**TRACKING VEHICLES AND CALCULATE SPEED**

Lo scopo di questo progetto è quello di utilizzare le attuali reti neurali addestrate per l’object detection su video. Il risultato che vogliamo ottenere è quello di creare un progetto in grado di analizzare un video contenente veicoli in movimento, di salvarsi i veicoli per riconoscerli nel video e di stimare la velocità di essi come pixel percorsi per frame.

Il primo passo da compiere è quello di prendere familiarità con le attuali reti e di studiarne pregi e difetti. A questo scopo ho creato il file allegato “RETI NEURALI PER OBJECT DETECTION” che racchiude brevemente le reti studiate per questo scopo. A fronte di quello che si può leggere ho deciso di procedere nel progetto con l’utilizzo di Single Shot Detector. Questa tipologia di rete è adatta al nostro scopo in quanto impiega un tempo ridotto per l’analisi di ogni frame restituendo un video in tempo reale rallentato, ma accettabile. Il problema delle SSD è nella scarsa precisione di analisi, questo problema però non è risultato tale in quanto i video analizzati erano video “semplici” ossia composti da una strada e dal passaggio di uno o più veicoli.

Il progetto è diviso in diversi file:

* Main
* DataMemory
* FunctionUtils
* Models
* Resource – VideoInput
* Mscoco\_label\_map

**Main**

File principale che contiene la struttura del progetto con i richiami ai vari file. La sua funzione è descritta nel seguente modo.

Tramite le librerie OpenCv carica il video da analizzare e crea uno scrittore per video, lo utilizzeremo per salvare un nuovo video in cui ogni frame conterrà i risultati a schermo della nostra analisi. Carica in memoria il file Msoco\_label\_map, questo non è altro che una lista di valori che utilizzeremo durante l’analisi della rete neurale per assegnare all’id del risultato il nome dell’oggetto. Esempio se troveremo una macchina la rete restituirà nei risultati l’id 3, che confrontato nella lista caricata ci restituirà il nome “car”.

Fatto ciò andiamo a caricare il modello di rete che vogliamo utilizzare nell’analisi. Abbiamo due possibilità. Utilizzare una rete salvata in precedenza sul computer oppure tramite le librerie di tensorlfow\_hub caricare direttamente dal web il modello. Questo ci permette di utilizzare diversi tipi preaddestrati semplicemente richiamando l’url corretto. Come esempio fornisco il seguente link dove prendere la lista di modelli caricabili <https://tfhub.dev/tensorflow/collections/object_detection/1> , dentro ogni modello sarà fornito un esempio di utilizzo con url necessario da inserire in hub.load(“”).

Per ultimo carico una lista di colori da utilizzare tramite la funzione Standard\_colors.

Ora inizia la fase di analisi video. Caricato a memoria tutto quello che ci serve il programma non fa altro che leggere frame per frame il video precedentemente caricato. Ogni frame viene analizzato dalla rete che restituisce una struttura di valori contenente, per nostro interesse, id riconosciuto, boxe che contiene l’oggetto e la percentuale di correttezza. Da questa struttura faccio una prima scrematura prendendo solo gli id di macchine, moto, bus, camion e che abbiamo una percentuale di correttezza superiore al 50%. Salvo per ogni valore che supera il processo Boxe, centro e classe. Superato questo step procedo con l’analisi tramite file DataMemory dello storico del video e dei valori relativi a posizione e velocità. Dopo di che disegno su frame ogni boxe, con nome e velocità, presente nel frame e in memoria, ogni suo centro e se presente il caso particolare di predizione traiettoria (spiegato nel file DataMemory). In fine salvo il frame nello scrittore di video e procedo al frame successivo. Finita l’analisi scrivo i risultati dello storico in un file txt e chiudo l’esecuzione.

**DataMemory**

Questo file presenta il fulcro dell’analisi. Esso contiene l’analisi che viene compiuta ad ogni frame per creare uno storico dei veicoli che passano nel video, ne calcola la velocità ed a ogni frame cerca veicoli nuovi oppure aggiorna tramite calcoli su posizione lo storico di quelli presenti. Per fare un esempio quando un nuovo veicolo viene trovato in un frame, a quello successivo si cerca il veicolo nello storico, se lo si trova si aggiorna la posizione, la velocità ecc.. se non lo si trova si crea un nuovo veicolo nello storico che successivamente verrà ricercato nei frame successivi.

La funzione principe di questo file è il TrackingMemory. Essa prende in input lo storico, i valori trovati e scremati della rete neurale, ed una distanza massima. Questa funzione è divisa in due controlli principali: Veicolo trovato, predizione di veicolo trovato. Ciclando per ogni risultato della rete i veicoli salvati in storico, calcolo le distanze tra il centro in analisi ed ogni centro dello storico, calcolo la velocità che servirebbe per passare da un centro all’altro come distanza in pixel / frame.

Il primo controllo confronta:

* Id delle classi combacino.
* distanza tra i centri sia minore di un valore massimo dato in ingresso alla funzione.
* L’ultimo aggiornamento del veicolo selezionato in storico non sia maggiore di 15 frame, ossia se un veicolo non è più stato trovato nei 15 frame successivi lo consideriamo uscito di scena e quindi non più utile in questa analisi.
* Controllo che la velocità calcolata per andare da punto a punto non sia 10 volte superiore a quella del veicolo storico, è improbabile un tale divario tra valori con eccezione che il veicolo abbia velocità ossia che non sia stato salvato come nuovo nel frame precedente.

Se passa questo controllo allora mi salvo l’id del valore nello storico. Fatto questo per tutti i valori dello storico guardo se ho qualche id salvato. Se ne ho 1 allora aggiorno quel valore, se ne ho più di uno confronto tra tutti per vedere quale è il più vicino alla posizione precedente.

Nel caso in cui nessun id sia stato trovato allora passo al secondo id, l’idea è quella di predirre la traiettoria di ogni veicolo in storio come retta passante per il centro attuale e quello precedente, sempre del veicolo in storico, e calcolare la distanza retta punto con il centro sotto esame.

Il controllo confronta:

* Distanza punto retta inferiore a valore massimo.
* Confronto che le accelerazioni dei veicoli siano nella stessa direzione.
* L’ultimo aggiornamento del veicolo selezionato in storico non sia maggiore di 15 frame.

Se trovo riscontro come prima salvo id, controllo quale id sia il più vicino e faccio update. Inoltre ritorno valore della retta e dei punti cosi da disegnare a schermo eccezione.

Se dopo questi due confronti non ho avuto riscontri quello che mi rimane da fare è creare un nuovo valore per lo storico.

La struttura dello storico è cosi definita

#DataMemory' definition

#list of vehicle's that update during the time of video

#[car1, car2, truck3, motorbike4]

#STRUCT

# [ id, [class] , [frame] , [center] , [velocity] , [boxes] , [iteration] ]

# id: is object in list

# [class] -> [class]

# contains only id's class of object

#

# [frame] -> [id\_frame, id\_previous\_frame]

# contains number of actual frame and number of previous frame in which it was detected

# [center] -> [center, prev\_center]

# contains center of actual prediction and previous.

#

# [velocity] -> [velocity, previous\_velocity, average\_velocity, deltaD]

# contains actual velocity, load like actual position-previous\_position / Time\_actual - Time\_previous ,

# previous velocity, mean velocity load like average of all velocity, deltaD load like actual position - previous position

#

#  [boxes] -> [boxes]

# contains boxe

# [iteration] -> [num\_of\_iteration]

# contains number of iteration of that object, how many frame detect it

**FunctionUtils**

Contiene 3 funzioni:

Standard\_colors() restituisce una lista di colori da usare per disegnare i risultati.

Load\_label() ritorna i valori dal file Msoco\_label\_path

WriteResult() scrive file txt contenente i valori dello storico.