



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Riconoscimento oggetti da flussi dati RGB e LIDAR

Relatore: Prof. Simone Bianco

Co-relatore: Davide Mazzini

Relazione della prova finale di:
Gianluca Scarpellini
Matricola 807541

Anno Accademico 2017-2018

A tutti coloro che mi hanno sostenuto senza ben capire quello che stavo facendo.
Non sono solo.

Ringraziamenti

Sono molte le persone che dovrei ringraziare e a cui vorrei dedicare questa pagina. Innanzitutto la mia famiglia, per avermi sostenuto e aiutato a raggiungere questo primo, importante traguardo. Ringrazio la mia fidanzata Giada per diversi suggerimenti nell’im paginazione e nell’estetica di questo lavoro, ma soprattutto per avermi ascoltato, supportato ed essermi stata vicina. Un grazie va anche i miei compagni di questi 3 anni di università, Davide Mattia Daniel e non solo, per i momenti di studio e di svago passati insieme. Mille ringraziamenti vanno a tutti i miei amici e a tutti coloro che si sono interessati, da informatici e non, al lavoro che stavo svolgendo. Ringrazio infine il mio correlatore, Davide Mazzini, per avermi sopportato e avermi dato suggerimenti essenziali durante questi 3 mesi di esperienza in laboratorio, e prof. Schettini per avermi dato quest’opportunità.

Abstract

La computer vision in ambito automotive ha visto un crescente interesse delle aziende negli ultimi anni (es. progetto Waymo, Google, 2009). Aziende hardware e software stanno focalizzando la loro attenzione su nuovi sensori e tecnologie di visione che affianchino le camere tradizionali. Tra i sensori più utilizzati in questo ambito, ci sono i laser scanner (LIDAR), i quali sono in grado di generare una rappresentazione dello spazio circostante in 3D sotto forma di nuvola di punti. Di conseguenza, sta emergendo chiaramente la necessità di strumenti adatti a interpretare nuove tipologie di dati e ricavarne informazioni. La computer vision ha avuto una rapida evoluzione nell'ultimo decennio e, con l'introduzione di Reti Neurali Convoluzionali (Alexnet, 2012), ha raggiunto livelli di precisione prima inaspettati. La tesi inizia con la descrizione del flusso dati LIDAR, e delle problematiche legate suo impiego per l'object detection. Il progetto di tesi ha previsto lo sviluppo di una pipeline di conversione che ha permesso di generare l'immagine di depth a partire dai dati grezzi point-cloud provenienti dal LIDAR. A questa fase, è seguita una fase di studio delle idee cardine delle reti artificiali specializzate nella detection di oggetti. In particolare è stato scelto di approfondire YOLO (Redmon et. al., 2016 [8]). L'implementazione YOLO per il framework pytorch è stata estesa all'interno della tesi all'impiego di dati RGB-D e addestrata sul dataset Kitti Vision Benchmark Suite, il quale è un progetto open source liberamente accessibile che mette a disposizione immagini RGB e dati LIDAR acquisiti strada in ambiente non controllato. I risultati ottenuti sono infine presentati suddividendoli in tre classi di difficoltà: easy, moderate, e hard.

Indice

Ringraziamenti	II
Abstract	III
1 Flusso dati Lidar	1
1.1 Introduzione: Cosa è un laser scanner?	1
1.2 Funzionamento	1
1.3 TOF (time of flight)	1
1.4 Portata e precisione	2
1.5 Problematiche	2
1.6 Pipeline di conversione	2
1.6.1 Step 1	2
1.6.2 Step 2	3
1.6.3 Nearest neighbour search (naive)	4
1.6.4 Nearest neighbour search (kd-tree)	4
1.6.5 k-Nearest neighbour search	4
2 Reti neurali per l'object detection	6
2.1 Modello	6
2.2 Loss	6
2.3 Apprendimento	7
2.4 Struttura delle Reti Neurali	8
2.4.1 Neurone	8
2.4.2 Layer Fully-connected	9
2.5 Profondità e potere espressivo	9
2.6 CNN	9
2.7 Layers di una CNN	9
2.7.1 In breve	9
2.7.2 INPUT	10
2.7.3 CONV	10
2.7.4 POOL	11
2.8 Yolo	11
2.8.1 In breve	11
2.8.2 Detection	12
2.8.3 Non-maximal suppression	13

3 The KITTI Vision Benchmark Suite	14
3.1 Are we ready for Autonomous Driving?	14
3.2 Setup	15
3.2.1 Laser scanner	15
3.2.2 Camere	15
3.2.3 Altra strumentistica	16
3.3 Caratteristiche	16
3.4 Difficoltà: Easy, Moderate, Hard	16
3.5 Metriche	17
3.5.1 Precision, Recall e Fscore	17
3.5.2 Average Precision	18
3.5.3 Mean Average Precision	18
4 Experimenti	19
4.1 RGB	19
4.1.1 Modello preaddestrato su dataset COCO	20
4.1.2 Modello addestrato su Kitti	20
4.1.3 Fine-tuning del modello con pesi COCO su dataset Kitti	21
4.1.4 Fine-tuning del modello con pesi pre-addestrati VOC su dataset Kitti con anchors COCO	21
4.1.5 Fine-tuning del modello con pesi pre-addestrati VOC su dataset Kitti con anchors VOC	22
4.1.6 Fine-tuning modelo con pesi pre-addestrati su immagini di depth	22
4.2 RGBD	24
4.3 Curva Precision/Recall	25
4.4 Riconoscimento real-time di oggetti	25
5 Conclusione e considerazioni finali	28
A Appendice	30
A.1 Point Cloud	30
A.2 RGB	30
A.3 RGBD	31
A.4 Pipeline di conversione: esempi di codice	32
A.4.1 Step 01	32
A.4.2 Step 02	32
A.4.3 Read Calibration	33
A.5 K-d tree	33
A.6 Back propagation	34
A.7 Stochastic Gradient Descent	35
A.8 Fine tuning	35
Riferimenti bibliografici	36

Elenco delle figure

1.1	Esempio di immagine di Depth (visualizzata con colormap) generate e la relativa immagine RGB (The Kitti Vision Benchmark Suite)	5
2.1	Modello di una rete neurale artificiale con 3 layer	8
2.2	Modello di un neurone artificiale e similitudine con il neurone biologico	8
2.3	same padding no strides convoluzionale	10
2.4	Layer padding strides odd	11
2.5	Max-pooling	12
2.6	Architettura YOLO	12
2.7	Interpetazione della risposta del modello YOLO	13
2.8	Esempio pipeline YOLO	13
3.1	LIDAR e videocamere montate	14
3.2	HDL-64E di Velodyne	15
3.3	Camera PointGrey Flea2	16
3.4	Definizione di Precision e Recall [1]	18
4.1	Esempio di frame risultato. Il modello restituisce le bounding box e la classe degli oggetti presenti	19
4.2	Esempio segmentazione tratto dal dataset COCO. La classe Bici e persona sono entità separate	20
4.3	Label predette tramite il modello dell'EXP 6 su immagine di Depth (viene qui riportata anche l'immagine RGB per chiarezza)	23
4.4	Curve precision/recall degli esperimenti fatti	25
4.5	Label predette tramite il modello dell'EXP 5 (Milano, 2018)	27
4.6	Label predette tramite il modello dell'EXP 5 (Milano, 2018)	27
A.1	Immagine di Depth tratta da Kitti Dataset; il canale d'intensità è scalato sull'intera colormap riportata sulla destra	31
A.2	Immagine RGB della stessa scena	31
A.3	Esempio tratto da Stanford CS231n	34

Elenco delle tabelle

4.1	AP per le 3 classi di inferenza e le 3 difficoltà del modello YOLO preaddestrato	20
4.2	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB	21
4.3	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB usando pesi pre-addestrati COCO 60 epoche	21
4.4	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB usando pesi pre-addestrati VOC con ancore COCO per 160 epoche	22
4.5	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB usando pesi pre-addestrati VOC 70 con ancore VOC per 70 epoche	22
4.6	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite Depth usando pesi pre-addestrati del EXP5 per 80 epoche	23
4.7	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGBD usando pesi pre-addestrati del EXP5 per 60 epoche	24
4.8	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGBD usando pesi pre-addestrati del EXP5 per 100 epoche	24
4.9	AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGBD usando pesi pre-addestrati del EXP5 per 140 epoche	24
4.10	Tabella riassuntiva dell'average precision, suddivisa per classe e difficoltà, per ciascun esperimento	26

Listings

A.1 Prima fase di conversione	32
A.2 Seconda fase di conversione	32
A.3 Lettura della calibrazione dello strumento	33

Capitolo 1

Flusso dati Lidar

1.1 Introduzione: Cosa è un laser scanner?

Un laser scanner è uno strumento di acquisizione attivo, che consente di determinare il valore di distanza degli oggetti dal dispositivo di acquisizione. I dati acquisiti sono salvati sottoforma di point-cloud (si approfondisce all'appendice A.1). Il termine Laser lascia intuire il principio di funzionamento dello strumento. Un fascio di laser è proiettato nell'area circostante lo strumento, mentre un sensore apposito acquisisce successivamente la risposta degli oggetti situati nella portata dello strumento.

1.2 Funzionamento

Nei sistemi moderni, i laser sono usati per fare una scansione dell'area che circonda il sensore. La mappa generata può rappresentare un'area di 360° dello spazio attorno al sensore. Nella sua versione più semplice, il laser scanner consta di un emettitore di beam laser e di un sensore di ricezione in grado di registrare luce della stessa lunghezza d'onda del raggio emesso. Appositi specchii ampliano il singolo laser, suddividendo i beam su più piani differenti allo scopo di ottenere una mappa di più piani XY lungo l'asse Z¹. Lo specchio ruota attorno all'asse di rotazione Z, acquisendo informazione spaziale dell'area circostante. La frequenza di rotazione dello specchio può raggiungere i 44 Hz, permettendo l'acquisizione di una molitudine di dati grezzi.

1.3 TOF (time of flight)

Il sensore ricettore cattura i beam riflessi dagli oggetti dello spazio circostante. Data la velocità della luce nell'ambiente (in funzione della temperatura), lo strumento è capace di calcolare la distanza dell'emettitore dall'oggetto a partire dal tempo $\Delta_t/2$ impiegato dalla luce riflessa per raggiungere sensore ricettore. Un problema che si pone è la necessità di

¹Il frame del sensore di acquisizione si compone di 3 assi ortogonali: x = avanti, y = sinistra, z = su

strumentazioni in grado di discriminare sino al $10e^{-6}$ secondi, a seconda della rapidità con cui i beam riflessi raggiungono il sensore.

1.4 Portata e precisione

La portata di un laser scanner varia a seconda del metodo di valutazione della distanza. I dispositivi TOF, che misurano la distanza valutando il tempo di volo, possono raggiungere una range massimo di 1000 m. L'errore di acquisizione di un laser-scanner è distribuito secondo una normale, e il valore di sigma rappresenta la precisione dello strumento. La precisione si attesta a circa di $4 \div 6$ mm per una distanza di 100 m, e decresce all'aumentare la distanza.

1.5 Problematiche

I sensori attivi, in particolare quei sensori che usano fasci di luce, incontrano forte limitazioni laddove i beam laser incontrano superfici riflettenti o vetri. Oggetti specchianti potrebbero infatti deviare il laser, compromettendone la ricezione da parte del sensore. Il difetto dei sensori a laser si palesa ancor di più con le superfici trasparenti:l'oggetto non riflette i beam, che di conseguenza non producono eco nel sensore ricettivo. Laddove i beam non sono ricevuti dal sensore, il valore assegnato al punto è di out-of-range, ossia il valore massimo. Oggetti trasparenti o riflettenti si possono comunemente trovare in strada (es. finestrini, targhe) o in sua prossimità.

1.6 Pipeline di conversione

La conversione dei point-cloud a disposizione in immagini in scala di grigio è stato fondamentale allo scopo proposto. A differenza di [10], l'approccio seguito ha permesso di generare immagini di depth della porzione di spazio frontale allo strumento di acquisizione. L'immagine di Depth così ottenuta può successivamente essere elaborata individualmente, come immagine 8bit a 256 livelli, o sovrapposta con l'immagine RGB frontale, al fine di ottenere un'immagine a 4 canali RGBD.

1.6.1 Step 1

Il primo step di conversione è rappresentabile come una funzione $f(Im(t), S_{LIDAR}, P, t) = m_t$, dove $Im(t)$ rappresenta l'immagine RGB acquisita con la camera sinistra al tempo t , S_{LIDAR} rappresenta l'insieme dei punti acquisiti dal laser scanner al tempo t e P è la matrice di proiezione dei punti di S_{LIDAR} nel frame 2D dell'immagine . Il risultato è una mappa m_t che racchiude le informazioni acquisite e dipende direttamente dall'istante di acquisizione.

Le informazioni sulla calibrazione degli strumenti² sono necessarie al fine di generare l'immagine di depth finale. La calibrazione è solitamente effettuata all'inizio delle registrazioni e mantenuta uguale per l'intera sessione giornaliera. La calibrazione comprende la matrice di roto-traslazione R_{rect} e la matrice di proiezione post-raddrizzamento P_{rect} . E' inoltre fondamentale estrarre dal file di calibrazione rigida la matrice di rotazione $R_{\text{velo}}^{\text{cam}}$ e di traslazione $t_{\text{velo}}^{\text{cam}}$: la composizione delle due matrici risulta in una matrice 4x4 di roto-traslazione

$$T_{\text{velo}}^{\text{cam}} = \begin{bmatrix} R_{\text{velo}}^{\text{cam}} & t_{\text{velo}}^{\text{cam}} \\ 0 & 1 \end{bmatrix} \quad (1.1)$$

Un punto LIDAR 3D x può essere proiettato sull'immagine della telecamera i tramite una matrice di proiezione. La telecamera scelta in questa pipeline è la telecamera RGB sinistra. La matrice di proiezione

$$P = P_{\text{rect}}^{L_RGB} R_{\text{rect}} T_{\text{velo}}^{\text{cam}} \quad (1.2)$$

permette di trasformare il vettore associato al punto LIDAR $p_v = (x_v, y_v, z_v)$ in un punto $p' = (x', y')$ del frame della telecamera scelta.

1.6.2 Step 2

Per ciascuna mappa m_t si estrae quindi la giornata di registrazione associata g_t e si leggono le informazioni di calibrazione degli strumenti, al fine di calcolare la matrice (1.2). La point-cloud di dati LIDAR associati a m_t viene proiettata in un immagine 2D di punti.

$$\text{LIDAR}_{2D}(x', y') = P p(x, y, z), \forall p(x, y, z) \in S_{\text{LIDAR}} \quad (1.3)$$

Questa non è ancora l'immagine di Depth finale. E' innanzitutto necessaria una successiva elaborazione che escluda i punti esterni all'immagine RGB. Si crea un vettore booleano

$$\text{keep} = \begin{cases} 1 & \text{se } p' \text{ rientra in size(Im)} \\ 0 & \text{altrimenti} \end{cases} \quad (1.4)$$

che è successivamente applicato alla matrice 1.3

$$\text{Depth}_{\text{raw}} = \text{LIDAR}_{2D}(\text{keep}, 1) \quad (1.5)$$

L'equazione 1.5 genera una matrice di punti $n \times 1$, dove $n = n_{\text{righe}} \times n_{\text{colonne}}$ dell'immagine RGB. Per il canale di profondità è stato utilizzato il canale x dei dati LIDAR grezzi. Questa scelta è stata dettata dal fatto che:

²Nota: La trasformazione non può prescindere dal setup di acquisizione. Le informazioni qui riportate fanno quindi riferimento al dataset Kitti Vision Benchamrk Suite [5]. I frame sono disposti come:

- Camera: x = destra, y = giù, z = avanti
- Velodyne: x = avanti, y = sinistra, z = su

- l'asse x è difatti proiettato in avanti rispetto al sensore Velodyne. L'informazione x per ciascun punto è quindi la proiezione in lunghezza del punto stesso;
- l'informazione di distanza longitudinale alla direzione di moto dell'auto è solitamente il principale parametro con cui il guidatore misura la distanza dal mezzo che lo precede e dagli oggetti della strada;

Dal reshape della matrice ottenuta si ricava infine l'immagine di Depth $n_{\text{righe}} \times n_{\text{colonne}}$. L'immagine così generata presenta tuttavia notevoli svantaggi. Lo strumento di acquisizione di depth, difatti, non è in grado di generare punti 3D densi quanto un'immagine. 1.5 presenterà quindi punti di informazione sparsi, e buchi interni che possono compromettere l'utilità dell'immagine stessa. E' quindi necessario procedere per interpolazione, al fine di ottenere un immagine di Depth densa e sovrapponibile all'immagine RGB.

1.6.3 Nearest neighbour search (naive)

Al fine di generare un'immagine di Depth densa, in cui tutti i pixel hanno un valore assegnato, è necessario ricorrere a un algoritmo di interpolazione. La scelta è ricaduta su Nearest neighbour search. L'implementazione di Nearest neighbour search naive è risultata computazionalmente svantaggiosa. Questa soluzione consiste nel calcolare la distanza di ciascun query point da tutti gli altri punti del database, selezionando successivamente il punto più vicino. Il tempo di esecuzione dell'approccio è $O(dN)_{d=2}$, ossia lineare rispetto al numero di punti N presenti nell'immagine. Un'approccio più efficiente è possibile.

1.6.4 Nearest neighbour search (kd-tree)

L'algoritmo Nearest neighbour search su un kd-tree (di cui si riporta una spiegazione a A.5) è un'efficiente alternativa al approccio naive. Per lo scopo è stata impiegata l'implementazione open-source della collection VLFeat. Il tempo di esecuzione medio dell'approccio kd-tree è $O(\log N)$ in fase di ricerca e $O(N \log N)$ in fase di costruzione dell'albero.

1.6.5 k-Nearest neighbour search

L'algoritmo può essere esteso al fine di individuare i k punti di distanza minima dal query point. Sono stati generati due set di immagini Depth con rispettivamente k=1 e k=3. Valori di k>3 non sono stati indice di miglioramenti evidenti.

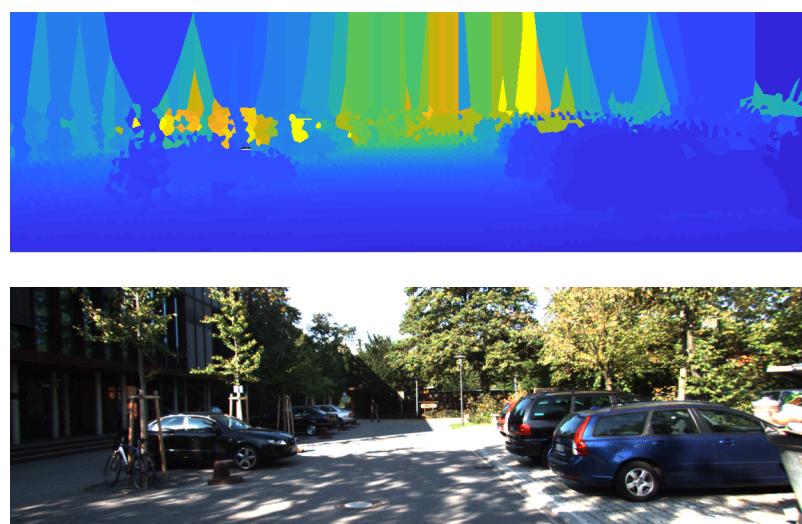


Figura 1.1: Esempio di immagine di Depth (visualizzata con colormap) generate e la relativa immagine RGB (The Kitti Vision Benchmark Suite)

Capitolo 2

Reti neurali per l'object detection

2.1 Modello

Si può definire un modello di apprendimento supervisionato (nella sua forma più semplice) quale una funzione lineare $\vec{y} = f(\vec{x})$ che, dato in input un vettore colonna \vec{x} restituisce un vettore di classi \vec{s} di dimensioni $n \times 1$, dove n è il numero di classi da predire. Si interpretano i valori $s(i), \forall i$ quali score associati a ciascuna classe i . Il calcolo del vettore di score si ottiene come:

$$s = f(x) = W_1 x \quad (2.1)$$

Definiamo W matrice di attivazione, o dei pesi, del modello. Quest'ultimo può essere esteso nella funzione:

$$s = f(x) = W_2 \max(0, W_1 x) \quad (2.2)$$

In (2.2) si è esteso il modello (2.1) introducendo un seconda matrice di attivazione W_2 e la funzione non lineare max. L'uso della non-linearità introduce nel modello una maggiore complessità computazionale, ma ha dei benefici. Per il principio di sovrapposizione degli effetti, due funzioni lineari tra loro legate possono essere collassate in un'unica matrice. La funzione max permette al modello di disgiungere le due matrici, che sono così considerate come due layer separati. Seguendo lo stesso principio è possibile estendere il modello, introducendo più layer tra loro separati da funzioni lineari.

2.2 Loss

Il modello come quello descritto dall'equazione (2.2) non è ancora sufficiente all'apprendimento. E' difatti necessario introdurre il concetto di apprendimento (o Training) e di errore (o Loss). Un modello, al fine di apprendere e migliorarsi, deve disporre dell'informazione di errore: deve difatti conoscere di quanto la sua predizione si discosta dalla realtà. Dato un vettore di input x_i e la classe y_i a cui il vettore x_i appartiene, e un iper-parametro di margine Δ si definisce funzione di Loss L la funzione:

$$L_i = \sum_{j \neq i} \max(0, s_j - s_{y_i} + \Delta) \quad (2.3)$$

Il significato della funzione (2.3) è il seguente: l’errore L_i che il modello commette è pari alla somma delle differenze tra lo score corretto s_{y_i} e i valori scorretti $s_{j \neq i}$. Il parametro Δ , definito in fase di progettazione, permette di considerare solo le differenze tra score maggiori di Δ , ossia $s_j - s_{y_i} > \Delta$. La sommatoria è utilizzata per accumulare le differenze (ossia gli errori) per ogni classe di predizione del task. E’ utile notare che più di una matrice di pesi potrebbe portare a un valore di loss ottimale. E’ compito di chi progetta il modello considerare una funzione di regolarizzazione $R(W)$ che imponga una “preferenza” tra diverse matrici W . La funzione R che si può considerare tiene conto della norma L2 dei pesi:

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad (2.4)$$

La loss nella sua forma completa è pari alla media valori di loss per ciascuna classe, sommata di un valore di regolarizzazione $R(W)$ con peso λ .

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W) , \quad (2.5)$$

La funzione di regolarizzazione permette una maggiore generalizzazione durante l’apprendimento, imponendo una preferenza per vettori di norma minore. Il valore L dell’espressione (2.5) è usato per valutare quanto il modello è preciso nella sua predizione, e di conseguenza quanto ha appreso.

2.3 Apprendimento

L’apprendimento dei valori dei pesi (ossia, delle matrici di attivazione) necessita di un vettore direzionale: un gradiente, interpretabile come la derivata della funzione, che dia una direzione di crescita di ciascun parametro. La derivata di una funzione f in un punto può infatti essere interpretata quale la direzione della funzione in un intorno dello stesso. Si parla quindi di “sensitività” della funzione rispetto a ciascuna delle sue variabili. Dato un modello quale (eq. 2.2) e una funzione di loss (eq. 2.3), si definisce operazione di “forward” l’espressione:

$$L(f(x)) \quad (2.6)$$

La composizione di modello e Loss rappresenta la funzione di un problema di minimizzazione: è necessario minimizzare il valore di output di 2.6. Calcolato quindi il gradiente della funzione rispetto a ciascun parametro mediante back-propagation (si fa riferimento a A.6), si impiega un algoritmo numerico di minimizzazione (tra i più semplici e utilizzati l’algoritmo Stochastic Gradient Descent approfondito a A.7) al fine di minimizzare l’errore e migliorare la capacità predittiva del modello. Dal punto di vista analitico, una rete neurale può quindi essere espressa come una combinazione di gate più o meno complessi. Il processo di forward ($L(f(x))$) restituisce il valore di loss del modello, mentre ciascun gate conserva il proprio valore di output. Tramite back-propagation si valuta il gradiente della $L(f(x))$ rispetto ai pesi di ciascun neurone della rete. Infine, un algoritmo iterativo di approssimazione stocastica aggiorna i pesi dei neuroni al fine di minimizzare il valore di loss.

2.4 Struttura delle Reti Neurali

Una rete neurale è modellabile come una composizione di funzioni (e.g. 2.1). Ciascuna funzione può essere interpretata come un "layer" che compone la rete. La "profondità" di una rete neurale si definisce come il numero di layer di cui la rete è composta.

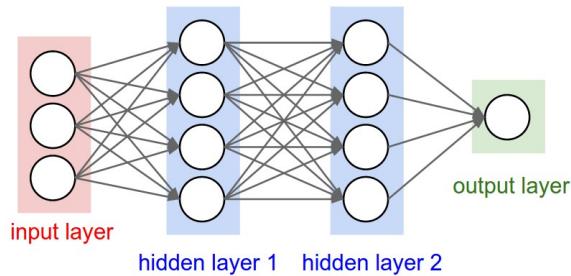


Figura 2.1: Modello di una rete neurale artificiale con 3 layer

2.4.1 Neurone

Si definisce neurone il componente minimale di una rete neurale artificiale. E' costituito da una matrice di attivazione W $n \times m$, dove n è il numero di valori in input e m il numero di output, e da un vettore di bias b . La risposta del neurone a un input x è:

$$f\left(\sum_i w_i x_i + b_i\right) \quad (2.7)$$

La funzione non lineare f , detta anche funzione di attivazione, è scelta in fase implementativa ed è spesso modellata con la funzione max. In quest'ultimo caso, prende il nome di ReLu

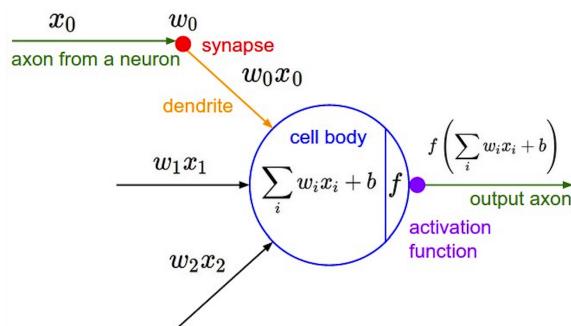


Figura 2.2: Modello di un neurone artificiale e similitudine con il neurone biologico

2.4.2 Layer Fully-connected

Un layer fully-connected è costituito da neuroni di collegamento tra due layer adiacenti (di qualunque tipo). Ciascun neurone del layer viene attivato da tutti i valori di input. Data W matrice del layer FC, interpretiamo ciascun neurone come una riga della matrice W. Ciascuna riga, ossia ciascun vettore di attivazione, deve avere numero di colonne pari al numero di input.

2.5 Profondità e potere espressivo

Può essere mostrato che data una qualunque funzione continua (x) e un margine di errore ϵ , esiste una rete neurale costituita $N(x)$ da un layer Fully-connected e un layer di output tale che:

$$\forall x, |f(x) - N(x)| < \epsilon \quad (2.8)$$

Ossia, una rete neurale $N(x)$ è un approssimatore universale di una qualunque funzione continua. Inoltre, si evince induttivamente che una rete con due o più layer è (matematicamente) ugualmente espressiva. Perché modellare una rete neurale con più di due layer? Si è osservato empiricamente che la profondità di una rete può essere un fattore chiave nella sua capacità di convergere alla funzione $f(x)$ desiderata. La profondità si è rivelata particolarmente importante nella modellazione di reti neurali convoluzionali.

2.6 CNN

I concetti introdotti ai paragrafi precedenti sono qui estesi al caso di reti finalizzate all’individuazione e alla classificazione di oggetti all’interno di un’immagine. Si assume quindi che l’input di una rete neurale convoluzionale (CNN) sia esplicitamente un’immagine di $n \times m$ pixel (eventualmente a più canali). In particolare, i layer di una rete convoluzionale sono organizzati in 3 dimensioni: larghezza, lunghezza e profondità (in questo caso riferita alla dimensione del singolo layer). Più layer 2D affiancati in profondità permettono l’estensione di filtri a due o più dimensioni.

2.7 Layers di una CNN

2.7.1 In breve

Una rete neurale convoluzionale, come per le reti neurali già introdotte in 2.1, si struttura come una composizione di layer tra loro connessi.

$$\text{INPUT} \rightarrow [\text{CONV} \rightarrow \text{ReLU}] * N \rightarrow \text{POOL?} * M \rightarrow [\text{FC} \rightarrow \text{ReLU}] * K \rightarrow \text{FC} \quad (2.9)$$

L’equazione 2.9 è un modello di possibile rete neurale convoluzionale: l’input (un’immagine a 1 o più canali) è data in input al modello. La coppia layer convoluzionale (CONV)- non linearità (ReLU) è impiegata per acquisire informazioni circa i pattern delle classi su cui fare inferenza. Un layer di pooling (POOL) segue una sequenza di CONV-ReLu e riduce

le dimensioni del tensore. Una serie di layer fully-connected (FC), seguiti da ReLu salvo per l’ultimo layer, producono le informazioni finali. L’ultimo layer FC produce il tensore di output.

2.7.2 INPUT

Il tensore in input alla rete può essere interpretato come un layer di valori fissi. Il layer di input è infatti privo di pesi, non essendo coinvolto nel processo di training.

2.7.3 CONV

I layer fondamentali di una rete specializzata nel riconoscimento di immagini. I neuroni di un layer convoluzionale sono connessi a una sola regione ristretta del input, a differenza dei layer fully-connected. I valori della regione dell’immagine sono moltiplicati per i pesi di attivazione del neurone, secondo il modello 2.7. In fase implementativa è anche possibile specificare la dimensione in profondità del layer: nel caso di layer a profondità > 1 , ciascuno strato 2D è indipendente dagli altri. La risposta del layer è la composizione ordinata delle risposte di ciascuno strato 2D.

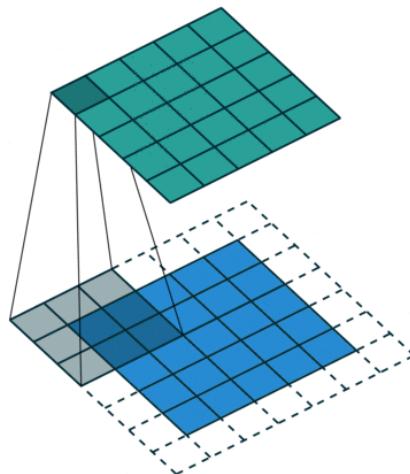


Figura 2.3: same padding no strides convoluzionale

I parametri convoluzionali, che sono applicati a tutte le convoluzioni del layer, riguardano: padding, dimensione del kernel e stride.

- La dimensione del kernel, indicata come un tensore (n, m, k) , rappresenta la dimensione del tensore di attivazione di ciascun neurone, e corrisponde allo spazio dell’input (anch’esso rappresentabile come tensore a più di una dimensione) che ciascun neurone considera (nella figura 2.3 il kernel è una matrice quadrata 3x3).

- Il padding rappresenta il numero di righe e colonne da aggiungere al vettore di input (nella figura 2.3 il valore di padding è 2).
- Il valore di stride rappresenta il passo che segue una convoluzione. Dati due neuroni adiacenti appartenenti allo stesso layer e impostato un valore di stride, i due neuroni prenderanno in considerazione regioni dell’input a distanza pari al valore di stride. A parità di kernel e padding, un valore di stride > 1 riduce il numero di neuroni del layer necessari a considerare l’intero tensore di input. Nella figura 2.4 i parametri sono : kernel quadrato 3x3, padding 1, stride 2. La dimensione dell’input (in blu) è una matrice di 6x6. La dimensione del layer è pari a:

$$S_{\text{Layer}} = (S_{\text{Im}} - S_{\text{kernel}} + \text{padding})/\text{stride} + 1 \quad (2.10)$$

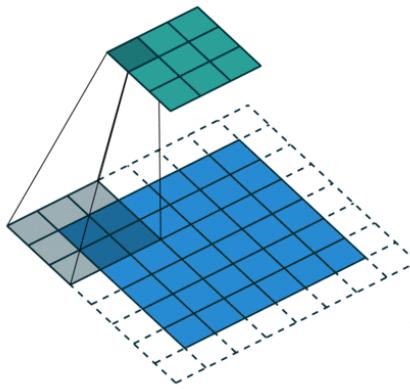


Figura 2.4: Layer padding strides odd

2.7.4 POOL

I layer di pooling sono impiegati allo scopo di ridurre le dimensioni del tensore di input. A tale scopo, è necessaria definire una funzione che discriminai tra i valori di input quello più significativo. La funzione comunemente scelta è max. Un parametro di stride permette di indicare il passo di movimento del filtro max, e quindi la dimensione dell’output. Il layer di pooling è privo di pesi, e non è quindi soggetto ad alcun apprendimento.

2.8 Yolo

2.8.1 In breve

YOLO [8] è una rete neurale convoluzionale che restituisce, per l’immagine di input data, informazioni relative sia alle bounding box degli oggetti presenti sia la classe degli oggetti stessi. Le performance sono migliori rispetto a reti che considerano questi due task (object

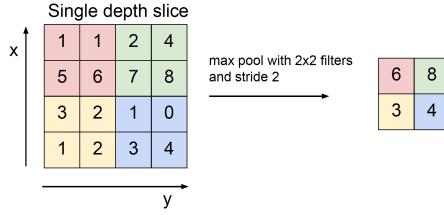


Figura 2.5: Max-pooling

detection e object recognition) separatamente, e che prevedono una pipeline complessa di più reti tra loro connesse (e.g. RNN). Inoltre, durante il processo di apprendimento la rete riceve in input l’immagine nella sua interezza, ed è quindi guidata ad apprendere le feature globali dell’immagine. Un forte vincolo è tuttavia imposto dalla suddivisione dell’immagine in celle discrete: a ciascuna cella è associata un’unica classe, limitando il numero di oggetti vicini che il modello può predire.

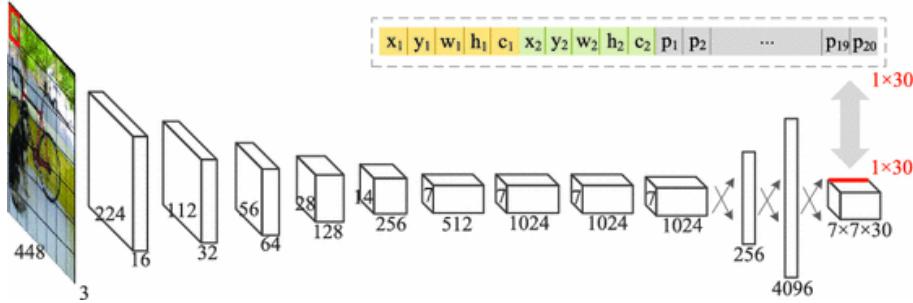


Figura 2.6: Architettura YOLO

2.8.2 Detection

Per il task di object detection si può procedere con un approccio di regressione con l’aiuto di anchors. Si definisce anchor una bounding box di dimensioni fissate in fase di implementazione. Compito della regressione è individuare l’offset, ossia la differenza, tra la anchor data e la reale bounding box dell’oggetto. Si divide l’immagine in blocchi di uguale dimensione, e si assegna un set di ancore per ciascun blocco al fine di generare una griglia di ancore sovrapposta all’immagine. Per ciascun ancora presente la rete restituisce 5 valori: x , y , w , h , confidence, C (come in figura 2.7). Il centro della bounding box (x , y) rispetto ai bordi della cella. I valori di larghezza (w) e altezza (h) sono espressi relativi all’immagine. Infine la rete aggiunge un valore di confidenza della bounding box. La confidenza è definita come $\text{Prob}(\text{Object}) * \text{IOU}$. Nel caso peggiore, la confidenza è pari a 0, nel caso migliore la confidenza è pari all’intersection over union tra la bounding box predetta e la ground truth. Infine, a ciascun blocchetto è associato un vettore di probabilità C relativa alle

classi su cui si fa inferenza. Il layer FC in coda al modello (si veda figura 2.6), è un layer di dimensione $S \times S \times (B \times 5 + C_l)$, dove: S è il numero di celle in cui si suddividono le righe (e le colonne) dell'immagine; C_l è il numero di classi presenti; B è il numero di bounding box predette per ciascuna cella.

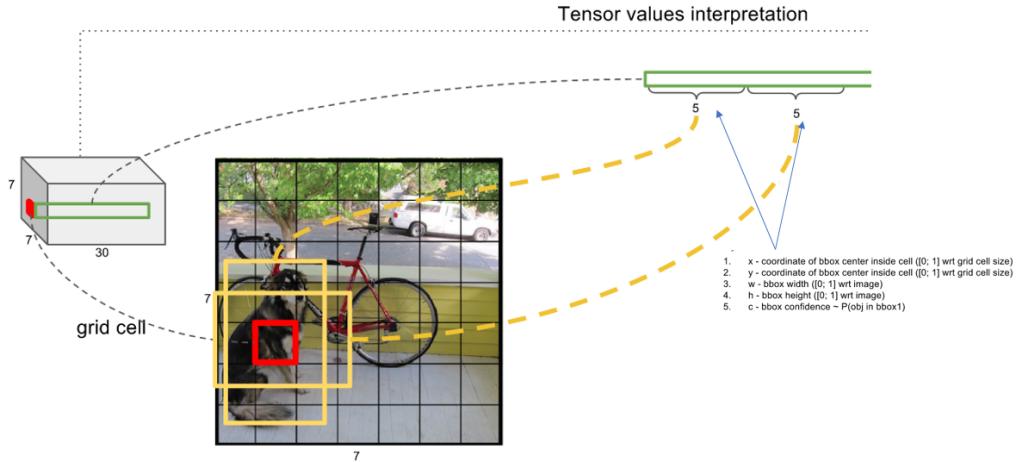


Figura 2.7: Interpretazione della risposta del modello YOLO

2.8.3 Non-maximal suppression

Per ridurre il numero di bounding box predette, YOLO procede sogliando le predizioni con confidenza C minore. In particolare, si procede ordinando le bounding box predette per C decrescente. Successivamente, si procede verificando ciascuna predizione: una bounding box b viene ignorata nel risultato finale se esiste almeno una bounding box b' tra quelle già incontrate tale che $\text{IOU}(b, b') > 0.5$.

Il modello associa alla cella la bounding box (tra le B predette) con C maggiore. In figura 2.8 vengono riportati i passi di esecuzione della pipeline YOLO con in input un'immagine d'esempio.

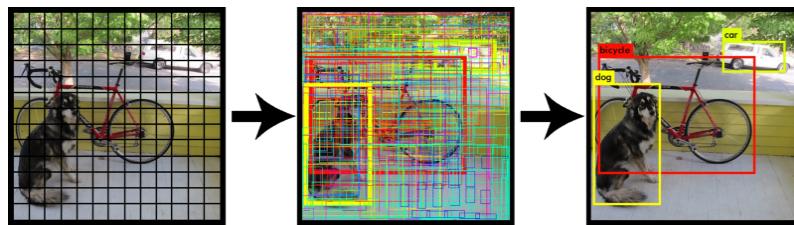


Figura 2.8: Esempio pipeline YOLO

Capitolo 3

The KITTI Vision Benchmark Suite

3.1 Are we ready for Autonomous Driving?

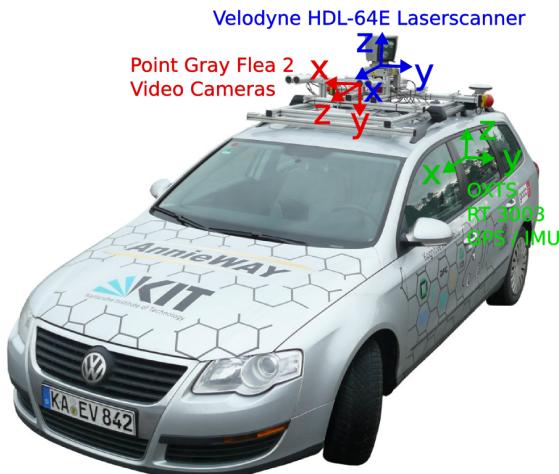


Figura 3.1: LIDAR e videocamere montate

Il Dataset Kitti[5] è un dataset open source che mette a disposizione di ricercatori e università dati RGB (acquisiti con 4 camera ad alta risoluzione) e dati LIDAR (acquisiti con un laser scanner Velodyne). Il progetto è nato come collaborazione A project del Karlsruhe Institute of Technology (Germania) e del Toyota Technological Institute (Chicago, US). Sono presenti fino a 15 automobili e 30 pedoni per immagine. I risultati degli algoritmi allo stato dell'arte sul Kitti hanno dimostrato la necessità, sia di enti pubblici sia di aziende private, di implementare nuove tecnologie al fine di ottenere una tecnologia

spendibile nel mondo reale (ossia non limitata ad esperimenti di laboratorio). Nelle pagine sequenti si presentano tecniche e tool con cui affrontare la sfida proposta dal Kitti Benchmark, in particolare affinando lo stato dell’arte delle CNN e introducendo il flusso di dati LIDAR come informazione di input. Le camere sono montate a livello dell’asfalto, e producono immagini di 1382 x 512 pixel. Telecamere e laser scanner sono sincronizzati e acquisiscono a 10 frame per secondo. I valori di calibrazione degli strumenti sono valutati all’inizio di ciascuna giornata di ripresa, e indicano nei particolari le distanze e le pose degli strumenti.

3.2 Setup

3.2.1 Laser scanner

Il laser scanner adottato da Kitti per l’acquisizione dei dati LIDAR è un HDL-64E di Velodyne (esempio del strumento in figura 3.3). Il datasheet dello strumento mette in luce la qualità di questa tecnologia [2]. Si tratta di uno strumento TOF (time-of-flight) allo stato dell’arte, capace di ottenere dati precisi in ambiente non controllato e in movimento. I dati 3D LiDAR generati contengono le misurazione delle distanze da ciascun punto, calcolate come il tempo necessario alla luce di raggiungere l’oggetto ed essere acquisiti dal sensore sotto-forma di eco. Specifiche:

- 64 piani spaziali
- Range massimo di 120m
- Fino a 2.2 milioni di punti per secondo
- Acquisizione a 360°
- Frequenza di rotazione dai 5 ai 20 Hz per secondo



Figura 3.2: HDL-64E di Velodyne

3.2.2 Camere

L’automobile è stata equipaggiata da 2 video camere RGB (sinistra e destra) e 2 video camere in scala di grigi (sinistra e destra) PointGrey Flea2. Le telecamere hanno ripreso con

frequenza di 10 Hz sincrone con il Laser Scanner. La dimensione delle immagini acquisite è pari a 1392x512 pixel. Le telecamere sinistre sono state montate a una distanza di 54 cm dalle destre . Per il progetto di tesi sono state utilizzate le immagini acquisite dalla sola camera sinistra RGB: le camere RGB è difatti utile in task di riconoscimento e classificazione di immagini, a scapito di un minore contrasto rispetto alle camera in scala di grigi.



Figura 3.3: Camera PointGrey Flea2

3.2.3 Altra strumentistica

- GPS/IMU: unità di localizzazione GPS, con errore di localizzazione inferiore ai 5 cm
- Computer di bordo

3.3 Caratteristiche

Sono stati impiegati i dati LIDAR e RGB relativi al task Kitti *Object Detection Evaluation 2012*. Sono a disposizione 7480 frame ripresi in condizioni ambientali differenti. Tra le classi 7 di oggetti presenti, ne sono state scelte tre tra loro ben distinte e distinguibili: Auto, pedoni, ciclisti. Gli oggetti si presentano nell'immagini con diversi gradi di occlusione¹ e troncamento². E' stata adottata la partizione di dataset proposta da [7], che prevede 3712 frame destinate all'apprendimento e 3768 immagini di validation dei risultati.

3.4 Difficoltà: Easy, Moderate, Hard

Per la valutazione dei risultati si è adottato il metodo di valutazione proposta da Kitti. Questo prevede una misurazione delle performance di object detection secondo il criterio PASCAL[4]. Kitti impone un'intersezione tra bounding box dell'oggetto e ground truth maggiore del 70 % per le automobili, maggiore del 50 % per ciclisti e pedoni. Le tre classi

¹Con *occlusione* si intende la porzione di visibilità dell'oggetto all'interno della scena. Il valore di occlusione va da un minimo di 1 (completamente occluso) a 4 (completamente visibile).

²Con *troncamento* si intende le percentuale dell'oggetto troncata dall'acquisizione . Il valore di troncamento è espresso in percentuale, da 100% (completamente troncato) a 0% (completamente presente).

di difficoltà proposte da Kitti sono distinte in base all'altezza della bounding box, al livello di occlusione e al valore di troncamento. Definizione:

- Easy: Altezza Minima: 40 Px, Massima occlusione: completamente visibile, Massimo troncamento: 15 %
- Moderate: Altezza Minima: 25 Px, Massima occlusione: parzialmente occluso , Massimo troncamento: 30 %
- Hard: Altezza Minima: 25 Px, Massima occlusione: Difficile da vedere, Massimo troncamento: 50 %

Kitti evidenzia che 2% degli oggetti appartenenti alla categoria Hard non sono stati correttamente individuati dall'occhio umano: il valore massimo di riconoscimento per la categoria Hard è quindi 98%. I falsi positivi sono assegnati alla categoria Easy quando l'altezza della loro bounding box ricade nel range 25-39 px, e a tutte le categorie quando l'altezza è minore di 25 px.

3.5 Metriche

3.5.1 Precision, Recall e Fscore

Sono state adottate tre metriche per validare i risultati ottenuti dal modello. I tre valori sono calcolati sul dataset di validation al termine di ogni epoca, al fine di mostrare il corso dell'apprendimento. La rete neurale adottata (2.8) risponde con una lista di oggetti predetti. Si considera una predizione (bounding box e classificazione dell'oggetto) corretta se: l'intersection over union è tra bounding box dell'oggetto e ground truth è maggiore di 50% e la sua classe è corretta. Si definiscono successivamente le tre metriche sono: precision, recall 3.4 e fscore. Il valore di recall indica il rapporto tra il numero di risposte corrette e il numero di oggetti effettivamente presenti.

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} \cup \text{false negatives}} \quad (3.1)$$

Il valore di precision indica il rapporto tra il numero di risposte corrette e il totale delle proposte date dal modello.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} \cup \text{false positives}} \quad (3.2)$$

Il valore di fascore è la media armonizzata delle due metriche precedenti, e coincide con la radice quadrata del rapporto tra la loro media geometrica e la loro media aritmetica. E' un valore riepilogativo.

$$\text{fscore} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{fscore}} \quad (3.3)$$

3.5.2 Average Precision

Il valore di Average Precision (AP), definito in [4], permette di misurare le performance del modello nel task di detection e recognition di una specifica classe di oggetti. Vengono calcolate le metriche di precision e recall del modello rispetto alla classe esame, utilizzando i frame del dataset di validation. Successivamente, si calcola la metrica tramite due passaggi:

- Si definisce la curva di precision / recall $p/r(r)$ ponendo per ogni valore di recall r il massimo valore di precision ottenuto per ogni valore di $r' \geq r$

$$p/r(r) = \max(p(r') : r' \geq r) \quad (3.4)$$

- Si calcola la metrica AP come l'area sottostante la curva definita in 3.4 mediante integrazione numerica

3.5.3 Mean Average Precision

La metrica Mean Average Precision (mAP) è un indicatore globale delle performance di detection e recognition del modello. Si ottiene come media aritmetica dei valori di AP calcolati per ogni classe

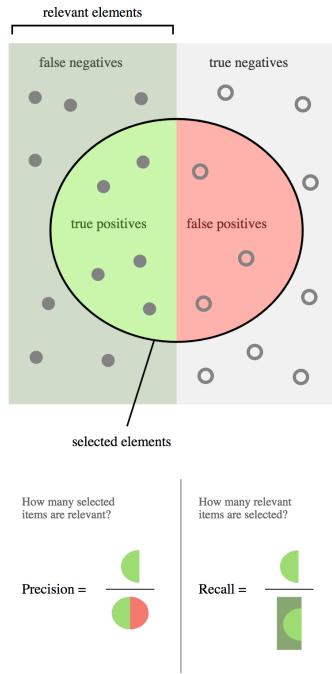


Figura 3.4: Definizione di Precision e Recall [1]

Capitolo 4

Esperimenti

Usando l'implementazione pytorch di una versione recente del modello introdotto 2, sono stati svolti una serie di esperimenti su immagini RGB, Depth e RGBD. Si è scelto di optare per un resize dei frame a 1280x416 pixels, rispettando i vincoli di YOLO sulle dimensioni delle immagini di input. Ulteriori modifiche significative all'implementazione sono riportate nei relativi esperimenti. I modelli di inferenza ottenuti sono stati quindi valutati adottando le metriche di 3. Le classi di oggetti, suddivise come proposto da Kitti in 3 categorie di difficoltà, sono valutate secondo l'Average Precision. Si riportano inoltre i valori di precision, recall, fscore per gli esperimenti più significativi.



Figura 4.1: Esempio di frame risultato. Il modello restituisce le bounding box e la classe degli oggetti presenti

4.1 RGB

La prima fase di esperimenti è stata incentrata nel valutare le performance del modello con immagini RGB. Si è innanzitutto valutato il modello preaddestrato, dal quale si è proceduto per ottenere performance complessive migliori.

4.1.1 Modello preaddestrato su dataset COCO

Il primo esperimento ha avuto il fine di valutare le prestazioni dell’architettura YOLO con pesi pre-addestrati. I pesi a disposizione sono stati ricavati da una fase di apprendimento di 6800 epoche sul dataset COCO. I risultati ottenuti (tabella 4.1) mettono in luce una differenza tra COCO e Kitti: la classe ciclisti. Difatti, come si evince dall’esempio in figura 4.2, COCO non prevede una classe per "persone che guidano una bici". Le due classi (persone e bici) sono considerate separatamente; in fase di validation i risultati che si ottengono sono quindi insoddisfacenti nella classe ciclisti.

AP	Easy %	Moderate %	Hard %
Car	46.98	44.19	27.71
Cyclist	0.00	0.00	0.00
Person	49.81	36.70	30.12

Tabella 4.1: AP per le 3 classi di inferenza e le 3 difficoltà del modello YOLO preaddestrato



Figura 4.2: Esempio segmentazione tratto dal dataset COCO. La classe Bici e persona sono entità separate

4.1.2 Modello addestrato su Kitti

L’esperimento seguendo è stato svolto al fine di sopperire alla mancanza della classe ciclisti. In particolare, si è addestrato il modello YOLO sul dataset di training descritto in 3.3 per 180 epoche. I risultati ottenuti sono riportati nella tabella 4.2.

AP	Easy %	Moderate %	Hard %
Car	14.06	14.13	9.57
Cyclist	6.21	4.30	1.41
Person	17.97	14.58	12.21

Tabella 4.2: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB

Il numero di epoche dedicate alla fase di apprendimento è stato insufficiente al fine di ottenere buoni risultati di performance globali. Tuttavia, il modello ha appreso, seppur in quantità minima, informazioni riguardanti la classe ciclisti.

4.1.3 Fine-tuning del modello con pesi COCO su dataset Kitti

L'esperimento numero 3 ha seguito l'approccio fine-tuning (si veda appendice A.8). Il modello con pesi pre-addestrati sul dataset COCO è stato successivamente addestrato sul dataset Kitti per 60 epoche. I risultati riportati in tabella 4.3 mettono in luce le differenze tra i due dataset. Le prestazioni globali sono scarse rispetto al modello di partenza. Unica nota positiva è l'apprendimento, seppur minimo, di informazioni sulla classe ciclisti.

AP	Easy %	Moderate %	Hard %
Car	9.00	10.00	6.94
Cyclist	2.00	1.00	0.00
Person	39.71	29.36	24.75

Tabella 4.3: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB usando pesi pre-addestrati COCO 60 epoche

4.1.4 Fine-tuning del modello con pesi pre-addestrati VOC su dataset Kitti con anchors COCO

Con l'esperimento 4 si è scelto un approccio differente: si sono adottati i pesi pre-addestrati su dataset VOC PASCAL, non dissimile dal dataset Kitti. Si è scelto di impiegare dapprima le anchors proposte da YOLO per il dataset COCO. L'apprendimento è stato di 160 epoche. Dai risultati in tabella 4.4 si evincono buone prestazioni: l'AP dei ciclisti, in particolare nella categoria Easy ove le bounding box sono di dimensioni maggiori, raggiungono i livelli per la stessa categoria delle restanti classi. La Mean Average Precision si attesta a 26.78%.

AP	Easy %	Moderate %	Hard %
Car	39.00	37.00	24.00
Cyclist	33.00	7.00	3.00
Person	42.00	29.00	27.00

Tabella 4.4: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB usando pesi pre-addestrati VOC con ancore COCO per 160 epocha

4.1.5 Fine-tuning del modello con pesi pre-addestrati VOC su dataset Kitti con anchors VOC

L'esperimento 5, ultimo esperimento con dati RGB, si è invece scelto di impiegare le anchors proposte da YOLO per il dataset VOC PASCAL. Le prestazioni ottenute con un training di 70 epocha sono riportate in tabella 4.5 e sono risultate le migliori. La Mean Average Precision si attesta a 31.44%.

AP	Easy %	Moderate %	Hard %
Car	54.00	53.00	33.00
Cyclist	32.00	10.00	2.00
Person	42.00	31.00	27.00

Tabella 4.5: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGB usando pesi pre-addestrati VOC 70 con ancore VOC per 70 epocha

4.1.6 Fine-tuning modello con pesi pre-addestrati su immagini di depth

L'esperimento numero 6 è stato svolto al fine di mettere in luce le potenzialità delle immagini di Depth ottenute tramite il procedimento descritto in 1.6. Le immagini di distanza sono in scala di grigi ed è stato quindi necessario adattare l'architettura, che richiede input a 3 canali, al fine di ottenere risultati comparabili. La soluzione adottata utilizza i pesi pre-addestrati per tutti i layer meno il primo; quest'ultimo è infatti generato ex-novo (con pesi casuali) correggendo i parametri relativi all'input. L'apprendimento con le immagini di depth, che risultano carenti di particolari rispetto all'immagine acquisita con camera, hanno prodotto risultati (si veda tabella 4.6) equiparabili all'esperimento in 4.4. L'apprendimento è stato svolto per 80 epocha e sono state adottate le ancore per VOC.

AP	Easy %	Moderate %	Hard %
Car	43.84	41.11	29.72
Cyclist	20.64	3.66	3.41
Person	37.75	27.01	24.65

Tabella 4.6: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite Depth usando pesi pre-addestrati del EXP5 per 80 epochhe

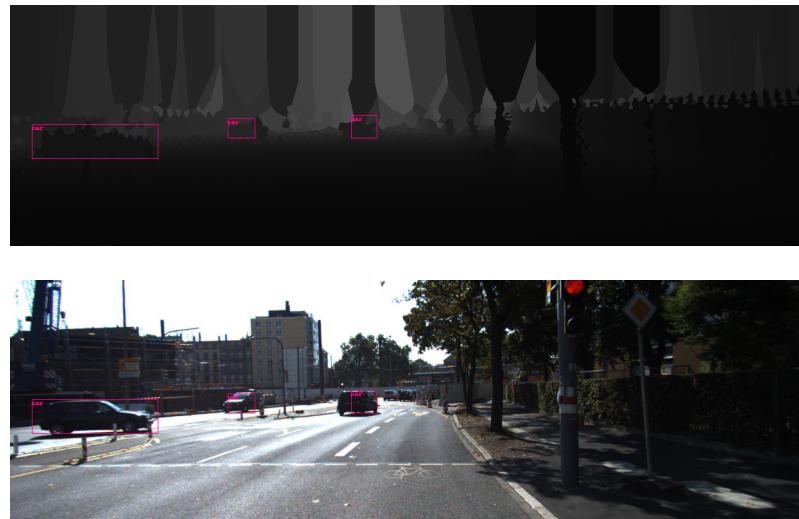


Figura 4.3: Label predette tramite il modello dell'EXP 6 su immagine di Depth (viene qui riportata anche l'immagine RGB per chiarezza)

4.2 RGBD

L’ultima serie di esperimenti è stata svolto allo scopo di impiegare le informazioni colori (RGB) e di distanza (Depth). E’ stato necessario adattare il modello al fine di ricevere come input frame a 4 canali: la soluzione è stata la stessa proposta in 4.1.6. Per tutti gli esperimenti sono stati adottati i pesi pre-addestrati ottenuti dall’esperimento 5. Quest’ultimo ha difatti garantito i risultati migliori nella sezione RGB. Il modello è stato successivamente addestrato usando ancora VOC: i parametri di anchors proposti da YOLO per il dataset VOC-PASCAL sono difatti risultate le migliori (si confrontino esperimenti 4 e 5). Il tempo di addestramento è stato di 60 (tabella 4.6), 100 (tabella 4.7) e 140 epoche (tabella 4.8). Dal confronto delle tabelle mette in luce il margine di miglioramento all’aumentare delle epoche dedicate all’apprendimento. Si può quindi supporre che una fase di apprendimento più lunga, e quindi un numero maggiore di epoche, o un dataset di dimensioni maggiore potrebbero portare a performance migliori. L’esperimento 9 RGBD non è stato difatti all’altezza della controparte RGB (esperimento 5). Ciò è probabilmente da imputare a immagini di depth eccessivamente rumorose. Con un numero di epoche maggiore la rete neurale potrebbe apprendere se e quando impiegare l’informazioni del canale di depth ed egualare i valori dell’esperimento RGB.

AP	Easy %	Moderate %	Hard %
Car	21.72	20.73	15.10
Cyclist	7.32	1.02	1.49
Person	6.85	4.70	4.80

Tabella 4.7: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGBD usando pesi pre-addestrati del EXP5 per 60 epoche

AP	Easy %	Moderate %	Hard %
Car	38.54	35.15	26.21
Cyclist	14.51	2.70	2.38
Person	38.88	28.14	25.86

Tabella 4.8: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGBD usando pesi pre-addestrati del EXP5 per 100 epoche

AP	Easy %	Moderate %	Hard %
Car	39.44	37.09	26.39
Cyclist	17.99	4.57	3.39
Person	35.93	26.01	23.72

Tabella 4.9: AP per le 3 classi di inferenza e le 3 difficoltà della rete Yolo addestrata sul Kitti Benchmark Suite RGBD usando pesi pre-addestrati del EXP5 per 140 epoche

4.3 Curva Precision/Recall

La curva di precision-recall mette in luce, per diversi valori di threshold di confidenza, il rapporto tra i valori di precision e di recall del modello. Il threshold è fatto variare da 0.01 a 0.99. A valori minori di thr corrisponde una maggiore precision, a scapito di una minore recall. Il modello, per threshold bassi, restituisce meno risultati ma assi più accurati. Si ha quindi la possibilità, laddove certi valori di precision e recall fossero richiesti per lo svolgimento del task, di determinare il valore di threshold ideale per ciascun esperimento.

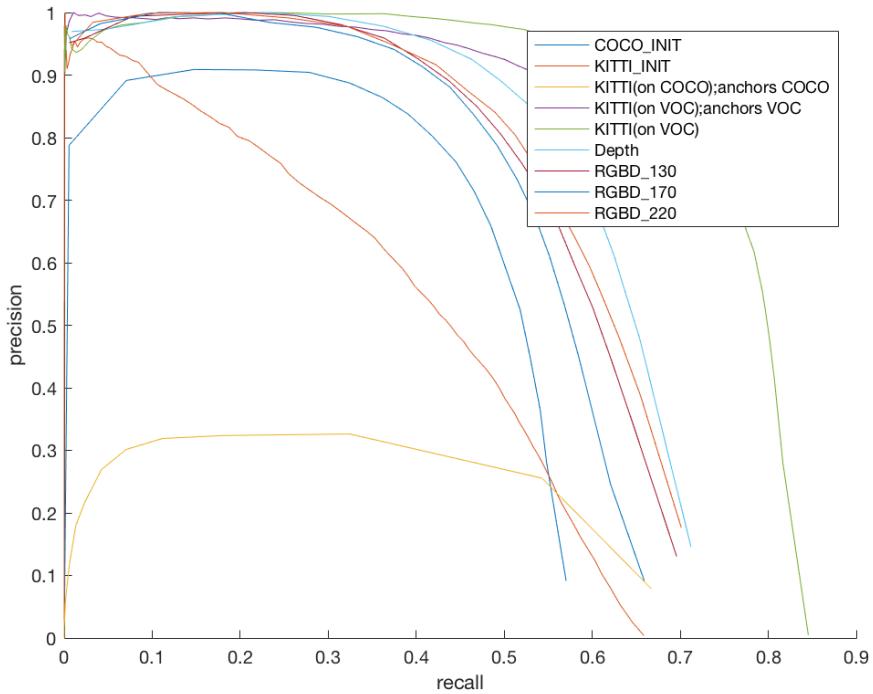


Figura 4.4: Curve precision/recall degli esperimenti fatti

4.4 Riconoscimento real-time di oggetti

Si sono quindi scelti un modello e un valore di threshold appropriato al fine di mostrare un'applicazione concreta di quanto prodotto. La scelta è ricaduta sul modello in esame nell'esperimento 5, mentre il threshold, dall'analisi del grafico di KITTI(on VOC) in figura 4.4, è stato impostato a 0.5. Il modello ha registrato un tempo di inferenza tra gli 0.04 secondi (25 frame/sec) e i 0.03 secondi (30 frame/sec), e può quindi lavorare in tempo reale durante la ripresa, se questa si mantiene sui 25 fps. Il modello è stato utilizzato per fare inferenza sui frame di un video registrato in ambiente non completamente controllate (all'aperto e con cielo sereno, in ambiente che presenta ombre e poca omogeneità). Il video è stato registrato con un semplice smartphone e non è stato sottoposto ad alcun tipo di

preprocessing. Da alcuni frame estratti si evince che il threshold impiegato è un buon compromesso tra la precision (sia nell’assegnamento delle label sia nel posizionamento del bounding box) e recall. Nei frame 4.5 non tutti i pedoni sono riconosciuti: le persone troppo lontane dalla camera e le automobili anche parzialmente occluse non sono state riconosciute. Nei frame 4.6 buona parte dei pedoni in ombra (sulla sinistra) non sono stati correttamente riconosciuti, mentre le automobili (sulla destra) sono riconosciute solo in parte. Un limite che qui evidente dell’architettura YOLO è la discriminazione di oggetti in zone dell’immagine in cui questi sono densamente presenti (ad esempio, una folla di persone o una fila di automobili).

	AP %			Car			Cyclist			Person	
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard		
exp1	46.98	44.19	27.71	0.00	0.00	0.00	49.81	36.70	30.50		
exp2	14.06	14.13	9.57	6.21	4.30	1.41	17.97	14.58	12.21		
exp3	9.00	10.00	6.94	2.00	1.00	0.00	39.71	29.36	24.75		
exp4	39.00	37.00	24.00	33.00	7.00	3.00	42.00	29.00	27.00		
exp5	54.00	53.00	33.00	32.00	10.00	2.00	42.00	31.00	27.00		
exp6	43.84	41.11	29.72	20.64	3.66	3.41	37.75	27.01	24.65		
exp7	21.72	20.73	15.10	7.32	1.02	1.49	6.85	4.70	4.80		
exp8	38.54	35.15	26.31	14.51	2.70	2.38	38.88	28.14	25.86		
exp9	39.44	37.09	26.39	18.00	4.57	3.39	35.93	26.01	23.72		

Tabella 4.10: Tabella riassuntiva dell’average precision, suddivisa per classe e difficoltà, per ciascun esperimento



Figura 4.5: Label predette tramite il modello dell'EXP 5 (Milano, 2018)



Figura 4.6: Label predette tramite il modello dell'EXP 5 (Milano, 2018)

Capitolo 5

Conclusione e considerazioni finali

Lo scopo del lavoro svolto è stato impiegare il corrente stato dell'arte nelle reti neurali convoluzionali per l'object detection (si veda 2.8) al fine di creare un modello capace di individuare e riconoscere pedoni, ciclisti e automobili in condizioni non controllate da un flusso di dati RGBD. La scelta di CNN è ricaduta sul modello YOLO [8]. Il dataset The Kitti Vision Benchmark Suite mette a disposizione, per ciascuna scena ripresa, immagini RGB con i relativi dati LIDAR grezzi. La scelta seguita in questo progetto di tesi ha portato alla generazione di un dataset di immagini di depth tratte da scene reali. I dati LIDAR del dataset sono stati acquisiti tramite laser scanner, i quali sono utili strumenti con cui è possibile ottenere informazioni di profondità della scena. I dati sono contenuti in strutture dati grezze dette point-cloud.

Sono stati quindi effettuati 9 esperimenti su immagini RGB, Depth e RGBD. I modelli addestrati sono stati valutati con la metrica di Average Precision proposta da VOC PASCAL. Le 3 classi di oggetto sono state suddivise in 3 categorie di difficoltà secondo la dimensione delle bounding box e il loro grado di occlusione e troncamento. La suddivisione si è rivelata robusta: la categoria Easy è stata negli esperimenti quella con le performance migliori. In particolare per la classe bici, che presenta bounding box di dimensioni ridotte, la differenza di AP tra Easy e Hard è stata del 14% per l'esperimento 9 RGBD e di oltre il 20% nell'esperimento 5 RGB.

Per le immagini RGBD (RGB + depth), la modifica proposta in 4.2 è stata necessaria al fine di adattare la rete neurale YOLO, la quale è sviluppata per immagini RGB, a ricevere input con 4 canali. Il primo layer, nel caso degli esperimenti Depth e RGBD, è stato interamente sostituito da un layer con kernel della profondità corretta; tale sostituzione ha permesso di mantenere i pesi pre-addestrati su immagini a 3 canali, velocizzando i tempi di addestramento. L'impiego di YOLO ha mostrato un peggioramento quando alle immagini RGB è stato affiancato il canale di profondità. L'esperimento su depth (tabella 4.6) è confrontabile con l'esperimento 4 RGB (tabella 4.4), mentre le performance risultano peggiori se confrontate con l'esperimento 5 RGB. Quest'ultimo, come si evince dalla tabella 4.5, è inoltre risultato migliore rispetto all'esperimento 9 RGBD. Difatti, il valore di mAP per i due esperimenti è stata rispettivamente 31.44% e 25.75%. Tuttavia, confrontando gli esperimenti 7, 8 e 9 RGBD si evince una tendenza di miglioramento delle performance

all'aumentare del numero di epoche dedicate alla fase di apprendimento.

L'immagini di depth, generate tramite la pipeline descritta in 1.6, hanno determinato un peggioramento delle performance. Questo declino delle performance può essere ricondotto alla rumorosità del canale depth. Difatti, il laser scanner acquisisce dati profondità sparsi, mentre le immagini RGB sono dense in ogni loro punto. Al fine di ottenere immagini di depth dense è stato necessario interpolare i valori laddove quest'ultimi erano assenti. Le immagini ottenute risultano rumorose e con pochi dettagli, in contrapposizione con un'immagine RGB. Sviluppi futuri in merito potrebbero quindi riguardare l'acquisizione di immagini di depth frontali con valori densi, al fine di sfruttare il corrente stato dell'arte delle CNN per inferire direttamente su immagini delle profondità. In alternativa, un approccio software potrebbe consistere nell'affinare la pipeline proposta in 1.6 introducendo una fase di riduzione del rumore (tramite algoritmo o con l'ausilio di CNN appositamente addestrate).

Appendice A

Appendice

Di seguito sono brevemente spiegati alcuni dei concetti fondamentali per l'interpretazione del testo e dei risultati ottenuti.

A.1 Point Cloud

Una point-cloud è una struttura dati adottata dai sistemi di acquisizione tridimensionali per il salvataggio di una mappa 3D. Viene fisicamente implementata come una matrice nx4, dove:

- n : numero di punti dello spazio, ossia numero delle acquisizioni fatte dallo strumento
- canale 1 : posizione del punto rispetto all'asse X dello strumento
- canale 2 : posizione del punto rispetto all'asse Y dello strumento
- canale 3 : posizione del punto rispetto all'asse Z dello strumento
- canale 4 : valore di luminanza (ossia, di intensità luminosa) acquisita nel punto XYZ

A.2 RGB

Il modello colore RGB è un modello additivo che permette la codifica spaziale dell'informazione cromatica di un'immagine. Lo spazio discretizzato è rappresentato dai pixel in cui l'immagine è suddivisa, mentre il colore è suddiviso su tre canali: R (Red), G (Green) e B (Blue). I tre canali corrispondono ai tre colori additivi primari. Ciascun pixel dell'immagine ha quindi associato un valore per ognuno dei canali cromatici. Il valore cromatico in output (ad esempio, in fase di visualizzazione o stampa) è ottenuto dalla somma pesata dei tre colori primari.

A.3 RGBD

Il modello RGBD è un'estensione a 4 canali del modello standard RGB (Red-Green-Blue). E' adottato per introdurre nell'immagine un quarto canale D, ossia un valore di Depth (profondità o distanza) del punto nell'immagine. Nell'immagine d'esempio A.1 è riportato affianco il rapporto colore/distanza del soggetto. La distanza del pedone dallo strumento di acquisizione è di circa 15 metri.

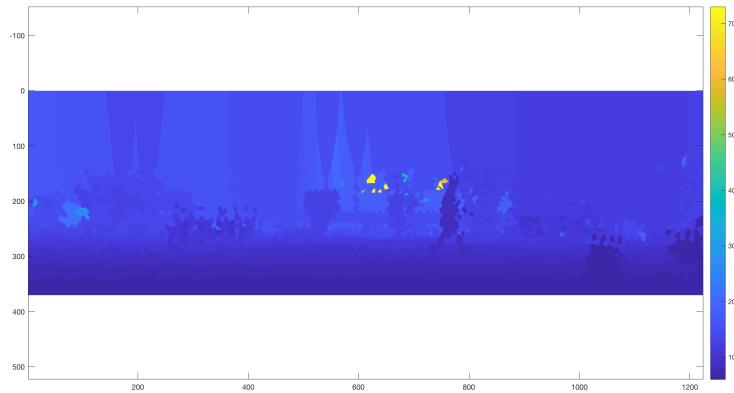


Figura A.1: Immagine di Depth tratta da Kitti Dataset; il canale d'intensità è scalato sull'intera colormap riportata sulla destra

L'immagine di Depth soffre tuttavia di poca nitidezza nei dettagli e nelle forme, ed è priva di informazioni relative al colore degli oggetti presenti. L'idea di comporre immagine Depth e RGB è quindi una scelta giustificabile.



Figura A.2: Immagine RGB della stessa scena

A.4 Pipeline di conversione: esempi di codice

A.4.1 Step 01

```
1      % compute projection matrix velodyne->image plane
2      R_cam_to_rect = eye(4);
3      [P, Tr_velo_to_cam, R] = readCalibration(calibpath,
4          n_frame, ...
5          cam);
6      R_cam_to_rect(1:3,1:3) = R;
```

Listing A.1: Prima fase di conversione

A.4.2 Step 02

```
1      [~, filename_in, ~] = fileparts(filelist(i).name);
2      %filename_out = sprintf('%06s.mat', counter);
3
4      % Load depth and rgb
5      load(fullfile(inpath, filelist(i).name));
6
7      velo_points = depth;
8      P = P_velo_to_img;
9
10     % load velodyne points
11     % remove every 5th point for display speed
12     velo_points = velo_points(1:5:end,:);
13
14     % remove all points behind image plane (approximation
15     idx = velo_points(:,1)<5;
16     velo_points(idx,:) = [];
17
18     %velo_points_2D = project(velo(:,1:3), P_velo_to_img);
19     velo_points_2D = projectToImage(velo_points(:, 1:3)', P)';
20     % project to image plane (exclude luminance)
21     to_keep = velo_points_2D(:,1)>=1 & ...
22         velo_points_2D(:,1)<=size(rgb,2) & velo_points_2D(:,2)>=1
23         & ...
24         velo_points_2D(:,2)<=size(rgb,1);
25     velo_points_2D = round(velo_points_2D(to_keep,:));
26     velo_points = round(velo_points(to_keep,:));
```

Listing A.2: Seconda fase di conversione

A.4.3 Read Calibration

```

1 function [P, Tr_velo_to_cam, R_cam_to_rect] =
2     readCalibration(calib_dir, img_idx, cam)
3
4 % load 3x4 projection matrix
5 P = dlmread(sprintf('%s/%06d.txt', calib_dir, img_idx), ' ', 0, 1);
6
7 Tr_velo_to_cam = P(6,:);
8 R_cam_to_rect = P(5,1:9);
9
10 P = P(cam+1,:);
11 P = reshape(P, [4,3])';
12 R_cam_to_rect = reshape(R_cam_to_rect, [3,3])';
13 Tr_velo_to_cam = reshape(Tr_velo_to_cam, [4,3])';
14 Tr_velo_to_cam = [Tr_velo_to_cam; 0 0 0 1];
15 end

```

Listing A.3: Lettura della calibrazione dello strumento

A.5 K-d tree

Un kd-tree è una struttura dati organizzata per punti appartenenti a un spazio k dimensionale. Questo tipo di struttura dati è ottimizzato per la ricerca di vicini in più dimensioni ed è spesso utilizzato per task di nearest neighbour e k - nearest neighbour. In particolare il procedimento di ricerca è il seguente: Dato un punto di cui individuare il minimo, l'albero viene visitato confrontando il valore nella dimensione del nodo corrente con il valore della stessa deminsione del punto al fine di scegliere la direzione di discesa (sinistra se minore-uguale, destra se maggiore). Quando la visita raggiunge un nodo foglia, questo viene indicato come "current best" e la sua distanza dal punto dato diventa la minima distanza. Successivamente, l'algoritmo procede ricorsivamente verso la radice. Per ogni nodo coinvolto nel processo ricorsivo, se questo è più vicino al punto dato rispetto al punto di "current best", il nodo diventa il nuovo punto di "current best" e la distanza minima viene aggiornata. Quindi, l'algoritmo verifica se punti appartenenti all'altro ramo dell'albero potrebbero avere distanza minore della minima distanza corrente. A tale scopo, si verifica se la distanza tra il nodo corrente e il punto dato è minore rispetto alla distanza minima. L'algoritmo si conclude quando il nodo radice raggiunto: il punto di "current best" è l'effettivo punto di minima distanza. Generalmente viene usata come misura di distanza il modulo al quadrato che del vettore che congiunge i due punti. In questo modo non è necessario calcolare la radice quadrata, e la computazione risulta più efficiente. L'algoritmo può essere esteso al caso k-nearest neighbour mantenendo, in fase di ricerca, i k punti vicini (nel caso base si considera quindi k = 1) .

A.6 Back propagation

Back propagation è un metodo numerico per il calcolo del gradiente di una funzione differenziabile che si appoggia sulla regola di derivate a catena. Fondamentale ricordare che la derivata di una funzione rispetto a una variabile x_i rappresenta la sensitività della funzione rispetto al valore di quella specifica variabile. Data una funzione banale quale

$$f(x, y) = xy \quad (\text{A.1})$$

è semplice determinare l'espressione analitica delle derivate parziali rispetto agli assi x e y. L'introduzione di una complessità maggiore, come nell'equazione

$$f(x, y, z) = (x + y)z = qz \quad (\text{A.2})$$

è semplificabile ponendo $q(x, y) = (x + y)$ e calcolando la derivata per $f(q, z)$, seguita dalla derivata di $q(x, y)$. La regola delle derivate a catena è una formula analitica che lega le variabili di una funzione rispetto alla derivata parziale della funzione stessa. In particolare si può dimostrare che:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \quad (\text{A.3})$$

ossia, il valore della derivata parziale della funzione f rispetto alla variabile x è pari al valore della derivata parziale rispetto a q moltiplicato per la derivata parziale di q rispetto a x. Si può quindi procedere risolvendo la funzione f rispetto a una tripla (x, y, z), e procedendo successivamente a ritroso al fine di calcolare il peso di ciascuna variabile sul valore finale della funzione. Nella figura A.3 si esprime la funzione di cui valutare il gradiente con l'uso

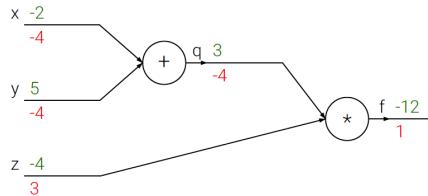


Figura A.3: Esempio tratto da Stanford CS231n

di nodi elementari (detti anche gate) per i quali la derivata analitica è nota. In particolare:

- Gate sommatoria(x, y): per il quale $\frac{\partial f}{\partial x} = 1$ e $\frac{\partial f}{\partial y} = 1$
- Gate moltiplicativo(x, y): per il quale $\frac{\partial f}{\partial x} = y$ e $\frac{\partial f}{\partial y} = x$

Si possono costruire gate di funzioni più complesse (e.g. max, loss, convoluzioni) semplicemente valutando l'equazione analitica della derivata del gate rispetto a ciascun input. Framework di reti neurali (e.g. pytorch, tensorflow) mettono a disposizione dello sviluppatore gate per il quali l'equazione delle derivate è già stata valutata.

A.7 Stochastic Gradient Descent

Stochastic gradient descent (spesso anche SGD) è un algoritmo di approssimazione stocastica che sfrutta il percorso di discesa del gradiente di una funzione differenziabile al fine di individuare iterativamente il massimo (o il minimo) della funzione. Difatti, il gradiente (estensione al campo R^n del concetto di derivata) è un vettore che esprime la direzione di massima crescita della funzione in un punto. L'algoritmo è riassumibile come:

$$\hat{w} = w - \alpha \nabla L(w) \quad (\text{A.4})$$

Iterando l'algoritmo sul set di dati di training, si raffina l'approssimazione di \hat{w} quale minimo della funzione. Difatti, al vettore corrente dei pesi w è sottratto il gradiente funzione da minimizzare L_i all'istante i . Un valore α , detto learning rate, modula la dimensione di ciascun passo iterativo ed è un importante valore del processo di training. Si noti che il calcolo del gradiente $\nabla L(w)$ è un compito non banale che trova un'elegante soluzione nel metodo di back-propagation.

A.8 Fine tuning

La dimensione del dataset utilizzato per effettuare il training è una caratteristica fondamentale del processo di apprendimento. Una rete neurale convoluzionale può avere milioni di parametri (pesi) da aggiornare tramite SGD (appendice A.7) dopo ogni iterazione. Effettuare il training di un numero elevato di pesi con un dataset ridotto influenza la capacità della CNN di generalizzare le informazioni presenti e può portare all'overfitting. Dataset quali COCO (>220.000 immagini con label) sono stati acquisiti col preciso intento di fornire un quantitativo significativo di immagini RGB. Il costo di acquisizione e di labeling delle immagini può essere significativo. Il dataset The Kitti Vision Benchmark Suite mette a disposizione un numero limitato di frame RGB (7.480 frame), ma offre immagini acquisite su strada con diverse condizioni ambientali e, per ciascuna, point-cloud da oltre 1.000.000 di punti. Al fine di sfruttare sia dataset di grandi dimensioni sia dataset ridotti e specifici per un determinato task, si effettua il fine-tuning. Una rete neurale convoluzionale può essere addestrata su un dataset generico, come COCO o VOC PASCAL. Successivamente, si interrompe la fase di apprendimento e si sostituisce il dataset di training con un secondo dataset più ridotto. Il parametro learning-rate, che determina la velocità di apprendimento dell'algoritmo SGD, viene ridotto di 10 o 100 volte: i pesi pre-addestrati sono già stati generati e non devono essere distorti eccessivamente. Un'alternativa consiste nell'inizializzare i pesi di uno o più layer (i primi o gli ultimi) dopo la prima fase di apprendimento. Durante la seconda fase di apprendimento, l'apprendimento risulterà particolarmente efficace per i layer di cui si vogliono rivalutare i pesi.

Bibliografia

- [1] *By Walber [CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0>)] from Wikimedia Commons.*
- [2] *Datasheet HDL-64E - Velodyne (<http://velodynelidar.com/hdl-64e.html>).*
- [3] Vincent Dumoulin e Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. eprint: [1603.07285](https://arxiv.org/abs/1603.07285).
- [4] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [5] Andreas Geiger, Philip Lenz e Raquel Urtasun. «Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite». In: (2012).
- [6] Jonathan Hui. *real-time-object-detection-with-yolo-yolov2*. URL: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.
- [7] Jungwook Lee Ali Harakeh Steven Waslander Jason Ku Melissa Mozifian. «Joint 3D Proposal Generation and Object Detection from View Aggregation». In: *arXiv* (2017). eprint: [1712.0229](https://arxiv.org/abs/1712.0229).
- [8] Joseph Redmon e Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». In: *arXiv* (2016).
- [9] Stanford. *CS231n Stanford*. URL: <http://cs231n.stanford.edu>.
- [10] Ji Wan Bo Li Tian Xia Xiaozhi Chen Huimin Ma. *Multi-View 3D Object Detection Network for Autonomous Driving*. 2017. eprint: [1611.07759](https://arxiv.org/abs/1611.07759).