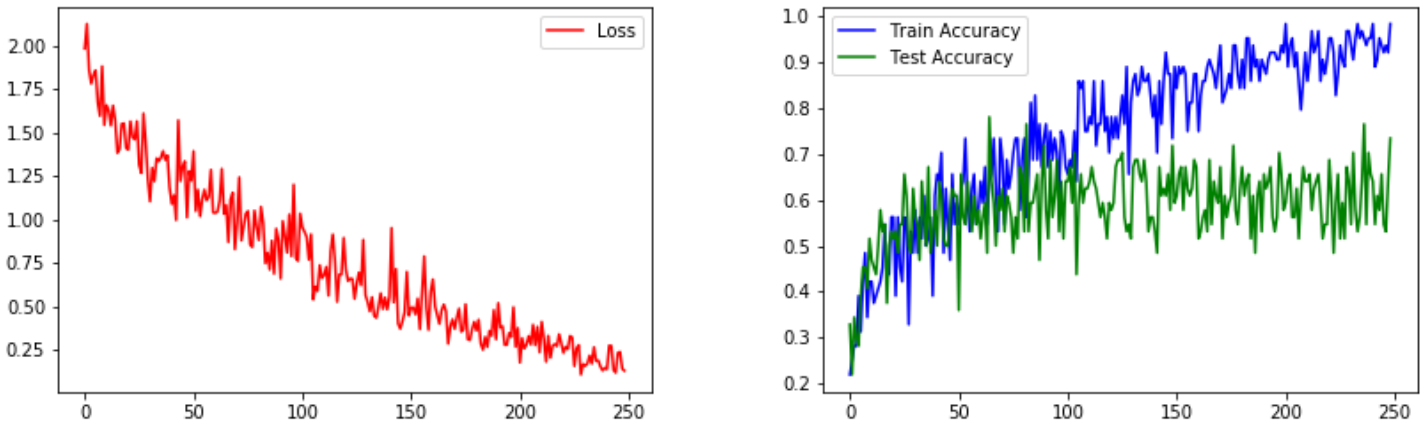
Il preprocessing può essere effettuato sostanzialmente in due modi

diversi:

1. prima dell'allenamento: in questo caso il dataset viene modificato interamente prima di essere fornito in input alla rete. È evidente che questo approccio risulta essere sia casuale in fase di modifica delle immagini, sia deterministico in fase di allenamento, poiché le immagini fornite alla rete continuano ad essere sempre le stesse, seppur modificate;

2. contestualmente all'allenamento: in questo caso i batches del dataset vengono modificati di volta in volta, quindi l'apprendimento risulta dinamico e soprattutto non deterministico. Ad ogni epoca la rete si allena su immagini completamente diverse, apprendendo molte più features di quanto accadeva in precedenza.

In figura 9 vengono riportati i risultati nel caso in cui il preprocessing venga fatto prima dell'allenamento. Si noti come le percentuali



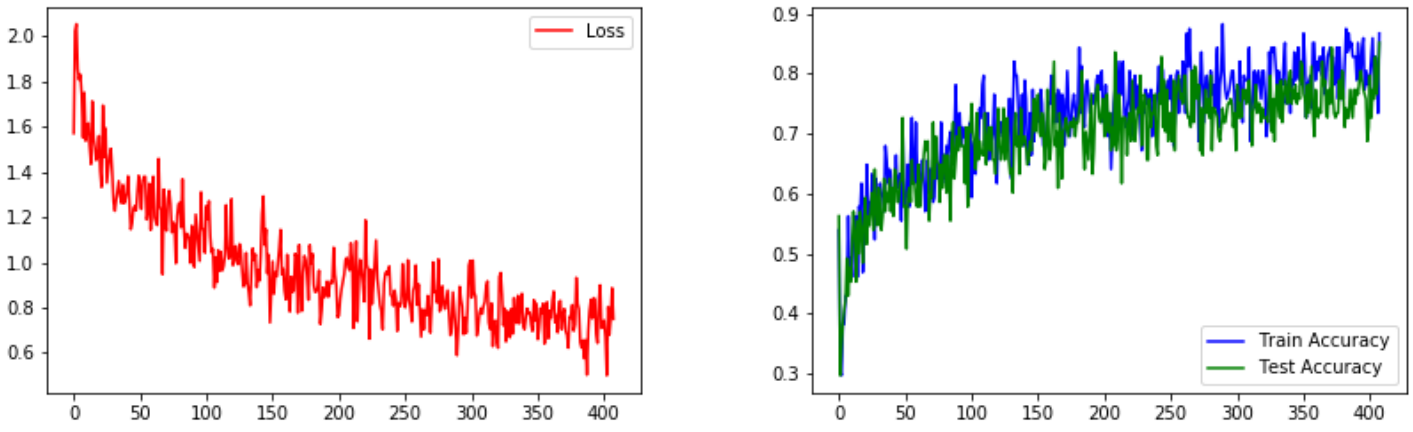
(a) Loss

(b) Train&Test Accuracy

Figura 9: Cifar_Third_Deterministic_Preprocessing

di successo sul test set siano leggermente migliori rispetto a quelle

ottenute con la seconda rete: questo è dovuto al fatto che, sebbene le 50.000 immagini di base siano le stesse, l'allenamento è stato eseguito estraendo batch randomici di 64 immagini ad ogni epoca. Questo favorisce un apprendimento sparso e non ripetitivo, come avveniva invece con la seconda rete, in cui in ogni epoca la rete veniva allenata con tutte le immagini passate sempre nello stesso ordine. In particolare si nota che, dopo 25.000 iterazioni, l'efficienza della rete è di circa il 73%, mentre l'accuratezza sul train set è prossima al 100%, e quindi si va verso l'overfitting.
In figura 10, invece, si possono osservare i risultati ottenuti qualora le modifiche vengano effettuate contestualmente alla fase di training. Come ci si aspettava, non si verifica overfitting poiché la crescita del-



(a) Loss

(b) Train&Test Accuracy

Figura 10: Cifar_Third_Random_Preprocessing

l'accuratezza del training segue quella del test. Inoltre, poiché la rete apprende in maniera casuale su batches randomici presi dal dataset, si ottengono percentuali nettamente superiori a quelle ottenute con la seconda rete: dopo circa 40.000 epoche (ne sono servite così tante poiché in input alla rete, ad ogni epoca, è stato fornito un

piccolo batch randomico di 64 immagini, e non, come accadeva per la seconda rete, l'intero dataset) si raggiunge un'accuratezza di predizione sul test set di circa l'85%. Questo dimostra come rendere allenamento e modifica contestuali sia l'approccio più efficiente per il riconoscimento di immagini afferenti a categorie completamente diverse fra loro.

Conclusioni

In questo progetto si è studiato in maniera approfondita e diversificata il problema del riconoscimento di immagini che si presentano in forme e colori completamente diversi anche fra stesse categorie di appartenenza.
I risultati ottenuti e documentati in questo lavoro portano a confermare la validità delle CNN rispetto ad altre architetture per la classificazione di immagini RGB, e soprattutto attestano come un allenamento adeguato sia subordinato ad una modifica casuale delle immagini e contestuale alla fase di training. Ciò consente alla rete, dopo un numero sufficiente di epoche, di estrapolare milioni di informazioni aggiuntive sulla natura delle varie classi, cosa impossibile quando il *feed* avviene sempre attraverso le medesime figure.

Fonti bibliografiche e Sitografia

- *http://deeplearning.net/datasets/*
- *http://cs231n.github.io/convolutional-networks/*
- *https://arxiv.org/abs/1412.6980v8*
- *https://medium.com/@nishantnikhil/adam-optimizer-notes-ddac4fd7218*
- *http://ruder.io/optimizing-gradient-descent/index.html#adam*
- *http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf*
- *https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9*
- *https://www.tensorflow.org/*
- **Géron A.**, *Hands-On Machine Learning with Scikit-Learn & TensorFlow*, **USA**, **O'Reilly Media**, **2017**