Università di Catania

Trashbin Triplet Classifier

Progetto Deep Learning (LM-18)
Università degli Studi di Catania - A.A 2021/2022

Danilo Leocata
Docente: Giovanni Maria Farinella, Antonino Furnari

June 4, 2022

1 Introduzione

L'obbiettivo del progetto è realizzare una rete siamese su un dataset contenente secchi della spazzatura. Per classificare la capienza rimamente di bidoni utilizzati per la spazzatura, e che in particolare sia in grado di distinguere tra: pieno, vuoto, a metà (codice Github TODO:) È stato già fatto uno studio sul dataset (raccolto con TODO: e disponibile TODO:) il cui progetto è disponibile al seguente indirizzo.

Il progetto è stato sviluppato utilizzando 'python 3.9.9' e 'pytorch-lighting'' versione . Il training del modello è stato effettuato su MacBook Pro (16-inch, 2019) con processore Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, ram: 16 GB 16 GB 2667 MHz DDR4 e GPU AMD Radeon Pro 5300M 4 GB Intel UHD Graphics 630 1536 MB. Sfortunatamente, ad oggi, il modello di GPU non è supportato per l'accelerazione del training e di conseguenza il training è stato effettuato su CPU.

Il link del progetto attuale è disponibile al seguente indirizzo.

2 Scelta dell'architettura

È stato trovato opportuno l'utilizzo di una *Rete Triplet* per il raggiungimento dell'obbiettivo assegnato, dato che il dataset è composto da 3 classi, è stato pensato che questo approccio sarebbe stato migliore per dare una netta 'differenza' tra la classe 'mezzopieno' e 'vuoto' e 'pieno' (scrivi meglio). Ogni tripletta (I_i, I_j, I_k) contiene dunque tre elementi

- L'ancora I_i - L'esempio positivo I_j (che in breve ha la stessa classe di I_i) - L'esempio negativo I_k, cioé un elemento diverso dalle classi di I_i, I_j

La coppia I_i, I_j sarà un esempio positivo, e la coppia I_i, I_k esempio negativo: I_i verrà mappato vicino a I_j e lontano da I_k, offrendo sempre un esempio positivo e uno negativi relativo allo stesso elemento.

Immagine architettura di esempio

3 Preparazione del dataset

Partendo dal dataset originale (link), si è avuta la necessità di riadattarlo in triplette per farlo funzionare per il modello scelto: ad ogni elemento di ancora verrà associato un elemento positivo ed uno negativo che sarà scelto randomicamente in base alle due classi disponibili (esempio: se l'ancora è 'vuoto', l'elemento negativo sarà 'pieno' o 'mezzo') È stato trovato più efficiente, principalmente per effettuare le prove, 'fissare' il dataset in un csv, evitando di creare le triplette dinamicamente. Per generalizzare ancora di più, sarebbe utile eseguire il codice di creazione del dataset più volte e cambiare il datamodule durante il load del checkpoint: infatti .sample() utilizzato pescherà randomicamente dal dataframe un elemento randomico: in questo modo vi è improbabile che le triplette generate siano uguali ad altre.

4 Scelta del modello

Nello studio precedentemente effettuato, sono stati presi in esame alcuni modelli pretrained, in particolare è stata implementata la tecnica del feature extraction che ha ottenuto dei buoni risultati. (elenco dei modelli provati?)
Per le prove iniziali è stato preso in esame mobilenetV2.

5 Scelta del batch size e learning rate

È stato visto (dove?? link???) e testato che nel metric learning, batch più grandi possono portare ad una migliore generalizzazione di batch piccoli, oltre a ridurre di poco il tempo durata del training, di conseguenza dopo varie prove effettuate Per evitare errori causati dalla memoria, è stata utilizzata una funzione proprietaria di pytorch lighting che permetetva di provare i batch size, evitando riempimenti di memoria successivi .... (?????)

inserisci immagine di prova delle architetture
Riguardo al learning rate, inizialmente è stato utilizzato quello del progetto precdeente ma con scarsi risultati. È stata utilizzata una funzione proprietaria di pytorch lighting che permetteva di trovare il migliore learning rate, effettuando un test .....

inserisci immagine di prova delle architetture

6 Scelta della loss

Sono state prese in considerazioni due loss....

References

[1] An Effective Algorithm for Minimum Weighted Vertex Cover Problem

[2] A memory-based iterated local search algorithm for the multi-depot open vehicle routing problem