Belletti Michele

Progetto 3 – Computer vision

Lo scopo di questo progetto è quello di realizzare una Convolutional Neural Networks (CNN) in modo da classificare correttamente la categoria di appartenenza di una foto.
Il dataset usato per il training della rete è quello utilizzato nella ricerca [1] composta da 15 categorie.
Il linguaggio scelto per la realizzazione del progetto è stato MATLAB.

Parte 1

La prima parte del progetto è incentrata nella realizzazione della seguente CNN (figura 1) e nel suo allenamento a riconoscere la categoria di appartenenza di una foto.

#	type	size
1	Image Input	64×64×1 images
2	Convolution	8 3×3 convolutions with stride 1
3	ReLU	
4	Max Pooling	2×2 max pooling with stride 2
5	Convolution	16 3×3 convolutions with stride 1
6	ReLU	
7	Max Pooling	2×2 max pooling with stride 2
8	Convolution	32 3×3 convolutions with stride 1
9	ReLU	
10	Fully Connected	15
11	Softmax	softmax
12	Classification Output	crossentropyex

Figura 1

Per la realizzazione della struttura della rete è stato utilizzata l'app deep network design presente in MATLAB, che permette di esportare in codice la struttura della CNN usando un interfaccia grafica.

Manipolazione dei dati

Per il training e la verifica, i dati a disposizione sono stati divisi in un rapporto 85% per il training ed il restante per il testing.
I dati per il training sono stati ulteriormente divisi, utilizzando la funzione di MATLAB, *splitEachLabel()*, in questo modo:
un 85 % per il training e il restante 15 per la validazione.
Adesso le foto, per essere utilizzate dalla CNN, devono essere dimensionate per essere adatte all'ingresso del primo layer (64x64 pixel x1 livello colore) e per far questo, utilizzando la funzione *augmentedImageDatastore(),* possiamo ottenere una dimensione della foto che rispetta il vincolo ed inoltre permette di modificare le foto in modo di avere casuali variazioni in traslazione e rotazione delle stesse.

Impostazione learning

Di seguito si preparano le impostazioni per il training utilizzando la funzione *trainingOptions()* ponendo nel nostro caso:

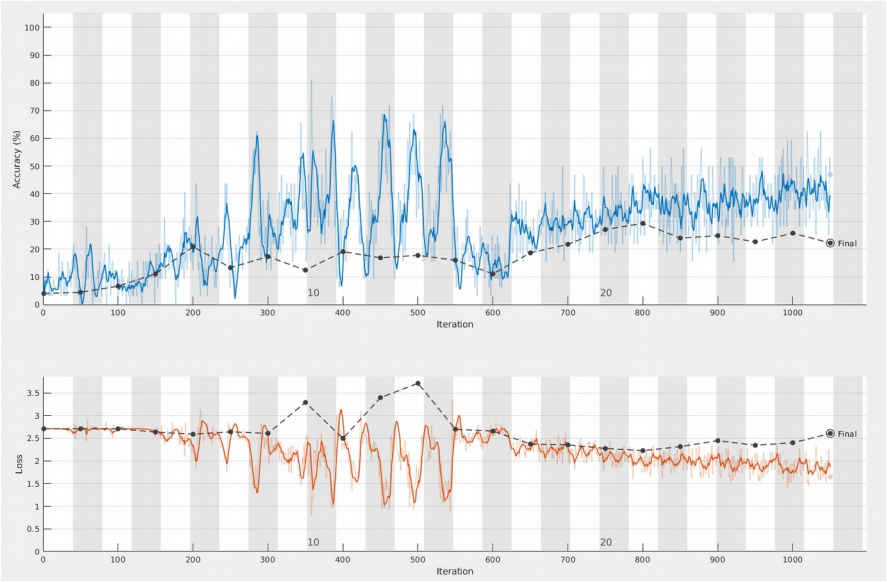
- *'sgdm'* , per utilizzare il stochastic gradient descent with momentum;
- 'MiniBatchSize', 32, per avere una minibatch size di 32.

Infine, note le caratteristiche imposte per il progetto, per avere dei pesi iniziali derivati da una Gaussiana con media 0 e deviazione standard 0,01 si modificano i layer convolution2dLayer() e fullyConnectedLayer() inserendo la seguente componente: "WeightsInitializer",'narrow-normal'. Mentre per avere un bias iniziale uguale a 0 non è richiesta alcuna variabile perché è un impostazione di default in MATLAB.

Risultati

Adesso si riportano i dati ottenuti durante il training del risultato migliore.
I metodi scelti per terminare il training sono stati i seguenti:

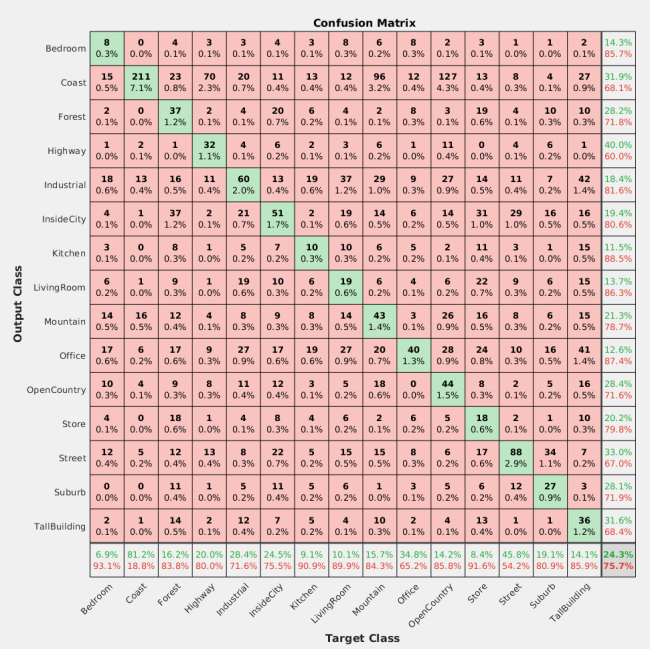
- fissare un limite max di epoche, cioè quante volte l'insieme di foto, usate per il learning (epoch), viene ripetuto prima di fermarsi;
- fissare un valore massimo di quante volte la funzione Loss ha superato il suo valore minimo sui dati di validazione, nel caso migliore il valore era posto a 5.



Si nota che per il caso migliore (Figura 2) il sistema è intervenuto prima della fine del valore massimo di epoche fissate ottenendo il valore massimo di accuratezza del 21,64 % sulle immagini di test.

Figura 2

Con una confusion matrix qui riportata (Figura 2):



Se il modello non viene precedentemente fermato si ottiene un peggioramento dell'accuratezza del 3% sui test data.

Figura 3

Parte 2

La seconda parte del progetto consiste nel migliorare le performance del modello utilizzando le seguenti variabili:

- aumentare le capacità di learning aggiungendo alle foto delle traslazioni casuali, rotazioni e riflessioni rispetto agli assi principali. Utilizzando la funzione *imageDataAugmenter()* e ponendo le seguenti variabili alla funzione *'RandXReflection',1* ; *'RandYReflection',1* ; *RandXTranslation',[-3 3]* ; *'RandYTranslation',[-3 3]* e *'RandRotation',[-5,5]* ha portato ad un aumento del 5-10% all'accuratezza (ora 25-30%) sulle immagini per il test, il tutto a discapito di un maggiore tempo di training;
- aggiungere un batch normalization layer prima del layer ReLU, il che ha portato ad ottenere un'accuratezza del 50% circa, con un tempo di learning molto minore;

- modificare la dimensione dei layer di convoluzione usando le seguenti dimensioni 3x3, 5x5 e 7x7 rispettivamente per il primo, secondo e terzo layer, il che ha portato ad un aumento dell'accuratezza media del 5%;
- modificare eventuali altri parametri:
  - modificando la minibatchsize incrementandola, a parità di maxepoch di 50, il che ha portato ad un aumento del 1-2 %, data la poca differenza tra 128 e 256 ho lasciato il valore su 128;
  - passando ad un maxepoch di 100, che porta il sistema in regime di overfitting, ha mantenuto mediamente costante il risultato di accuratezza;
  - scegliendo un altro metodo di risoluzione, il metodo rmsprop non ha modificato di tanto il risultato, invece il metodo adam ne ha aumentato il risultato medio del 5 %
  - variando i parametri per i pesi, il sistema non ha modificato di molto il risultato portando a lasciare i valori di default
- aggiungere un layer di dropout non ha permesso ai grafici di accuratezza e di validazione di distanziarsi troppo, permettendo di non andare in overfitting ed ottenendo una percentuale di accuratezza simile al precedente;
- applicare un ensemble of networks di 5 reti CNN. Tali reti hanno avuto il training set uguale e poi attraverso un sistema di votazione è stata scelta la predizione più comune. Le CNN sono state modificate variando il maxepoch, minibatch size e ripetendo un insieme di layer, modificandone la dimensione e il kernel. Queste operazioni hanno portato ad avere un accuratezza media del 65 % con un picco di precisione del 66,30%

In conclusione, inizialmente il sistema aveva un'accuratezza media del 23% ed alla fine, dopo aver applicato le operazioni sopra descritte, abbiamo raggiunto un'accuratezza del 65%. Alla fine il metodo migliore è stato quello di porre un normalization layer prima del layer ReLU (incremento del 20%).

Parte 3

La terza parte si basa sul transfer learning utilizzando una rete già addestrata (AlexNet), questa parte si divide in due sezioni:

- la prima, predire la categoria di appartenenza modificando solo l'ultimo fully connected layer fc8 tenendo fissi i pesi dei layer precedenti (utilizzando la funzione di MATLAB *freezeWeights()*). Per far questo, dopo aver sostituito gli ultimi layer e modificando il layer fc8 per avere in output le 15 categorie del nostro training set, abbiamo cercato di ottenere il miglior risultato modificando le impostazioni sul learning dei pesi. Alla fine abbiamo ottenuto un accuratezza max del 81%;

- la seconda, addestrare una rete SVM sulle feature estratte dall'ultimo layer di convoluzione di AlexNet, MATLAB dispone della funzione *activation()* che fornisce le feature per un determinato layer (in questo caso 'conv5'). Presi i dettagli delle feature, utilizzando l'error correcting Output code (funzione MATLAB *fitcecoc()*) e ponendo il metodo risolutivo un sistema SVM lineare con metodo di comparazione one vs one abbiamo addestrato il sistema a capire quali determinate feature appartengono a una categoria. Il miglior risultato ottenuto è stato del 73,53%.

Bibliografia:

[1] Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), volume 2, pages 2169–2178.