ALMA MATER STUDIORUM · UNIVERSITA' DI BOLOGNA

Scuola di Ingegneria e Architettura

Corso di laurea in Ingegneria Elettronica e Telecomunicazioni

Metodologie di Corrispondenza Stereo Basate su Deep Learning per Superfici Altamente Riflettenti e Trasparenti: Dataset e Architettura

Tesi di Laurea in Ingegneria Elettronica e Telecomunicazioni

Relatore:

Prof. Luigi Di Stefano

Correlatori:

Pierluigi Zama Ramirez

Fabio Tosi

Presentata da:

Andrea Pumilia

1ª Sessione
Anno Accademico 2021/2022

Per aspera ad astra

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Nell'ambito della Stereo Vision, settore della Computer Vision, partendo da coppie di immagini RGB, si cerca di ricostruire la profondità della scena. La maggior parte degli algoritmi utilizzati per questo compito ipotizzano che tutte le superfici presenti nella scena siano lambertiane. Quando sono presenti superfici non lambertiane (riflettenti o trasparenti), gli algoritmi stereo esistenti sbagliano la predizione della profondità. Per risolvere questo problema, durante l'esperienza di tirocinio, si è realizzato un dataset contenente oggetti trasparenti e riflettenti che sono la base per l'allenamento della rete. Agli oggetti presenti nelle scene sono associate annotazioni 3D usate per allenare la rete. Invece, nel seguente lavoro di tesi, utilizzando l'algoritmo RAFT-Stereo [1], rete allo stato dell'arte per la stereo vision, si analizza come la rete modifica le sue prestazioni (predizione della disparità) se al suo interno viene inserito un modulo per la segmentazione semantica degli oggetti. Si introduce questo layer aggiuntivo perché, trovare la corrispondenza tra due punti appartenenti a superfici lambertiane, risulta essere molto complesso per una normale rete. Si vuole utilizzare l'informazione semantica per riconoscere questi tipi di superfici e così migliorarne la disparità. È stata scelta questa architettura neurale in quanto, durante l'esperienza di tirocinio riguardante la creazione del dataset Booster [2], è risultata la migliore su questo dataset. Si vuole indagare se la sua efficacia possa aumentare ulteriormente modificandone l'architettura cioè inserendo un modulo per la predizione di oggetti riflettenti e trasparenti contenuti in Booster. In particolare, l'obiettivo ultimo è vedere se il riconoscimento di superfici non lambertiane, da parte del modulo semantico, influenza la predizione della disparità migliorandola. Nell'ambito della stereo vision, gli elementi riflettenti e trasparenti risultano estremamente complessi da analizzare, ma restano tuttora oggetto di studio dati gli svariati settori di applicazione come la guida autonoma e la robotica.

Nel primo capitolo si introducono concetti chiave, per la comprensione di questa tesi, entrando nel dettaglio dell'algoritmo utilizzato per tutti gli esperimenti. Nel secondo capitolo si descrivono i dataset utilizzati e si elencano altri dataset trovati durante le

ricerche da tenere in considerazione come alternativa ai primi. Nel terzo capitolo si presentano gli esperimenti svolti e le relative scelte fatte in corso d'opera. Infine, nell'ultimo capitolo, si mostrano i risultati ottenuti.

Capitolo 1

Deep Learning per la Stereo Vision

In questa sezione si esaminano due macro-concetti necessari per la comprensione degli esperimenti svolti: il Deep Learning e la Stereo Vision. All'interno di quest'ultimo, si approfondisce l'algoritmo RAFT-Stereo.

1.1 Cenni di Stereo Vision

La Stereo Vision è una branca della Computer Vision. Usando due o più fotocamere è possibile simulare la visione umana binoculare e quindi stimare la profondità della scena catturata. La stima della profondità si basa sulla ricerca di corrispondenze tra l'immagine di sinistra e quella di destra. La disparità è la differenza di coordinate orizzontali tra punti corrispondenti delle due immagini. È necessario che le scene fotografate siano ricche di dettagli in modo che il sistema riconosca la stessa porzione dell'oggetto in entrambe le foto. Per il calcolo della profondità è necessario conoscere anche i parametri intrinseci delle fotocamere e la posizione relativa tra le due: la distanza focale (f), la baseline (b).

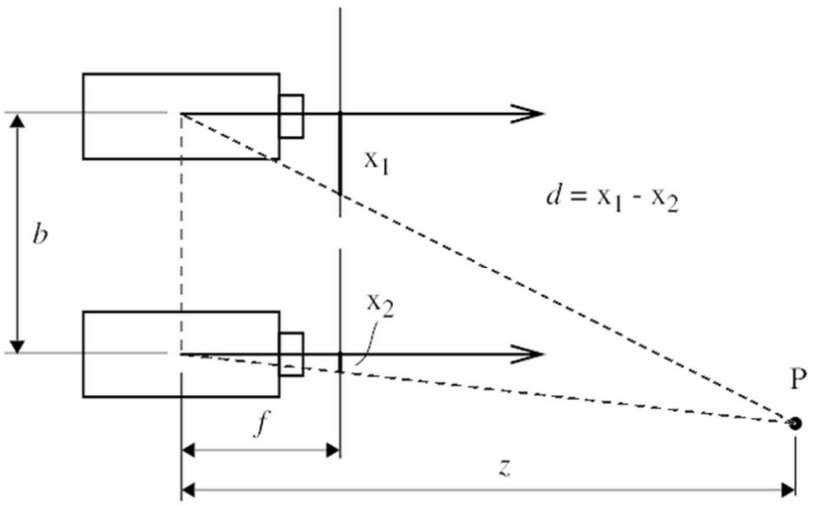

Figura 1.1: Sistema di fotocamere Stereo

A partire dalla disparità (d) si può ottenere la profondità (z) e viceversa, con la seguente formula:

$$z = f * b / d \tag{1.1}$$

1.1.1 Evoluzione della Stereo Vision

Come spiegato in [2], lo stereo matching è classificato come una tecnica di passive sensing in competizione con tecniche di active sensing quali la misurazione Time-Of-Flight (ToF), che sfrutta proiezione della luce o il Laser Imaging Detection and Ranging (LIDAR) che utilizza i laser. Lo stereo matching, in alcuni contesti, è preferibile perché non perturba l'ambiente circostante a differenza dei sensori per la stima della profondità come il LIDAR e il ToF. Inoltre, sensori come il LIDAR hanno limitazioni di range che possono essere superate usando un algoritmo stereo. I primi algoritmi per il calcolo della disparità superano i 40 anni di età; questi erano comuni

algoritmi step-by-step estremamente costosi dal punto di vista del tempo. Successivamente, con la diffusione del machine learning e l'aumento delle capacità computazionali, si sono impiegate tecniche di questo tipo per sostituire i singoli step delle pipeline dei vecchi algoritmi. Di recente, tecniche basate sul machine learning hanno permesso un notevole miglioramento; tuttavia, recenti sviluppi basati su tecniche end-to-end avanzate di deep learning, ossia una rete che riceve in input una coppia di immagini stereo e restituisce direttamente una mappa di disparità (senza l'utilizzo di tecniche di elaborazione esterne), ottennero risultati estremamente migliori. Per questi tipi di tecniche in fase di allenamento sono necessari grosse moli di dati etichettati in input. Invece, nel caso di algoritmi non end-to-end erano sufficienti poche centinaia di dati etichettati, ovvero coppie input e output target.

Le principali sfide nell'ambito della Stereo Vision restano tuttora il processamento di immagini ad alta risoluzione [2, 3] e l'analisi di scene contenenti superfici non lambertiane [3]. Era difficile allenare una rete stereo end-to-end per questi due task data la totale assenza di dataset contenenti questo tipo di scene. Per questo motivo, come riportato nella sezione 2.1, durante l'attività di tirocinio in previsione di questo lavoro di tesi si è realizzato il dataset Booster [3].

1.2 Overview sul Deep Learning

Come anticipato nel precedente paragrafo, è necessario introdurre il Deep Learning, una sottocategoria del Machine Learning, appartenente al mondo dell'intelligenza artificiale. Le architetture di Deep Learning si basano su reti neurali artificiali che a partire da grosse moli di dati imparano a svolgere task specifici diversamente dagli algoritmi tradizionali. I dati vengono dati in ingresso alla rete come in Fig. 1.1 attraverso un input layer, vengono processati in strati nascosti (hidden layer), interni all'architettura della rete e, infine, viene generato un risultato dall'ultimo strato (output layer). Oggi l'utilizzo di queste tecniche cresce sempre di più; i campi di applicazione

sono i più svariati: si passa dal riconoscimento di immagini alla scoperta di nuovi medicinali fino alla guida autonoma.

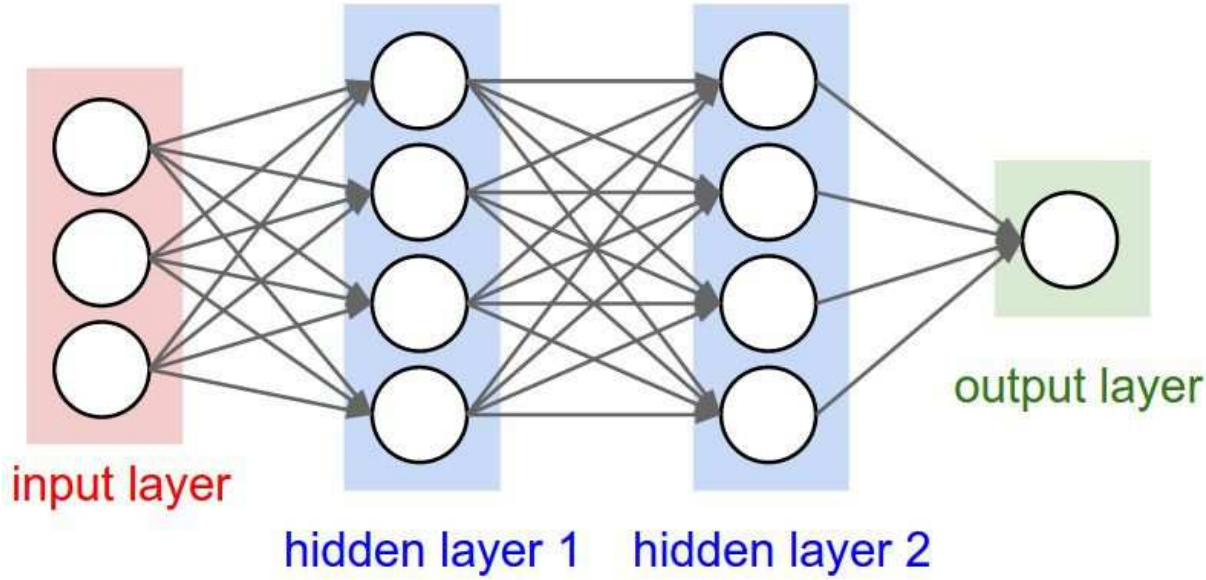

Fig. 1.2: Schema di una rete neurale: deep learning

La rete neurale artificiale prevede che le sia fornito un insieme di dati, comunemente diviso in due parti: training set e test set. La rete viene addestrata sul training set. L'allenamento può avvenire in due modi: supervised e unsupervised.
Nel primo caso, ogni dato di input viene etichettato con un desiderato valore di output, così il sistema, una volta calcolato il valore in uscita, lo confronta con l'etichetta (anche detta ground truth). In questo modo, la rete modifica il modello in base alla differenza tra il valore ottenuto e il valore atteso.
Al contrario, nel secondo caso, non viene fornito nessun valore atteso poiché la rete si costruisce strutture a partire dai dati forniti.
In verità, non esistono solo questi due modi per procedere nella fase di training, ma anche altri che sono ibridi dei precedenti descritti.
Successivamente al training, viene effettuato il test al fine di valutare lo stato di apprendimento della rete. In fase di test una porzione più piccola e non appartenente al training set del dataset viene processata e si restituiscono risultati sia qualitativi che quantitativi.

1.3 Descrizione di RAFT-Stereo

Tutti gli esperimenti all'interno di questa tesi si sono svolti utilizzando la rete neurale RAFT-Stereo [1] in quanto, come descritto nella sezione 2.1 e in [3], le migliori predizioni delle mappe di disparità e le groundtruth di Booster sono state ottenute usando questo algoritmo. Questa rete è stata proposta come modifica dell'originaria RAFT, rete neurale per un altro task della Computer Vision, l'optical flow. Come spiegato nella precedente sezione e nella Fig. 1.2, essendo una rete stereo RAFT-Stereo ha bisogno di due immagini rettificate. Il processo di rettificazione consiste nel riproiettare le immagini lungo un nuovo piano comune parallelo al segmento che congiunge i centri delle telecamere. Tale procedura consente di semplificare notevolmente il processo di ricerca di punti corrispondenti, in quanto dato un punto di coordinate (x, y) sull'immagine di riferimento, il punto ad esso omologo si trova alla medesima coordinata y. Queste due nuove immagini ottenute, che corrispondono ai dati di input, le chiameremo left (I_L) e right (I_R).

La struttura di RAFT-Stereo è composta da tre blocchi principali: un features extractor, una correlation pyramid, e un GRU-based update operator.

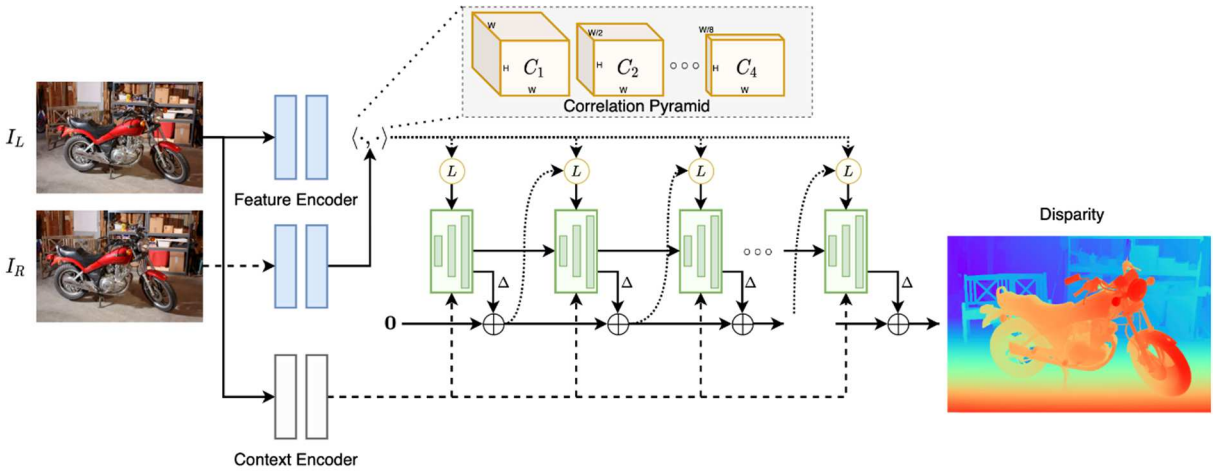

Fig. 1.3: struttura di RAFT-Stereo

Il features extractor, parte azzurra e bianca nella Fig. 1.2, è suddiviso in feature encoder e context encoder, che si occupano entrambi di estrarre attraverso residual block le caratteristiche presenti nella scena (il context encoder però lavora solo sull'immagine left (I_L)). Gli encoder producono le mappe di features a 1/8, a 1/16 e a 1/32 di risoluzione rispetto alle immagini in input. In particolare, nella sezione 3, si vede come il context encoder rende RAFT-Stereo adatta ad un'implementazione semantica.

La Correlation Pyramid, parte gialla in Fig. 1.3, prende in input le mappe di features ($f, g \in \mathbb{R}^{H \times W \times D}$) e costruisce dei volumi (matrici) di correlazione (C) facendo il prodotto scalare tra un pixel ed ogni altro pixel con la stessa coordinata y.

$$C_{mnk} = \sum_{h} f_{mnh} \cdot g_{mnh} \qquad C \in \mathbb{R}^{H \times W \times W} \tag{1.2}$$

Il GRU-based update operator, parte verde in Fig. 1.2, come input ha l'output della correlation pyramid e l'output del context encoder. Gli input vengono processati contemporaneamente a 1/8, a 1/16 e a 1/32 di risoluzione per superare problemi di porzioni di immagini troppo uniformi e senza dettagli. Come ultimo step c'è un modulo di sovracampionamento in modo da calcolare la disparità alla risoluzione più alta.

RAFT-Stereo, come anticipato, è un'architettura supervised. Per il controllo dell'apprendimento della rete utilizza una funzione di perdita di distanza L1:

$$L = \sum_{i=1}^{N} \gamma^{N-i} \; || \, d_{gt} - d_i \, || \quad dove \, \gamma = 0.9 \tag{1.3}$$

Capitolo 2

Datasets

In questo capitolo si introducono i dataset stereo utilizzati per gli esperimenti e altri candidati dataset contenenti oggetti trasparenti, trovati durante le ricerche, ma non utilizzati. Tutti i dataset possono suddividersi in due classi: reali e sintetici. Le immagini dei dataset di ambienti realistici sono state catturate con delle fotocamere e per ogni insieme di dati viene fornita la baseline e la distanza focale. I dataset sintetici sono creati al computer con appositi tool, per questo motivo permettono di ottenere mappe della disparità perfette, al contrario di quelli realistici.

Un ulteriore classificazione, che bisogna introdurre, è il modo in cui le immagini vengono raccolte. Nel caso RGB Stereo vengono catturate due immagini con due telecamere, una coppia left-right. Diversamente, nel caso RGB-D si usa una telecamera e dei sensori per ottenere la groundtruth (che per semplicità chiameremo gt) della depth. Con le RGB-D non si dispone quindi di una coppia di immagini. Al fine di questa tesi, tutti i dataset RGB-D elencati sono validi candidati per il training del modulo semantico dato che in ognuno è presente una maschera semantica.

2.1 Booster

Booster [3] è un dataset reale composto da 64 scene RGB-Stereo caratterizzato prevalentemente da superfici non lambertiane, per un totale di 419 coppie stereo. Ogni scena contiene più coppie di immagini con diversa illuinazione. All'interno del dataset sono presenti uno split "balanced" e uno "unbalanced". Il primo include una coppia stereo left-right a 12 Mpx, mentre nel secondo l'immagine di left ha una risoluzione di 12 Mpx e l'immagine right di 1.1 Mpx. Il training set è formato da 228 coppie stereo (38 scene) e il test set da 191 (26 scene) in entrambe i casi di split. All'interno di

questa tesi si utilizza solo il primo split. Booster comprende una maschera semantica per le superfici non lambertiane. Sono definite 4 classi: classe 0 per gli oggetti e superfici né riflettenti né trasparenti; classe 1 per gli oggetti leggermente trasparenti o riflettenti; classe 2 per elementi sempre più trasparenti o riflettenti; classe 3 per superfici puramente non lambertiane.

2.1.1 Creazione di Booster

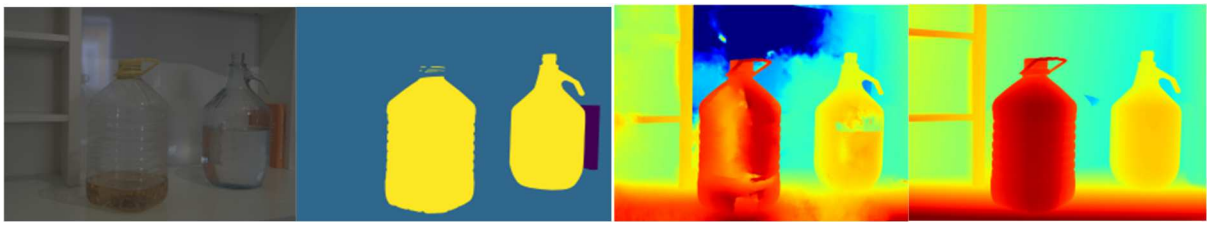

Fig. 2.1: Dataset Booster

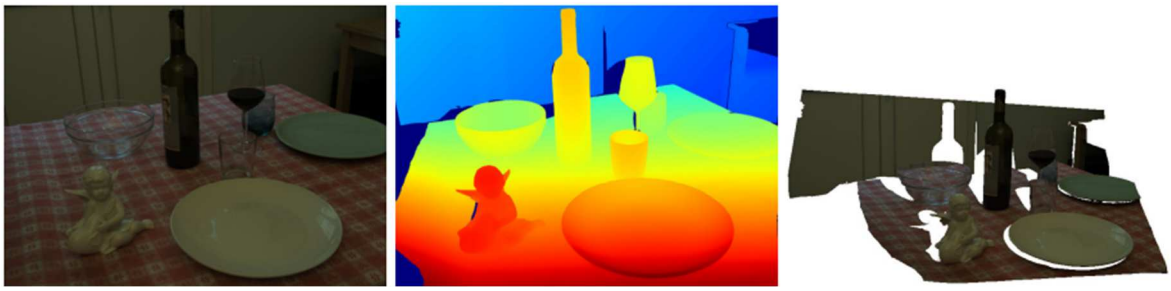

Fig. 2.2: Point cloud di Booster

Durante l'esperienza di tirocinio in preparazione della prova finale, si è realizzato il dataset Booster [3]. In ognuna delle scene, sia del training che del test set, sono presenti almeno due delle classi sopra definite. Sono state acquisite diverse coppie RGB left-right passive con fonti di illuminazione sia naturale che artificiale diverse per ottenere un insieme consistente di annotazioni (prima immagine da sinistra in Fig. 2.1).

In Fig. 2.2 è possibile vedere come la rete ricostruisce la scena tridimensionale osservando l'ultima immagine da sinistra (point cloud). Le sezioni bianche, dietro gli oggetti, sono parti occluse della scena rispetto al punto di vista delle telecamere. Altre sezioni bianche, presenti sui bordi, corrispondono a parti dell'immagine di sinistra che non sono contenute in quella di destra poiché le telecamere sono poste ad una distanza b (Fig. 1.1). Per la creazione delle ground truth si è scelto di usare RAFT-Stereo in quanto è l'algoritmo che performava meglio. La predizione della disparità degli elementi appartenenti alle classi semantiche 1, 2, 3 sebbene fosse la migliore con RAFT-Stereo, risultava comunque sbagliata (terza immagine da sinistra in Fig. 2.1). La riflessione e la trasparenza della luce portava l'algoritmo a predire profondità estremamente più vicine (riflessione) o più lontane (trasparenza). Nel secondo caso non considerava la presenza di un oggetto in quanto vedeva attraverso esso. Prima dell'acquisizione delle immagini attive, le superfici appartenenti alle classi 1, 2, 3, sono state ricoperte di vernice in modo da mascherare le loro proprietà non lambertiane (seconda immagine da sinistra in Fig. 2.3). Le immagini attive sono state acquisite con l'ausilio di sei proiettori per aumentare la trama presente sulla scena e quindi ottenere una ground truth migliore (ultima immagine da sinistra in Fig. 2.1), questa tecnica è chiamata Space Time Stereo [2]. Sulla scena veniva proiettato una sequenza di immagini piene di colori diversi (terza immagine da sinistra in Fig.2.3) che permette a una rete deep stereo di ottenere risultati migliori [5]. Le maschere semantiche per tutte e quattro le classi sono state create a mano (seconda immagine da sinistra in Fig. 2.1).

Fig. 2.3: Acquisizione scene di Booster e Space Time Stereo

2.2 SYNTHIA-SF (BMVC 2017)

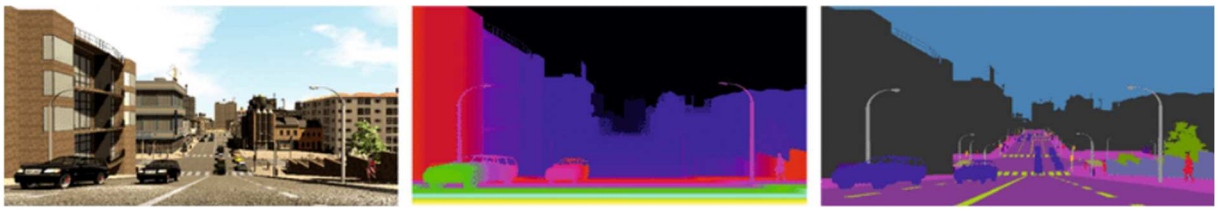

Fig 2.4: Dataset Synthia-SF (BMVC 2017)

SYNTHIA-SF(BMVC 2017) [4] è un dataset sintetico realizzato acquisendo immagini da sequenze video fittizie a 5FPS con risoluzione 1280x960. In esso sono presenti 6 sequenze video con diversi scenari e condizioni di traffico. In totale sono presenti 2224 scene ognuna contenente coppie stereo left-right (left: prima immagine da sinistra in Fig 2.4), gt per la depth (immagine centrale in Fig 2.4), maschera della semantica (terza immagine da sinistra in Fig 2.4), instance segmentation e parametri di calibrazione. Al fine di questa tesi si è utilizzato solamente la coppia left e right, la gt, la maschera e i parametri di calibrazione.

SYNTHIA-SF fornisce una gt per la semantica con 23 classi: in questo lavoro sono state usate solamente 16 classi data la totale assenza delle altre (non sono presenti la classe 0, 10, 13, 17, 18, 19, 22). Per il training set sono state scelte le prime cinque sequenze, l'ultima invece svolge il ruolo di test set. Le gt sono per la depth quindi in fase di processamento vengono convertite con la formula (1.1) per ottenere gt per la disparità, come richiesto dall'architettura di RAFT-Stereo.

2.3 FlyingThings3D

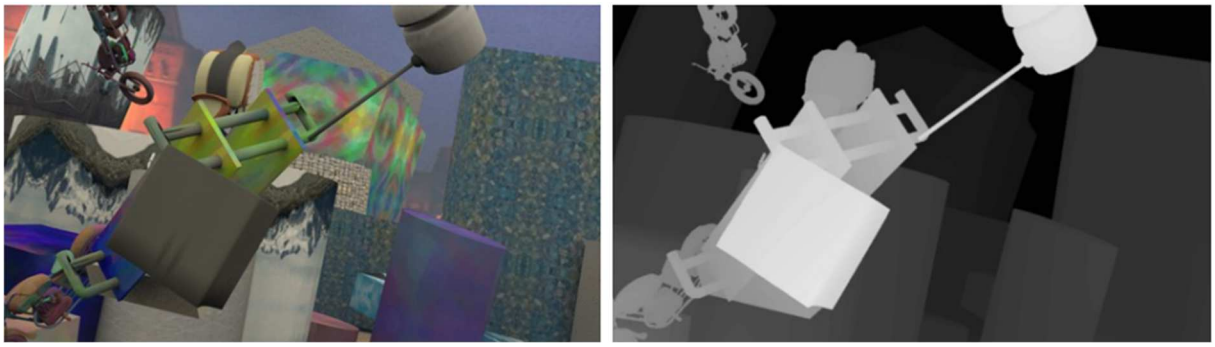

Fig 2.5: Dataset FlyingtThings3D

FlyingThings3D [5] è un dataset sintetico per optical flow, disparità e scene flow estimation ed è composto da circa 25000 coppie Stereo (prima immagine da sinistra in Fig. 2.5) con gt per la disparità (seconda immagine da sinistra in Fig. 2.5), con risoluzione 960x540. Le scene includono oggetti volanti lungo traiettorie 3D aleatorie. In questo dataset le informazioni semantiche non sono presenti e i creatori del dataset hanno già eseguito lo split in training e test set.

2.4 ClearGrasp

Fig 2.6: Dataset ClearGrasp

ClearGrasp [6] è un dataset composto da 50000 immagini sintetiche e 286 immagini reali come test. Nelle scene sono presenti oggetti trasparenti. Per ogni scena vengono fornite una sola immagine RGB-D (prima immagine da sinistra in Fig. 2.6) con risoluzione 1920x1080 e una maschera semantica (seconda immagine da sinistra in Fig. 2.6). In questa tesi, che si occupa di Stereo Vision, viene usato questo dataset non per la disparità bensì per l'addestramento del modulo semantico. Nelle maschere semantiche fornite sono presenti solo due classi, una per gli oggetti trasparenti e una per il resto della scena.

2.5 Altri datasets con superfici non lambertiane

Durante la prima parte di questo lavoro, si è proceduto alla ricerca di candidati dataset contenenti oggetti trasparenti per la Stereo Vision. Oltre a quelli sopra citati, che vengono usati, ne sono stati trovati altri che vale la pena elencare per completezza. Tutti i seguenti dataset contengono la gt della depth o disparità e la maschera semantica. (Per una maggiore chiarezza si omette di riportarlo in ogni sezione)

- Omniverse Object Dataset [7] è un dataset sintetico, esattamente come ClearGrasp dispone solo di una immagine e contiene oggetti trasparenti ed opachi. I modelli degli oggetti sono stati scelti tra quelli di ClearGrasp e di ShapeNet.
- KeyPose [8] è un dataset reale contenente 48000 scene. Per ogni scena è fornita una coppia left e right (RGB Stereo) a 720 pixel di risoluzione e anche un edge mask. Il dataset è stato costruito con 15 oggetti trasparenti appartenenti a 5 classi diverse.

- TransCG [9]: è un dataset reale contenente 57715 RGB-D da 130 differenti scene. Come per Omniverse e ClearGrasp viene fornita una sola RGB-D per il calcolo della depth. Le scene contengono oggetti trasparenti e non.
- ClearPose [10]: è un dataset reale contenente 360000 immagini RGB-D con 63 diversi oggetti trasparenti. Per ogni scena è fornita l'instance segmentation mask e la surface normals.
- TODD [11]: è un dataset reale contenente 15000 immagini RGB-D con 6 diversi oggetti trasparenti.
- TOD [12]: è un dataset reale contenente 48000 immagini RGB-D con 15 diversi oggetti trasparenti.
- StereoOBJ-1M [13]: è un dataset reale contenente circa 150000 coppie di immagini RGB Stereo con 7 diversi oggetti trasparenti.
- Trans10K [14]: è un dataset reale contenente 15000 immagini RGB-D con 10000 diversi oggetti trasparenti.

Capitolo 3

Semantic RAFT-Stereo

In questo capitolo si introduce l'ambiente di lavoro Python, il funzionamento di diversi moduli semantici e la loro implementazione in RAFT-Stereo. In questa tesi si utilizza unicamente Python come linguaggio e tutti i programmi vengono eseguiti su delle GPUs (Graphic Processings Unit).

3.1 Introduzione all'ambiente di lavoro Python

Python è un linguaggio di programmazione di "alto livello", ovvero è considerevole l'astrazione tra il suo funzionamento rispetto ad un linguaggio macchina e al funzionamento del calcolatore. Questo linguaggio è orientato alla programmazione ad oggetti, per questo possiede una grande versatilità e un ampio panorama di possibili applicazioni (scripting, applicazioni distribuite, computazione numerica e molto altro). Diversamente dai suoi principali competitor, C e Java, non presenta una particolare rigidità sintattica: non è necessario il punto e virgola ";" per terminare una riga di codice e al posto delle parentesi si utilizza l'indentazione. La grande flessibilità di Python è fornita anche dall'abbondante numero di librerie disponibili open source. Eseguendo un codice sorgente si entra in una fase di precompilazione in bytecode: non si tratta di un linguaggio interpretato nel vero senso del termine.

All'interno di questo lavoro sono stati utilizzati diverse librerie, si procede descrivendo le principali:

- Torch: è una libreria open source di machine learning, dispone di un ampio range di algoritmi per il deep learning. È alla base della struttura di RAFT-Stereo e

dell'utilizzo dei tensori, speciali array multidimensionali con una maggiore compatibilità computazionale con le GPUs usate in questa tesi. Un immediato esempio è la funzione di perdita usata per l'apprendimento del branch semantico (3.2) che è già implementata all'interno della libreria.

- Numpy: è alla base della creazione e manipolazione di array e matrici multidimensionali in Python, anch'essa open source. Al suo interno sono presenti svariate funzioni per eseguire operazioni trigonometriche, statistiche, algebriche e aritmetiche.
- Opencv: è un insieme di codice creato appositamente per risolvere problemi della computer vision. All'interno di questo lavoro viene utilizzata per ottenere risultati qualitativi della parte di test e per la modifica delle dimensioni delle immagini attraverso una funzione di interpolazione.
- Albumentations: è una libreria open source creata per la modifica delle immagini per tasks della computer vision. In particolare, viene usata in questa tesi nella fase di training in cui i dati vengono modificati per ottenere una maggiore robustezza e generalizzazione dell'apprendimento della rete.
- Matplotlib: è una libreria open source per la creazione di grafici statici o animati e per la visualizzazione interattiva. Permette di creare diversi layout e stili modificabili all'interno dei grafici.
- TensorboardX: è uno strumento di visualizzazione per esperimenti di machine learning. Permette la visualizzazione di metriche e risultati intermedi durante fasi di training.

3.2 Implementazione semantica di RAFT-Stereo

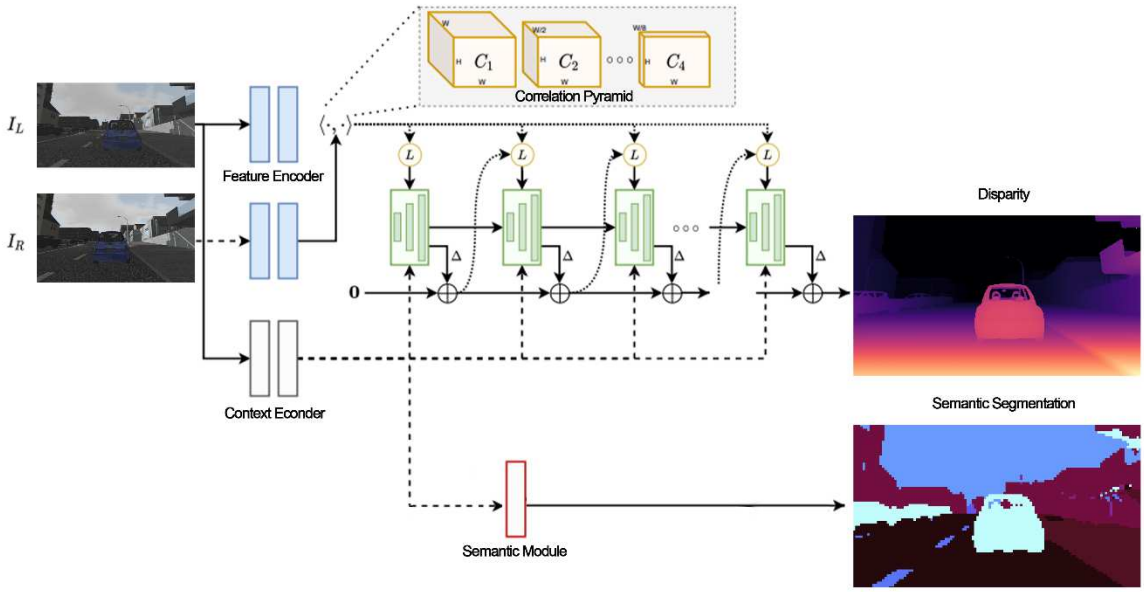

Fig. 3.1: architettura di Semantic RAFT-Stereo

Un modulo semantico è un blocco di una rete neurale che suddivide l'immagine in sezioni in base a ciò che riconosce al suo interno. Semantic Module è composto da uno o più strati, con il compito di imparare a predire la classe di appartenenza degli oggetti dell'immagine esaminata. Esattamente come per la disparità, in fase di training, viene fornita una gt, detta maschera semantica, come previsto dal supervised learning. Esistono esperimenti in cui si svolgono task di predizione della disparità e della semantica insieme (non ancora su RAFT-Stereo) e si valuta come una predizione influenzi l'altra e viceversa. Gli autori di [15] propongono due modi di unire le informazioni semantiche con quelle di profondità: Standalone Approach e Joint-Learning Approach.

Nel primo caso, la rete neurale viene allenata sui due task separatamente e dai risultati semantici ottenuti si procede a raffinare la mappa di disparità.

Nel secondo caso, invece, l'architettura della rete è pensata per estrarre dai dati di input informazioni necessarie per l'addestramento su entrambe i task. Come per il

precedente approccio avviene un miglioramento della mappa di disparità sulla base delle informazioni semantiche nell'ultimo step.
L'approccio seguito in questa tesi è simile al Joint-Learning Approach, con un'unica differenza: non avviene l'ultimo step di miglioramento della mappa di disparità, ma si lascia che l'informazione semantica estratta influenzi automaticamente la predizione della disparità.
Spostando l'attenzione sull'architettura di RAFT-Stereo, in Fig. 1.3, si nota la presenza di un context encoder che opera unicamente sull'immagine di left (I_L). Come precedentemente detto nella sezione 1.3, questo encoder estrae le feature, le caratteristiche dell'immagine in input. Per svolgere questo compito utilizza dei residual block, ossia coppie di layer convoluzionali con funzione di normalizzazione e di attivazione. In questo lavoro si utilizza l'output del context encoder come input per il modulo semantico, anch'esso basato su strati convoluzionali.

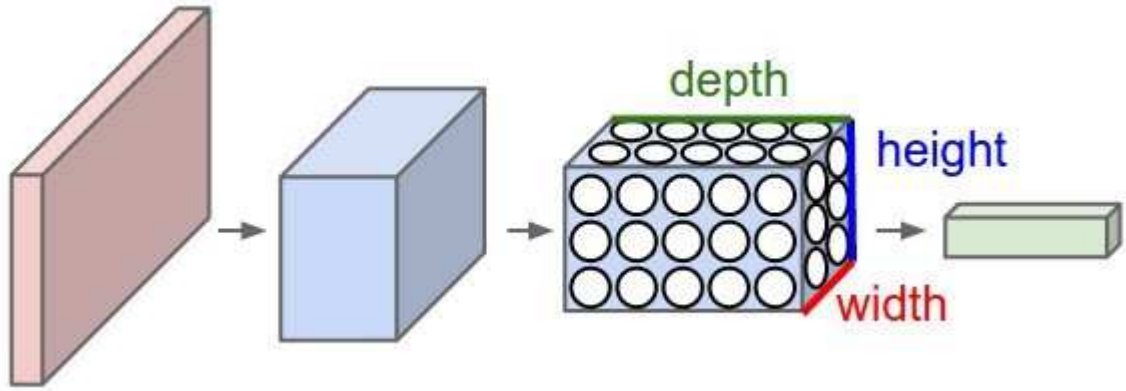

Fig 3.2 Rete neurale convoluzionale

Un layer convoluzionale bidimensionale è un insieme di filtri (anche detti kernel) che si spostano (secondo iperparametri definiti a priori) sul dato di input (bidimensionale) per creare in output dei dati con una fissata depth, Fig. 3.2, pari al numero di filtri presenti nel layer. Viene usato nella Computer Vision per la sua alta efficacia sulle immagini. Oltre alla modifica della dimensione di depth, in Fig. 3.2, che corrisponde al numero di canali di output, possono essere modificate anche la width e la height secondo le seguenti leggi:

$$Hout = \left[\frac{Hin\ + 2\ x\ padding[0] - dilation[0]\ x\ (kernel\_size[0] - 1) - 1}{stride[0]} + 1\right] \quad (3.1)$$

$$Wout = \left[\frac{Win\ + 2\ x\ padding[1] - dilation[1]\ x\ (kernel\_size[1] - 1) - 1}{stride[1]} + 1\right] \quad (3.2)$$

Hin, Hout, Win e *Wout* sono rispettivamente l'altezza, height, di ingresso e uscita e la larghezza, width, di ingresso e uscita. Variando i parametri contenuti nelle espressioni 3.1 e 3.2 è possibile modificare la dimensioni in output a piacimento. Il padding è il numero di 0 aggiunti ad entrambi i lati dell'input ($Hin\ x\ Win$) per ottenere la dimensione desiderata. Il kernel size è la dimensione del filtro che trasla su tutti i dati di input. Il dilation è la distanza tra gli elementi del filtro, non per forza essi devono essere contigui, possono essere intervallati da spazi vuoti. Lo stride è lo spostamento del kernel dopo ogni operazione di filtraggio.

(È stato utile introdurre ora la 3.1 e la 3.2 in quanto in questa tesi quasi tutti i moduli semantici proposti vengono realizzati con layer convoluzionali).

L'input dei moduli, per la predizione semantica, viene generato dal context encoder, che ha tre output, uno per ogni mappa di feature a diversa risoluzione. In questa tesi le mappe sono a 1/4, 1/8 e 1/16 della risoluzione dell'input, [1] prevede 1/8, 1/16 e 1/32. Ogni mappa possiede 128 canali (numero di kernel).

I moduli semantici qui proposti prendono in input una o più mappe e restituiscono output con numero di canali pari al numero di classi semantiche presenti nel dataset.

3.3 Tipi di modulo semantico

In questa sezione si introducono i vari moduli semantici utilizzati in questa tesi. Ad eccezione del multi-layer deconvoluzionale tutti si basano su layer convoluzionali modificando l'input secondo la 3.1 e la 3.2. I primi due moduli semantici proposti permettono l'analisi dell'input ad una qualsiasi risoluzione: tutte e tre le risoluzioni di output del context encoder di RAFT-Stereo sono validi dati in ingresso. I successivi prevedono una definita risoluzione o più risoluzioni in input.

3.3.1 Singolo layer convoluzionale

Questo modulo è il più semplice in assoluto, è composto da un singolo strato convoluzionale come si può vedere in Fig. 3.3.

```
class SegmentationHead(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(SegmentationHead,self).__init__()

        self.conv = nn.Conv2d(in_channels=input_dim,out_channels=output_dim,kernel_size=3,padding=1,stride=1)

    def forward(self,x):
        x = self.conv(x)
        return x
```

Fig. 3.3: codice del singolo layer convoluzionale

In Fig. 3.3 si notano due variabili di input: *input_dim* e *output_dim*. *Input_dim* corrisponde al numero di canali del dato in ingresso al layer. In questa tesi l'input del modulo semantico corrisponde all'output del context encoder di RAFT-Stereo, quindi *input_dim* = 128. Invece, *output_dim* è il numero di canali in uscita dal modulo semantico, quindi corrisponde al numero di classi semantiche presenti nel dataset considerato. Come in ogni modulo di una rete neurale è necessario definire una funzione di __init__ in cui si definiscono i layer ed una funzione di forward. La funzione di forward presenta come input la variabile x. X è il dato in ingresso al modulo e all'interno di questa funzione l'input passa in modo ordinato attraverso i layer presenti in forward. Infine, con l'ultima riga di codice si restituisce l'output.

Si confrontano i parametri scelti nella definizione del layer e le equazioni 3.1 e 3.2 e dopo un semplice calcolo è possibile vedere che le dimensioni di input e output corrispondono.

3.3.2 Multi-layer convoluzionale

Questo modulo semantico presenta solo layer convoluzionali. Si passa dal numero di canali in input al numero di canali richiesti in output attraverso più strati.

```
class SegmentationHead3(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(SegmentationHead3,self).__init__()


        self.conv1 = nn.Conv2d(in_channels=input_dim,out_channels=output_dim*4,kernel_size=3,padding=1,stride=1)
        self.conv2 = nn.Conv2d(in_channels=output_dim*4,out_channels=output_dim*2,kernel_size=3,padding=1,stride=1)
        self.conv3= nn.Conv2d(in_channels=output_dim*2,out_channels=output_dim,kernel_size=3,padding=1,stride=1)

    def forward(self,x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        return x
```

Fig. 3.4: codice del multi-layer conovoluzionale

Dalla Fig 3.4 è possibile notare che è stato scelto di usare 3 layer convoluzionali con parametri tutti uguali al di fuori del numero di canali di input e output. Come si può vedere il primo layer, conv1, riceve in ingresso l'output del context encoder e restituisce in uscita un tensore con 4 volte il numero di canali richiesti in output dal modulo. Conv2 come input riceve l'uscita di conv1 e dimezza il numero di canali e lo stesso accadde con conv3 che riceve in input l'output di conv2, dimezza il numero dei canali e fornisce in uscita al modulo il numero di canali corrispondente al numero di classi semantiche.

Come per il modulo nella sezione 3.3.1, sfruttando le equazioni 3.1 e 3.2 è immediato accorgersi che le dimensioni di output e quelle di input combaciano.

3.3.3 Multi-layer deconvoluzionale

Questo è l'unico modulo semantico proposto che non utilizza layer convoluzionali, ma deconvoluzionali, ossia convoluzioni trasposte. Infatti, le dimensioni in uscita da uno strato deconvoluzionale non si calcolano secondo le equazioni 3.1 e 3.2. L'output di un layer di questo tipo segue le seguenti leggi:

$$Hout = (Hin - 1) \times \text{stride}[0] - 2 \times \text{padding}[0] + \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) + \text{output\_padding}[0] + 1 \quad (3.3)$$

$$Wout = (Win - 1) \times \text{stride}[1] - 2 \times \text{padding}[1] + \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) + \text{output\_padding}[1] + 1 \quad (3.4)$$

Ad eccezione dell'output_padding tutti gli altri elementi presenti nell'equazioni 3.3 e 3.4 corrispondono a quelli delle equazioni 3.1 e 3.2. Si aggiunge un numero di 0 pari al valore di output_padding ad entrambe le dimensioni di output.

```
class SegmentationHead2(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(SegmentationHead2,self).__init__()

        self.deconv1 = nn.ConvTranspose2d(in_channels=input_dim, out_channels= output_dim*4, kernel_size=2, stride=2)
        self.deconv2 = nn.ConvTranspose2d(in_channels=output_dim*4, out_channels= output_dim*2, kernel_size=2, stride=2)
        self.deconv3 = nn.ConvTranspose2d(in_channels=output_dim*2, out_channels= output_dim, kernel_size=2, stride=2)

    def forward(self,x):
        x = self.deconv1(x)
        x = self.deconv2(x)
        x = self.deconv3(x)

        return x
```

Fig. 3.5: codice del multi-layer deconvoluzionale

Come nei precedenti moduli, nella Fig. 3.5 è possibile vedere che, le variabili di input sono le stesse: il numero di canali in ingresso, *input_dim*, e il numero di canali in uscita, *output_dim*. Questo modulo semantico è stato pensato per avere in ingresso un input con risoluzione 1/8 (il secondo output del context encoder). Osservando i

parametri degli strati deconvoluzionali e inserendoli all'interno delle equazioni 3.3 e 3.4, dopo un semplice calcolo, è possibile vedere che ogni layer restituisce un output con Hout = 2 * Hin e Wout = 2 * Win. Quindi in uscita dal modulo si ottengono Hout = 2 * 2 * 2 * Hin = 8 * Hin e Wout = 2 * 2 * 2 * Win = 8 * Win. Così facendo la predizione semantica risulta essere alla stessa risoluzione delle immagini left e right in input a RAFT-Stereo. Questo non vieta che gli possano essere dati in ingresso risoluzioni 1/4 o 1/16, la risoluzione di output risulterà sempre aumentata di un fattore 8.

3.3.4 PSP

PSP sta per Pyramid Scene Parsing ed è un modulo semantico proposto in [16] e anche utilizzato in [15]. Anche in questo caso, come nei precedenti, si cercano valori di errore sempre più bassi provando diverse architetture.

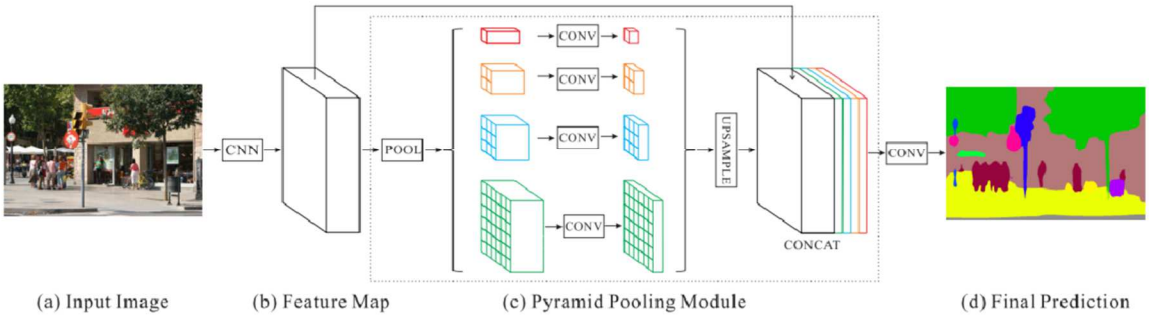

Fig 3.6: Struttura PSP

In questo lavoro la struttura del PSP viene leggermente modificata in modo da essere compatibile con l'architettura di RAFT-Stereo. La versione originale prevede strati di pooling prima dei layer convoluzionali come si può vedere in Fig 3.6. Tuttavia, RAFT-Stereo, attraverso il context encoder, svolge già il ruolo da sotto-campionatore che sarebbe compito di uno o più layer di pooling.

```python
class SegmentationHead_PSP(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(SegmentationHead_PSP,self).__init__()

        self.upsample_bil1 = nn.UpsamplingBilinear2d(scale_factor=2)
        self.upsample_bil2 = nn.UpsamplingBilinear2d(scale_factor=4)

        self.conv1 = nn.Conv2d(in_channels=input_dim,out_channels=input_dim/2,kernel_size=1)
        self.conv2 = nn.Conv2d(in_channels=input_dim,out_channels=input_dim/4,kernel_size=1)
        self.conv3 = nn.Conv2d(in_channels=input_dim + input_dim/2 +input_dim/4,out_channels=output_dim,kernel_size=3,padding=1,stride=1)

    def forward(self,x1,x2,x3):

        x2 = self.conv1(x2)
        x2 = self.upsample_bil1(x2)
        x3 = self.conv2(x3)
        x3 = self.upsample_bil2(x3)
        conc = torch.cat((x1,x2,x3),1)
        y = self.conv3(conc)

        return y
```

Fig. 3.7: codice del modulo PSP

Dopo lo strato di pooling che non è presente, non avvengono modifiche all'architettura PSP. Vengono definiti 3 layer convoluzionali e 2 layer di sovra-campionamento bilineare.

Conv1 dimezza il numero di canali e usando la 3.1 e la 3.2 è immediato ottenere che la larghezza e l'altezza dell'output corrispondono a quelle di input. Conv2 svolge lo stesso compito di conv1 con la sola differenza che il numero di canali viene diviso per 4. Conv3 invece riceve in ingresso la somma dei canali in output di conv1 e conv2 e il numero di canali di un input non modificato. Upsample_bil1 e Upsample_bil2 rispettivamente raddoppiano e quadruplicano l'altezza e la larghezza dei loro input. Diversamente dai moduli visti fino ad ora, il PSP ha come ingresso tutte e tre le mappe di features a diverse risoluzioni che sono output del context encoder. Osservando la funzione forward in Fig. 3.7 si nota che sono presenti 3 input: x1, x2, x3 (rispettivamente a risoluzione 1/4, 1/8, 1/16). Il numero di canali di ingresso a risoluzione 1/8 viene dimezzato e poi viene sovra-campionato con un fattore 2. Il numero di canali dell'input a risoluzione 1/16 viene diviso per 4 e successivamente sovra-campionato con un fattore 4. Infine, come mostrato in Fig. 3.6 l'input a risoluzione 1/4 e gli altri due sovra-campionati vengono concatenati in un unico tensore dato in ingresso a conv3 che produce un output con numero di canali pari al numero di classi semantiche presenti nel dataset.

3.3.5 ASPP

La sigla ASPP sta per Atrous Spatial Pyramid Pooling. Questo modulo semantico viene introdotto in [17] e la sua particolarità è l'utilizzo di diversi layer convoluzionali con diversi valori di dilation per catturare più informazioni possibili dall'input.

```
class SegmentationHead_ASPP(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(SegmentationHead_ASPP,self).__init__()

        self.norm1 = nn.BatchNorm2d(input_dim*2)

        self.avgpool= nn.AvgPool2d(kernel_size=4,stride=4)
        self.upsample_bil = nn.UpsamplingBilinear2d(scale_factor=4)

        self.conv1 = nn.Conv2d(in_channels=input_dim,out_channels=input_dim*2,kernel_size=3,padding=12,dilation=12,stride=1)
        self.conv2 = nn.Conv2d(in_channels=input_dim,out_channels=input_dim*2,kernel_size=3,padding=24,dilation=24,stride=1)
        self.conv3= nn.Conv2d(in_channels=input_dim,out_channels=input_dim*2,kernel_size=3,padding=36,dilation=36,stride=1)
        self.conv4= nn.Conv2d(in_channels=input_dim,out_channels=input_dim*2,kernel_size=1,padding=0,dilation=1,stride=1)
        self.conv5= nn.Conv2d(in_channels=input_dim*10,out_channels=input_dim*2,kernel_size=1,padding=0,dilation=1,stride=1)
        self.conv6= nn.Conv2d(in_channels=input_dim*2,out_channels=output_dim,kernel_size=1,padding=0,dilation=1,stride=1)
        self.conv7= nn.Conv2d(in_channels=input_dim,out_channels=input_dim*2,kernel_size=1,padding=0,dilation=1,stride=1)

    def forward(self,x):
        img_features= self.avgpool(x)
        img_features= self.conv7(img_features)
        img_features = self.upsample_bil(img_features)

        u = self.conv1(x)
        v = self.conv2(x)
        w = self.conv3(x)
        y = self.conv4(x)
        conc = torch.cat((self.norm1(u),self.norm1(v),self.norm1(w),self.norm1(y),self.norm1(img_features)),1)
        z = self.conv5(conc)
        z1 = self.norm1(z)
        return self.conv6(z1)
```

Fig. 3.8: codice del modulo semantico ASPP

In Fig. 3.8 il primo layer che viene definito è norm1. Questo strato si occupa di eseguire una Batch Normalization su un input con un numero di canali doppio rispetto a quelli in ingresso al modulo semantico. Il layer successivo è avgpool che esegue una funzione di pooling, AvgPool2d() modifica le dimensioni dell'input restituendo in output un tensore secondo le seguenti equazioni:

$$Hout = \left[\frac{Hin\ + 2\ x\ padding[0] - kernel\_size[0]}{stride[0]} + 1\right] \qquad (3.5)$$

$$Wout = \left[\frac{Win\ + 2\ x\ padding[1] - kernel\_size[1]}{stride[1]} + 1\right] \qquad (3.6)$$

Note le equazioni 3.5 e 3.6 è immediato notare che le dimensioni dell'output di avgpool sono 1/4 di quelle di input. Upsample_bil svolge un sovra-campionamento del dato in ingresso con fattore 4.

Spostando l'attenzione sui layer convoluzionali, si nota che conv1, conv2, conv3 hanno in ingresso il numero di canali di input del modulo e in uscita il doppio di quel numero. Tutti e tre gli strati hanno padding e dilation con stesso valore. Ad esempio, in conv1 dilation = 12 significa che tra un elemento del filtro e il successivo ci sono 12 spazi. Il padding invece viene scelto pari al dilation in modo tale che, secondo la 3.1 e la 3.2, le dimensioni in uscita risultino uguali a quelle in ingresso. Conv4 e conv7 svolgono la stessa funzione: raddoppiano il numero di canali e lasciano invariate le dimensioni. Per questi due layer viene scelto un filtro 1x1 in modo da modificare il meno possibile il dato in entrata. Conv5 riceve in input un tensore con numero di canali 10 volte il numero di quelli in ingresso al modulo e ne restituisce uno con numero di canali pari a due volte il numero di quelli in entrata al modulo. Come per conv4 e conv7, gli altri parametri sono impostati per modificare il meno possibile le informazioni in ingresso, infatti kernel_size = 1. Anche in questo strato le dimensioni di input e output combaciano. Infine, conv2 riceve in ingresso un numero di canali pari al doppio di quelli in input al modulo e ne restituisce un numero pari al numero di classi semantiche, non modificando l'altezza e la larghezza.

Osservando la funzione forward, in Fig. 3.8, si nota che è presente una sola variabile di ingresso; quindi, viene passata al modulo una sola mappa di features. L'input viene elaborato in cinque modi differenti. Nel primo caso passa attraverso il layer di pooling che riduce le dimensioni ad 1/4 di quelle in ingresso allo strato, poi nello strato conv7 in cui raddoppia i canali e infine le dimensioni ritornano uguali a quelle in ingresso al

modulo grazie al sovra-campionatore upsample_bil. Gli altri quattro casi prevedono un solo layer: dal conv1 al conv4. Eseguite queste elaborazioni tutti gli output ottenuti vengono concatenati lungo la dimensione 1 per ottenere un totale di canali pari a 10 volte il numero di canali in entrata al modulo. Prima che avvenga la concatenazione però su ognuno dei cinque output intermedi viene svolta la funzione di Batch Normalization, in modo da normalizzare i dati. Il tensore ottenuto con la concatenazione viene dato in ingresso a conv5 che divide per 5 il numero dei canali, viene fatta nuovamente una normalizzazione e con conv6 si ottiene il numero di canali pari al numero di classi semantiche.

Capitolo 4

Risultati sperimentali

In questo capitolo si introducono le principali metriche per la valutazione della semantica e della disparità, si mostrano i risultati che portano alla scelta di un modulo semantico piuttosto che altri, si descrivono gli iperparametri utilizzati e si valutano i risultati ottenuti dagli esperimenti condotti.

4.1 Metriche di valutazione

In questa sezione si analizzano le metriche di valutazione usate per valutare i risultati della disparità e della semantica.

4.1.1 Metriche per la disparità

Per valutare in modo quantitativo l'apprendimento della rete in fase di test per la disparità vengono utilizzate 6 differenti metriche: bad 2.0, bad 4.0, bad 6.0, bad 8.0, avgerr, rms. Le prime quattro sono delle percentuali e le ultime due dei numeri puri. L'input di RAFT-Stereo sono due immagini RGB rettificate perciò, come discusso nella sezione 1.3, punti corrispondenti nelle due immagini presentano la stessa coordinata verticale. Questo permette di valutare, con le prime quattro metriche, la percentuale di pixel in cui la differenza in valore assoluto fra la mappa di disparità predetta e la gt è superiore ad una certa soglia. Bad 2.0 è la percentuale di errore in cui l'errore stesso è maggiore di 2 pixel, lo stesso vale per le altre metriche bad x.0 con

soglia pari a x pixel. Invece, avgerr è la media del valore assoluto dell'errore in pixel e rms è la radice quadrata del quadrato dell'errore in pixel. La metrica principale della disparità è avgerr, su cui si sono basate le scelte fatte riguardanti quale modulo semantico utilizzare.

4.1.2 Metriche per la semantica

Per una valutazione numerica delle predizioni semantiche sono state introdotte due metriche: MIoU (Mean Intersection over Union) e IoU (Intersection over Union). La MIoU è la media della IoU di ogni classe presente nella scena. La IoU, anche conosciuta come indice di Jaccard, corrisponde al rapporto tra l'area di sovrapposizione e l'area di unione.

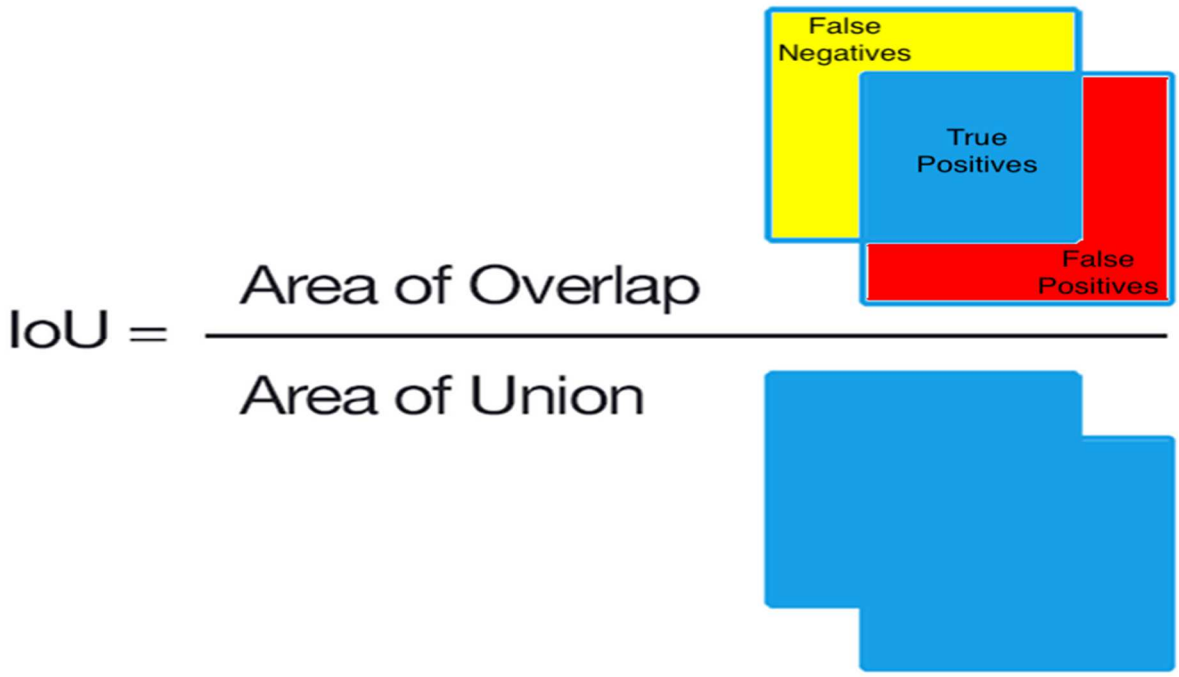

Fig. 4.1: Intersection over Union

In Fig. 4.1, l'area di overlap corrisponde all'area di sovrapposizione. False Negatives è la parte di immagine che, secondo la maschera semantica (gt), appartiene ad una certa classe; False Positives è una sezione dell'immagine che secondo la rete viene predetta appartenere a quella classe; True Positives è l'area in cui predizione e maschera semantica si sovrappongono.

Invece, l'area di unione, Area of Union corrisponde all'unione delle due porzioni False Negatives e False Positives.

In generale, i valori di MIoU sotto il 50% non sono considerati un buon risultato, sopra il 50% vengono considerati buoni e sopra il 90% eccellenti.

L'MIoU così come è stata descritta è la metrica usata nel campo dell'object detection, altra branca della computer vision. Per valutare la semantic segmetation utilizziamo un MIoU calcolato pixel per pixel, confrontando la predizione semantica con la gt.

4.2 Debug della rete e scelta del modulo semantico

Come anticipato nella sezione 3.2, non è presente nessuno stage finale per il miglioramento della mappa di disparità sulla base dei risultati semantici, ma si cerca di influenzare automaticamente il comportamento della rete con la sola predizione semantica. Per vedere l'efficacia di questa ipotesi, si è optato per un training su Synthia-SF che, come anticipato nella sezione 2.2, è un dataset sintetico che include sia gt della disparità che maschera semantica. Questo dataset viene utilizzato per comodità solo per valutare l'efficacia nel riconoscimento della corretta classe di appartenenza di comuni elementi in quanto non presenta superfici non lambertiane al suo interno che sono l'oggetto di studio di questa tesi. Il numero di canali in output da ogni modulo semantico deve essere 16 come il numero di classi all'interno del dataset. In fase di training, esattamente come per la disparità, è presente una funzione di

perdita 1.3, bisogna quindi introdurne una anche per la semantica. Viene scelta la funzione cross_entropy, la cui espressione è:

$$l(x,y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1\{y_n \neq ignore.index\}} l_n \qquad (4.1)$$

Dove l_n vale:

$$l_n = -w_{y_n} log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq ignore.index\} \qquad (4.2)$$

Nelle espressioni 4.1 e 4.2, x è l'input, y è il target (nel nostro caso la classe di appartenenza della maschera semantica), w è la matrice dei pesi, C è il numero di classi semantiche e N è il valore del minibatch processato ad ogni iterazione. Questa funzione di perdita esegue già una logsoftmax che serve per ottenere valori della predizione di appartenenza ad una determinata classe in forma probabilistica ossia compresi in [0,1], la cui somma restituisce 1. La classe di appartenenza viene scelta prendendo il massimo valore tra i vari canali che corrispondono alle classi semantiche.

RAFT-Stereo è stato allenato su Synthia-SF per ogni modulo semantico con i seguenti iperparametri: epoche = 50, batch = 6, crop_size = [256, 480] e max_disp = 192. Con 50 epoche si intende che ogni coppia di immagini del training set è stata data in input alla rete 50 volte; il valore di max_disp è stato impostato a 192, ossia la massima disparità calcolabile all'interno delle scene è 192. Per ottenere valori compresi tra 0 e 192 è stato fatto uno scaling dell'input di un fattore 1/2. Le immagini di input non vengono passate direttamente alla rete, ma vengono tagliate in porzioni di dimensioni 256x480 pixel e vengono fornite in ingresso a gruppi (batch) di 6 per volta.

Nella seguente tabella vengono riportati i risultati quantitativi di disparità e semantica:

Risoluz.	**nome**	**bad2.0**	**bad4.0**	**bad6.0**	**bad8.0**	**avgerr**	**rms**	**MIoU**
-	No_sem	3.69	2.02	1.47	1.16	0.7028	3.56	-
1/8	Sem1	3.37	1.89	1.38	1.10	0.6615	3,404	56.37
1/8	Deconv	3.73	2.03	1.47	1.16	0.7100	3.55	56.00
1/4+1/8 +1/16	Sem2	3.46	1.94	1.42	1.12	0.6657	3.4731	56.00/ 52.00/ 52.00
1/4	Sem3	3.45	1.91	1.38	1.09	0.6625	3.4458	54.12
1/16	Sem4	**3.35**	**1.86**	**1.35**	**1.07**	**0.6500**	**3.4025**	51.38
1/4+1/8 +1/16	PSP	3.59	1.96	1.41	1.12	0.6919	3.5658	**60.55**
1/8	ASPP	3.42	1.9	1.39	1.10	0.6546	3.432	49.37
1/4	Sem5	3.49	1.94	1.41	1.11	0.6693	3.4851	56.10
1/8	Sem6	3.68	2.03	1.47	1.16	0.6976	3.5444	56.20

Tab. 4.1: Risultati Synthia-SF

Fig. 4.2: Left, gt della disparità e predizione della disparita con Sem1 (da sinistra)

Fig. 4.3: Gt semantica, risultati Sem1 e Deconv (da sinistra)

Fig. 4.4: Risultati Sem2 a 1/4, Sem2 a 1/8, Sem2 a 1/16 (da sinistra)

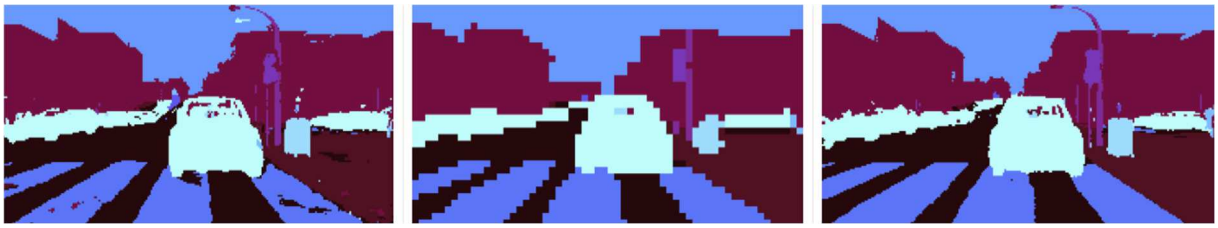

Fig. 4.5: Risultati Sem3, Sem4, PSP (da sinistra)

Fig. 4.6: Risultati ASPP, Sem5, Sem6 (da sinistra)

Nella prima riga in Tab. 4.1 sono riportati i risultati del test senza alcun modulo semantico. Questi valori sono fondamentali per fare un paragone con i successivi e valutare l'influenza del modulo sulla predizione della disparità. Nella prima colonna sono presenti i valori di risoluzione con cui sono calcolati le mappe semantiche. Tuttavia, i valori riportati rappresentano lo scaling fatto dal context encoder di RAFT-Stereo rispetto all'input. In Fig. 4.2 viene riportata la left dell'immagine di test considerata, la gt della disparità e una mappa della disparità predetta a 1/8 con Sem1.

Le altre mappe della disparità non vengono inserite in quanto una diminuzione di errore di qualche decimale non è percettibile. Il fattore di scaling nel caricamento dei dati è pari a due quindi l'intera colonna deve essere moltiplicata per un fattore 1/2 per conoscere il reale valore di risoluzione in ingresso al modulo semantico.

In Sem1, Sem3 e Sem4 viene utilizzato un singolo layer convoluzionale 3.3.1. A parità di modulo, si può notare che il migliore risultato di disparità si ottiene con la risoluzione in input più bassa (1/16). Tuttavia, la semantica predetta a 1/16 (Fig. 4.5) risulta essere molto grezza, infatti, i confini degli oggetti non risultano chiari. Tra le 3 risoluzioni, il miglior risultato di predizione semantica si ottiene con la risoluzione intermedia (1/8). È possibile notare un singolo grande errore all'interno della striscia pedonale in Fig. 4.3 (una sorta di buco), ma il resto della predizione sembra coerente con la maschera fornita, Fig. 4.3. Con la risoluzione più bassa, invece, è possibile vedere in Fig. 4.5 come i contorni siano definiti meglio, ma gli errori sono maggiori: sono presenti diversi buchi all'interno delle strisce pedonali, nel marciapiede e nel cielo vengono predetti classi in alcuni punti totalmente sbagliate. In tutti e tre i casi è netto il miglioramento della disparità rispetto al modello senza semantica.

Deconv è il modulo descritto in 3.3.3: la predizione semantica risulta essere buona. Il modulo, riportando in output la risoluzione pari a quella di input di RAFT-Stereo, produce una mappa semantica con confini molto marcati, come si può vedere in Fig. 4.2. Tuttavia, è palese che non influenzi la disparità in quanto si ottiene un valore di avgerr maggiore rispetto al caso No_sem.

Nel caso Sem2 vengono usati in parallelo 3 moduli semantici composti da un singolo layer convoluzionale 3.3.1 e si può notare una discreta predizione semantica ad ogni risoluzione (la migliore a risoluzione più alta) e anche un miglioramento nella disparità rispetto al caso senza semantica. Comparando la predizione a 1/8 fatta da Sem1 in Fig. 4.3 e il risultato ad 1/8 di Sem2 in Fig. 4.4, si vede come l'utilizzo delle tre risoluzioni sembri correggere qualitativamente gli errori di Sem1. In realtà osservando i risultati numerici questo non è vero in quanto Sem2 ad 1/8 perde più del 0.04 in MIoU.

Col modulo PSP, sezione 3.3.4, si ottiene il valore di MIoU più alto in assoluto, superiore a 0.6. Come si può vedere in Fig. 4.5, i contorni sono delineati e c'è una

buona coerenza con la maschera semantica, tuttavia, non avviene un corrispondente abbassamento dell'avgerr.

Diametralmente opposto al caso PSP, col modulo ASPP, sezione 3.3.5, si ottiene il risultato semantico più basso in assoluto, inferiore a 0.5. Mentre l'errore di disparità si riduce discretamente. È possibile vedere in Fig. 4.6 come la predizione semantica non riconosca l'auto sulla destra e gli spazi tra una striscia pedonale e la successiva.

Nel caso Sem5 si utilizza un Multi-layer convoluzionale con risoluzione 1/4, sezione 3.3.2. La predizione semantica è buona e anche l'errore di disparità si abbassa. Numericamente la semantica non risulta essere predetta male, tuttavia, qualitativamente sono presenti diversi fori all'interno delle strisce pedonali.

Nell'ultimo caso, Sem6, si usa un Multi-Layer convoluzionale con risoluzione 1/8. Come per Sem5, la predizione semantica restituisce buoni risultati. Confrontando i risultati qualitativi tra Sem1 e Sem6 non si nota una grande differenza e questo è confermato anche dai risultati numerici. Però l'errore di disparità non cala, rimane circa uguale al modello senza modulo semantico.

Il massimo valore di MIoU si ottiene nel caso PSP, mentre il minimo valore di avgerr si ottiene col modello Sem4. In entrambe i modelli una metrica risulta essere ottima, ma l'altra non è altrettanto soddisfacente. L'obiettivo ultimo è un miglioramento della disparità, tuttavia si vuole indagare anche l'influenza di un buon riconoscimento semantico nella predizione della profondità della scena. Un buon compromesso è Sem1: l'avgerr aumenta solo di 0.02 rispetto al caso migliore e si ottiene un valore di MIoU maggiore di 0.0499 rispetto a Sem4. I successivi esperimenti con dataset contenenti oggetti trasparenti e riflettenti vengono tutti eseguiti con Sem1 come modulo semantico.

4.3 Training eterogeneo

Ora che è stato scelto il modulo semantico più efficace, si procede alla configurazione di un training su dataset contenenti oggetti trasparenti. L'obiettivo è allenare la rete ed ottenere un modello da utilizzare su Booster (vedere sezione 2.4), che è un dataset reale molto piccolo con immagini ad alta risoluzione contenente superfici non lambertiani. Dato il numero limitato di immagini di Booster, si esegue un training su un dataset sintetico di grandi dimensioni contenente oggetti trasparenti e riflettenti con maschera semantica per poi procedere con un fine-tuning su Booster. Tuttavia, non esistono dataset stereo sintetici di grandi dimensioni con gt per la disparità e maschera semantica solo per gli oggetti non lambertiani. Tutti i dataset trovati nelle ricerche svolte che possiedono gt e maschera semantica sono per reti monoculari, non stereo. In questa tesi si è scelto di allenare il modulo semantico e la rete per la disparità su dataset differenti. Infatti, come dataset è stato scelto di fondere due dataset esistenti: FlyingThings3D, sezione 2.3, e ClearGrasp, sezione 2.4. Il primo è un dataset sintetico in cui non sono presenti superfici non lambertiane, mentre nel secondo sì. ClearGrasp fornisce una maschera semantica esattamente come viene richiesta, ossia solo due classi: classe 0 per tutte le superfici opache e classe 1 per gli oggetti trasparenti e riflettenti. Modificando l'architettura di RAFT-Stereo è stato possibile dare in ingresso alla rete non più due RGB, ma tre: le prime due per il calcolo della disparità appartenenti a FlyingThings3D e la terza, per la predizione semantica, appartenente a ClearGrasp. In questo modo bisogna eseguire il context encoder sia sulla left della disparità sia sull'RGB associata alla predizione semantica. Dato che le classi semantiche si riducono a due, si sceglie di modificare la funzione di perdita da cross_entropy, espressione 4.1 e 4.2, a binary_cross_entropy. La funzione binary_cross_entropy è la seguente:

$$\ell(x, y) = media(L) = media(\{l_1, \dots, l_N\}^T) \qquad (4.3)$$

$$l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)] \qquad (4.4)$$

Anche se sono presenti due classi semantiche, ora i canali di output del modulo semantico non sono 2, ma 1. Viene scelta la classe di appartenenza eseguendo una funzione sigmoidea:

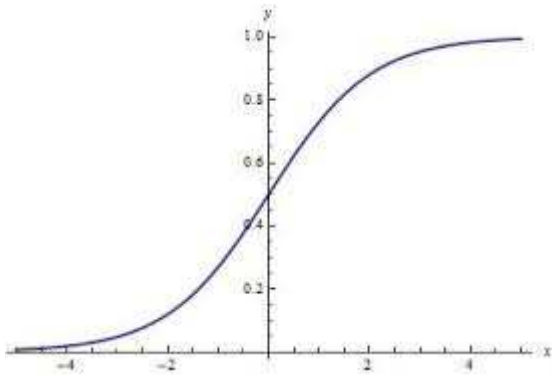

Fig. 4.7: Funzione sigmoidea

La funzione sigmoidea accetta in input valori compresi in $(-\infty, +\infty)$ e restituisce valori compresi tra [0,1]. La scelta della classe semantica viene fatta come segue: 1 dove la sigmoidea restituisce valori > 0.5 e 0 dove i valori sono <= 0.5.

4.4 Scelta degli iperparametri ed esperimenti di fine-tuning

Data la grandezza del dataset, per il training eterogeneo vengono modificati alcuni iperparametri rispetto al training su Synthia-SF per un limite fisico di memoria allocabile all'interno delle GPUs. Gli iperparametri usati risultano i seguenti: epoche = 50, max_disp = 256, batch = 4, crop_size = [256,480]. La modifica della batch è legata all'utilizzo di memoria, invece si sceglie di aumentare max_disp per avere un intervallo maggiore di valori.

All'interno di FlyingThings3D sono presenti valori elevati di disparità, perciò, affinché la rete apprenda correttamente come predirla, è stata inserita una maschera nella funzione di perdita: non vengono passati in ingresso tutti i pixel, ma solo quelli con valori di disparità della gt > 0 e gt <= max_disp (affinché un valore di disparità sia considerato valido deve essere positivo).

Dopo 50 epoche, eseguendo un test col modello ottenuto, le mappe semantiche risultano:

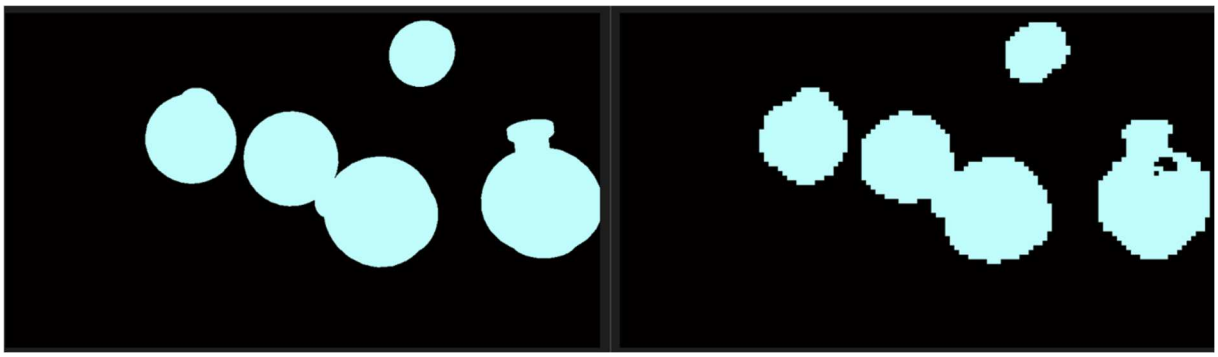

Fig. 4.8: Risultati qualitativi ClearGrasp

Come si può vedere in Fig. 4.8 si ottengono qualitativamente dei buoni risultati e i contorni degli oggetti trasparenti vengono quasi sempre rispettati. È presente un palese errore all'interno dell'oggetto più a destra, ma comunque può essere considerato un risultato soddisfacente. Eseguendo il training su due dataset diversi, non ha senso

indagare come si modifichi la predizione della disparità in funzione della semantica in quanto non sono correlate.

Usando il modello appena allenato, si procede modificando l'input della rete in modo che le immagini RGB in ingresso non siano più tre ma solo due, in quanto in Booster, sezione 2.1, sono presenti le informazioni semantiche necessarie; quindi, basta un'unica esecuzione del context encoder sulla left in input. Dato che il modello allenato su FlyingThings3D e ClearGrasp presenta solo 2 classi semantiche mentre Booster ne possiede 4, bisogna procedere con una modifica in fase di caricamento delle maschere semantiche.

Si è optato di procedere in due modi diversi: si assegna la classe 0 di Booster alla classe 0 e le restanti 3 alla classe 1; si assegnano le classi 0, 1, 2 di Booster alla classe 0 e la classe 3 di Booster alla classe 1.

Nel primo caso tutte le classi delle superfici non lambertiane presenti in Booster sono raggruppate insieme, mentre nel secondo le classi 1 e 2 di Booster vengono unite alla 0 perché contengono superfici molto ampie e non solo oggetti in sé.

La rete precedentemente allenata non ha mai incontrato superfici etichettate come non lambertiane quindi potrebbe avere difficoltà a predire la classe corretta. La classe 3 di Booster, invece, essendoci principalmente oggetti trasparenti e riflettenti e poche superfici, potrebbe risultare più compatibile col training su ClearGrasp.

Per il fine-tuning su Booster vengono scelti i seguenti iperparametri: epoche = 50, max_disp = 256, batch = 6, crop_size = [256,480]. Ogni immagine viene fornita con un fattore di scaling pari ad 1/2 e ad 1/4, dato il numero limitato di immagini in Booster e l'alta risoluzione.

Per vedere come variano i risultati della rete modificando la parte di apprendimento, sono stati eseguiti training di fine-tuning in due stage: nel primo caso sperimentato viene allenata prima solo la parte di rete legata alla disparità poi viene riallenata con entrambe le parti che apprendono; nel secondo caso prima viene allenata solo la prima parte di semantica poi viene riallenata con entrambe le parti che apprendono. Per eseguire un allenamento con solo una delle due parti che apprendono è sufficiente

modificare il codice eliminando il calcolo della funzione di perdita della parte che non deve apprendere.

4.5 Prestazioni

Nella seguente sezione si presentano i risultati ottenuti con la prima fase di training su FlyingThings3D e ClearGrasp e con il fine-tuning presentato nella sezione 4.4 secondo le metriche proposte nella sezione 4.1 e si mostra una seconda tabella, con le metriche solo per la disparità per ogni classe semantica di Booster, sezione 4.1.1.

bad 2.0	**bad 4.0**	**bad 6.0**	**bad 8.0**	**avgerr**	**rms**	**MIoU**	**IoU_0**	**IoU_1**
4.54	3.03	2.39	2.02	1.73	5.00	89.00	99.00	85.00

Tab. 4.2: Risultati FlyingThings3D-ClearGrasp

In Tab. 4.2 sono presenti i risultati di un test condotto con il training svolto su FlyingThings3D e ClearGrasp. Si ottengono valori buoni per la disparità ed eccellenti per la semantica. In particolare, si può notare come la classe 1, corrispondente alla classe degli oggetti trasparenti, venga predetta con un IoU pari a 0.85.

In seguito ai vari training di fine-tuning descritti nella sezione 4.4 si ottengono i seguenti risultati eseguendo un test su Booster:

tipo_sem	bad 2.0	bad 4.0	bad 6.0	bad 8.0	avgerr	rms	MIoU	IoU_0	IoU_1
No_sem_50	46.84	29.27	21.99	17.90	8.37	18.67	-	-	-
Sem1	49.92	32.15	24.40	19.92	9.89	21.89	42.00	41.00	**90.00**
Sem2	44.43	26.72	19.62	15.76	**7.55**	**17.30**	53.00	**99.00**	45.00
Sem1_2stageS	44.96	27.01	19.84	15.89	7.76	17.87	49.00	63.00	78.00
Sem2_2stageS	**44.11**	**25.99**	**18.82**	**14.97**	7.58	17.72	**54.00**	98.00	52.00

Tab 4.3: Risultati Booster 50 epoche per la disparità

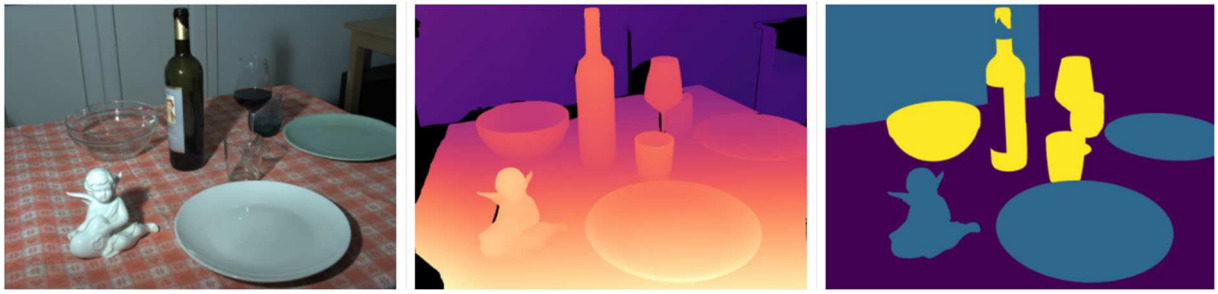

Fig. 4.9: Immagine passiva di test, gt della disparità e maschera semantica (da sinistra)

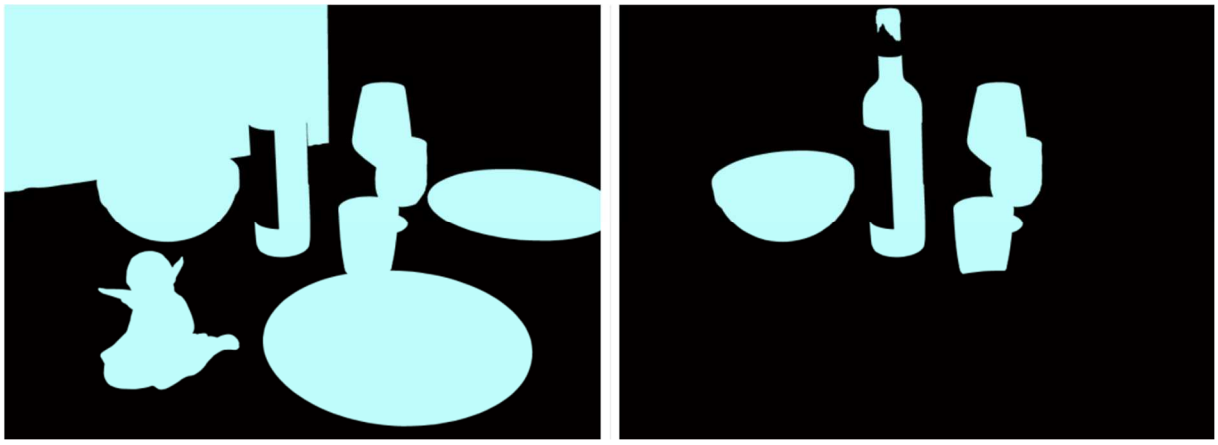

Fig. 4.10: Gt semantica per Sem1, Sem1_2StageS e Sem1_2StageD e maschera semantica per Sem2, Sem2_2StageS e Sem2_2StageD (da sinistra)

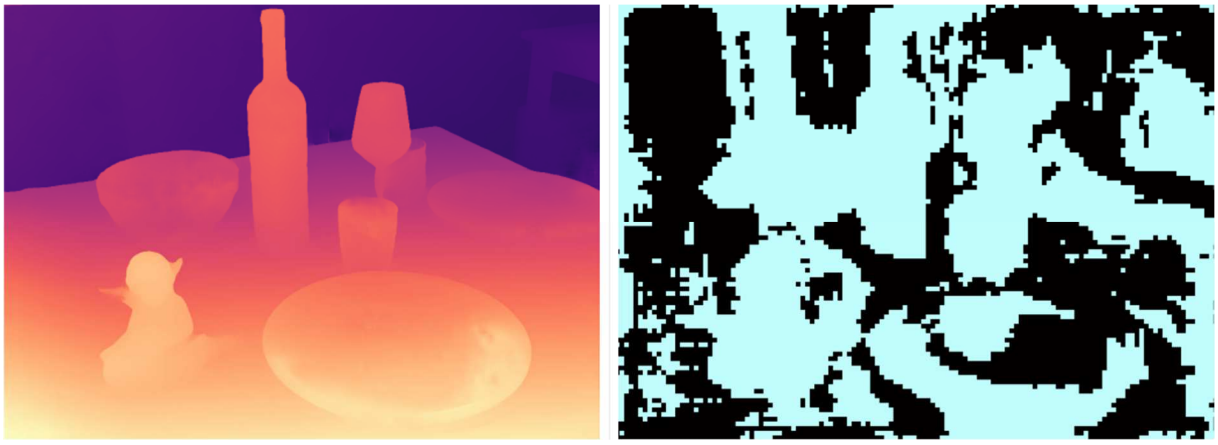

Fig. 4.11: Mappa predetta della disparità e della semantica di Sem1 (da sinistra)

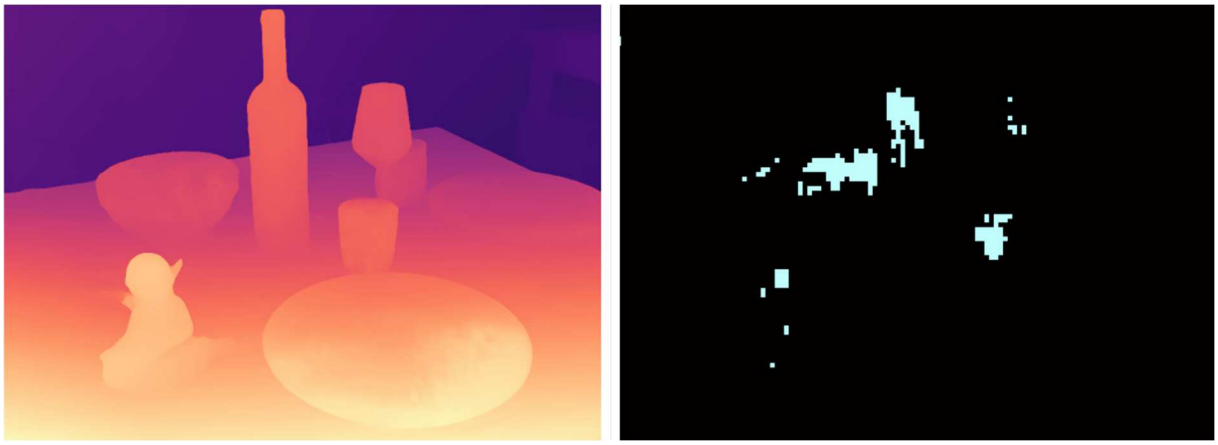

Fig. 4.12: Mappa predetta della disparità e della semantica di Sem2 (da sinistra)

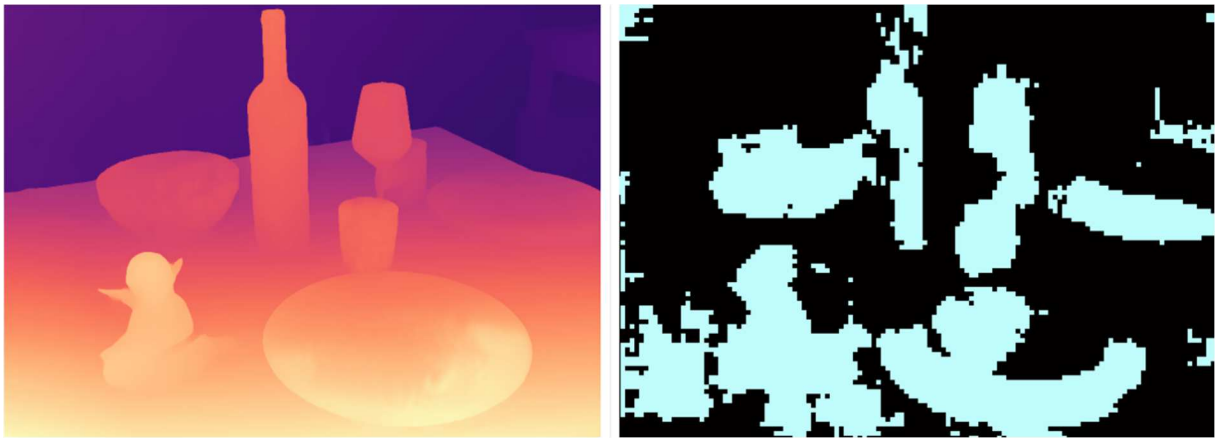

Fig. 4.13: Mappa predetta della disparità e della semantica di Sem1_2StageS (da sinistra)

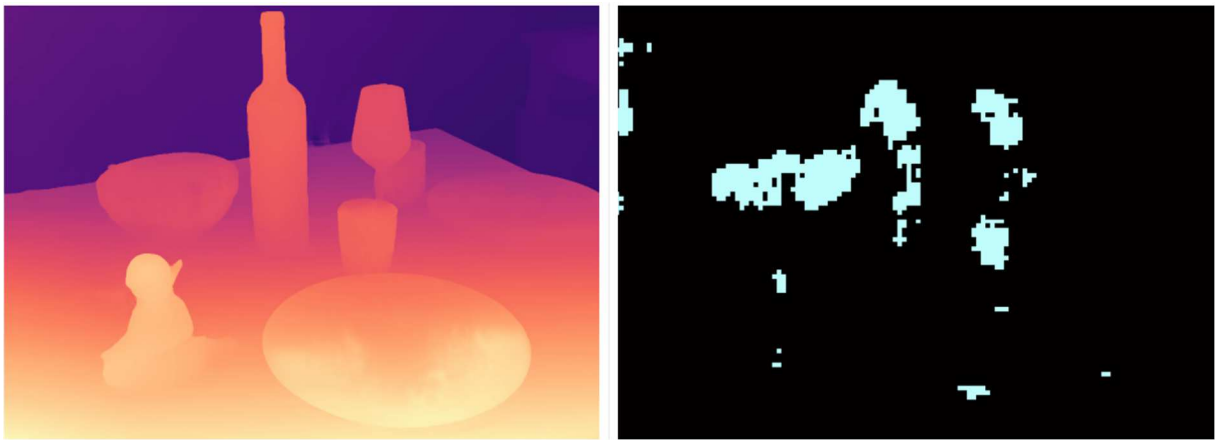

Fig. 4.14: Mappa predetta della disparità e della semantica di Sem2_2StageS (da sinistra)

Come per i precedenti risultati, è fondamentale inserire la riga contenente i valori di test del modello senza modulo semantico per valutare gli esperimenti condotti. Nella prima colonna, con la voce Sem1 si intende un training di fine-tuning su Booster con classi 1,2,3 di Booster accorpate nella classe 1.

Nella seconda riga Sem1, è possibile vedere come la rete non migliori le sue prestazioni con la predizione della semantica: l'avgerr peggiora di 1.52. Il modulo semantico non sembra funzionare particolarmente bene in questo caso, infatti, il test restituisce un MIoU pari a 0,42, che come descritto nella sezione 4.1.2, è inferiore a 0.5 quindi è un risultato non soddisfacente. Come anche descritto dai risultati numerici, in Fig. 4.11 si può vedere qualitativamente come la predizione semantica sia sbagliata. Sono riconoscibili contorni di alcuni oggetti, come ad esempio quello della bottiglia, ma la superfice parzialmente riflettente presente sulla sinistra non viene riconosciuta. In Fig. 4.10 sono presenti due grandi sezioni in cui viene predetta la classe 0 all'interno dell'area riflettente considerata.

Con la voce Sem2 si intende un training di fine-tuning su Booster con classi 0,1,2 accorpate nella classe 0. In questo caso il miglioramento è notevole: l'errore medio di disparità diminuisce di 0.82 rispetto al modello privo di modulo semantico. I risultati

numerici della semantica sono buoni. Però qualitativamente i contorni non sono rispettati (come si vede in Fig. 4.12) e l'area predetta appartenente alla classe 1 è estremamente minore a quella della maschera. Tuttavia, si può ipotizzare che la sola informazione di riconoscimento e posizione delle superfici non lambertiane, senza delimitarne i confini, porti un miglioramento della disparità. È interessante notare come rispetto al caso Sem1 sembra che i valori IoU_0 e IoU_1 si siano invertiti.

Un ulteriore ipotesi è che il miglioramento di Sem2 rispetto a Sem1 può essere motivato come segue: nella classe 3 di Booster sono presenti principalmente solo oggetti trasparenti e riflettenti e non superfici come nelle classi 1 e 2. Il modulo semantico, allenato col primo training su ClearGrasp, è abituato a riconoscere solamente gli oggetti in quanto in quel dataset sono presenti solo quelli.

Con Sem1_2StageS si intende un fine-tuning eseguito in 2 parti: prima è stato allenato solo il modulo semantico sulle classi usate in Sem1 e poi è stato riallenato anche con la predizione della disparità. Si può vedere dai risultati di test che in questo caso è presente un miglioramento nelle metriche della disparità rispetto al modello di riferimento. Però si ottiene un IoU_0 ancora troppo basso e la rete fatica a distinguere una superficie non lambertiana da una che non lo è.

Infine, con Sem2_2StageS si intende un fine-tuning diviso in due parti come per Sem1_2StageS con le classi di Booster divise come per Sem2. Questo caso restituisce valori in fase di test molto simili al caso Sem2 con un miglioramento delle prestazioni semantiche.

Le prestazioni semantiche variano molto tra il caso Sem1 e Sem1_2StageS, che differiscono solamente di 50 epoche di training del modulo semantico; potrebbe significare che l'apprendimento non è sufficientemente robusto nel caso in cui le classi 1, 2, 3 di Booster siano raggruppate nella classe 1.

Osservando i risultati sembrerebbe Sem2 il modello migliore in quanto, per il fine-tuning sono servite solamente 50 epoche e l'errore medio ottenuto per la disparità è il più basso. Però comparando la mappa semantica predetta di Sem2, Fig. 4.11, e quella di Sem2_2StageS, Fig. 4.14, si nota un chiaro miglioramento nella seconda.

Per valutare meglio le prestazioni del modulo semantico si sono calcolate le metriche della disparità per ogni classe di Booster:

tipo_sem	**Classe_Booster**	**bad 2.0**	**bad 4.0**	**bad 6.0**	**bad 8.0**	**avgerr**	**rms**
No_sem_50	0	44.69	24.96	17.19	12.75	5.56	11.28
	1	52.98	33.53	24.31	18.84	6.86	14.51
	2	70.91	52.43	40.70	33.26	9.21	13.00
	3	74.68	56.50	44.93	37.20	**17.88**	**24.69**
Sem1	0	45.08	24.82	16.25	11.59	4.54	10.38
	1	53.90	34.40	25.33	19.98	8.10	16.47
	2	72.32	53.47	41.47	33.44	10.69	16.96
	3	84.67	69.25	57.88	49.26	26.86	34.57
Sem2	0	41.39	20.72	12.67	8.64	3.82	8.28
	1	47.13	27.26	18.47	13.55	5.40	11.99
	2	69.96	49.74	37.29	30.06	9.06	13.24
	3	73.39	55.63	43.54	36.00	19.36	27.34
Sem1_2StageS	0	41.18	20.07	12.32	8.54	3.79	8.33
	1	48.59	27.84	18.48	13.34	**5.27**	**11.56**
	2	69.39	49.91	37.43	30.15	10.28	14.96
	3	73.10	54.88	43.52	36.27	19.62	27.84
Sem2_2StageS	0	**41.09**	**20.04**	**12.00**	**8.15**	**3.60**	**7.97**
	1	**46.39**	**26.30**	**17.59**	**12.59**	5.32	12.27
	2	**69.33**	**49.49**	**36.84**	**29.19**	**8.71**	**12.98**
	3	**71.65**	**54.77**	**43.21**	**35.99**	20.50	28.64

Tab 4.4: Risultati per ogni classe di Booster 50 epoche per la disparità

Come si evince in Tab 4.4:

-Sem2_2StageS è la migliore a predire la disparità per la classe 0 e per la classe 2.

-Sem1_2StageS ottiene l'avgerr minore per la classe 1.

-No_sem_50 è la migliore nella predizione della disparità della classe 3.

Nonostante il modulo semantico, è interessante notare come nel caso Sem2_2StageS la predizione della disparità migliora in ogni classe eccetto nella classe 3 in cui peggiora. Le ipotesi presentate nella sezione Sem2, sotto la tabella precedente, vengono confutate. In tutti e 4 i casi in cui viene inserito il modulo semantico, la predizione della disparità della classe 3 peggiora: si passa da un peggioramento minimo di avgerr di 1,48 di Sem2 a un massimo di 8,98 di Sem1. Spostando l'attenzione sulla Tab. 4.3 il peggioramento dell'avgerr si può motivare con il basso valore dell'IoU_1 nel caso Sem2 e Sem2_2StageS, che corrisponde alla sola predizione della semantica della classe 3. Invece, la predizione semantica delle classi 0, 1, 2, che corrispondono alle classi in cui avviene un miglioramento passando da Sem2 a Sem2_2StageS, risulta eccellente, come confermato dallo IoU_0. Quindi, si può concludere che riconoscere la presenza degli oggetti appartenenti alla classe 3 influenzi la predizione della disparità (avgerr) in due differenti modi: per le classi 0,1,2 migliorandola e per la classe 3 peggiorandola (il paragone è sempre fatto rispetto al caso senza modulo semantico).

Osservando invece le metriche bad x.0, a parte nel caso Sem1, migliorano tutte per tutte le classi. Nel caso della classe 3 la diminuzione di bad x.0 e l'aumento di avgerr dimostra che il numero di errori diminuisce, ma che quelli presenti devono essere maggiori affinchè cresca avgerr.

Come è stato introdotto nella sezione 4.4, per un ulteriore paragone, si è svolto anche il fine-tuning su Booster in due stage per la disparità, ossia 50 epoche in cui viene allenata unicamente la disparità e le successive 50 con l'allenamento completo della rete. Affinché si possa fare un confronto sensato è stato allenata un'ulteriore rete per 100 epoche senza modulo semantico.

tipo_sem	bad 2.0	bad 4.0	bad 6.0	bad 8.0	avgerr	rms	MIoU	IoU_0	IoU_1
No_sem_100	44.76	26.98	19.66	15.71	7.55	17.53	-	-	-
Sem1_2StageD	**43.28**	**25.51**	**18.64**	**14.92**	**7.12**	**16.91**	45.00	56.00	**79.00**
Sem2_2StageD	43.83	26.19	19.23	15.48	7.38	17.28	**55.00**	**98.00**	50.00

Tab 4.5: Risultati di Booster con 100 epoche per la disparità

tipo_sem	Classe_Booster	Bad 2.0	Bad 4.0	Bad 6.0	Bad 8.0	avgerr	rms
No_sem_100	0	41.70	22.03	14.17	9.82	4.42	9.80
	1	48.35	27.17	18.11	13.21	5.78	13.14
	2	68.94	48.63	36.00	28.61	8.18	**12.09**
	3	71.51	52.65	**41.49**	**34.52**	**16.16**	**23.53**
Sem1_2StageD	0	**39.95**	**19.02**	**11.64**	**7.92**	**3.65**	**8.26**
	1	45.62	25.23	16.63	12.04	5.06	12.02
	2	67.79	47.28	35.39	28.55	8.63	12.97
	3	**69.30**	**51.98**	41.67	34.98	18.28	26.62
Sem2_2StageD	0	40.51	20.32	12.63	9.00	3.85	8.60
	1	**44.98**	**24.80**	**16.27**	**11.73**	**5.05**	**11.73**
	2	**67.49**	**46.47**	**34.82**	**28.07**	**8.08**	12.45
	3	71.26	52.47	41.78	35.13	18.86	27.11

Tab 4.6: Risultati per ogni classe di Booster con 100 epoche per la disparità

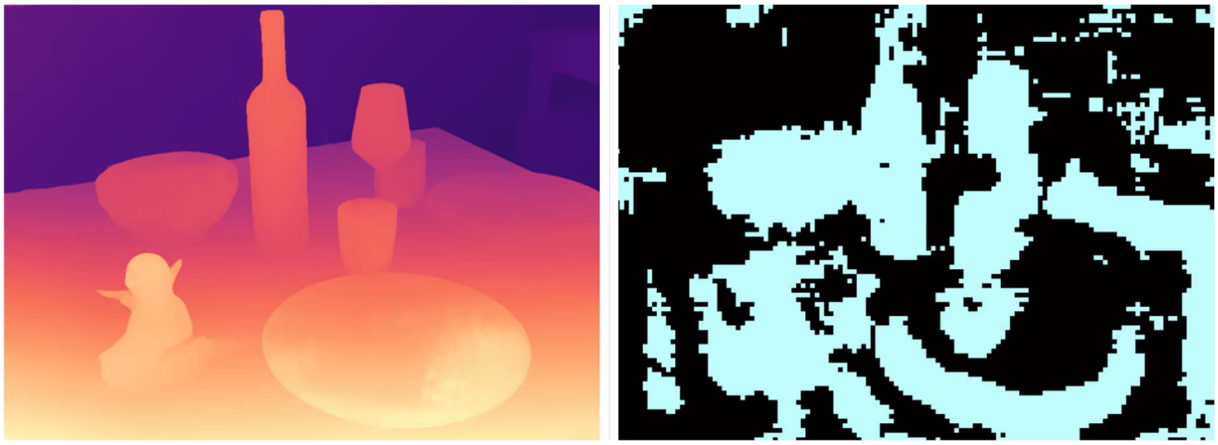

Fig. 4.15: Mappa predetta della disparità e della semantica di Sem1_2StageD (da sinistra)

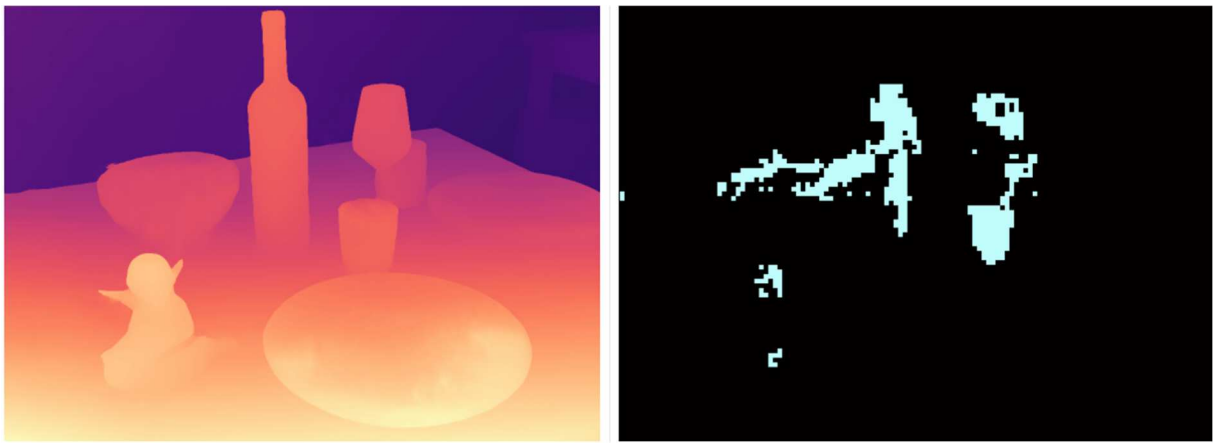

Fig. 4.16: Mappa predetta della disparità e della semantica di Sem2_2StageD (da sinistra)

Nella Tab 4.6 è possibile vedere come Sem1_2StageD si comporti esattamente come Sem1_2StageS, in Tab 4.4. La predizione della disparità, per le classi 0 e 1 di Booster migliora, mentre per le classi 2 e 3 peggiora. Vale lo stesso per Sem2_2StageD e Sem2_2StageS: la predizione delle classi 0, 1, 2 migliora e quella della classe 3 peggiora (sempre rispetto al caso senza modulo semantico). Sia in Tab 4.6 che in Tab 4.4, tra il caso Sem1_2Stage e Sem2_2Stage il migliore nella predizione della disparità della classe 3 è il Sem1_2Stage.

Confrontando la mappa della semantica predetta in Fig. 4.14 e quella in Fig. 4.16, la predizione migliora nel secondo caso. Si potrebbe ipotizzare che si manifesti l'effetto complementare a quello che stiamo cercando: il modello 2StageD viene prima allenato solo sulla predizione della disparità. Quindi sembrerebbe che l'informazione della profondità influenzi la mappa semantica migliorandola [15]. Di conseguenza si possono formulare due ipotesi:

1 - Una migliore informazione semantica ottimizza, a sua volta, la predizione della disparità che continua ad essere allenata nelle ultime 50 epoche dato che in Tab 4.6 si ottengono errori minori rispetto a Tab. 4.4.

2 - La diminuzione d'errore dipende unicamente dal complessivo allenamento di 100 epoche per la predizione disparità, 50 in più del caso 2StageS.

4.6 Conclusioni e futuri lavori

Dagli esperimenti condotti su RAFT-Stereo, si può concludere che la predizione della disparità viene influenzata inserendo un modulo semantico. Dall'esame dei risultati su ogni classe di Booster, si ricava che, cercando di riconoscere unicamente le superfici non lambertiane, nel migliore dei casi si ottiene un'ottimizzazione parziale: la predizione della disparità migliora per oggetti lambertiani e per quelli con una riflettenza o trasparenza bassa o media, mentre peggiora per le superfici estremamente riflettenti e trasparenti.
In generale, invece, si può affermare che, in tutti i modelli esaminati eccetto Sem1, l'informazione semantica delle superfici non lambertiane, anche se grezza, influenza positivamente la disparità in quanto l'avgerr complessivo diminuisce.
Questo è solo un primo lavoro sull'implementazione semantica di RAFT-Stereo. Esistono molti tipi di modulo semantico ed è possibile che se ne trovino alcuni con una maggiore compatibilità con la rete. Un'altra possibile modifica da effettuare è non lasciare che la semantica influenzi passivamente la disparità, ma influenzarla in modo attivo inserendo un blocco finale per un miglioramento.
Un possibile lavoro futuro è la modifica del modulo semantico presentato nella sezione 3.5 con diversi valori di dilation, dato che l'avgerr risulta buono.
Un ulteriore cambiamento potrebbe essere eseguito nel caso Sem2 (in cui nella classe 1 risiede solo la classe 3 di Booster) bilanciando la classe 1 tramite data augmentation e successivamente pesando l'errore di questa classe nella funzione di perdita.
Un'ultima modifica possibile potrebbe essere valutare la disparità solo per quei pixel dove è stata correttamente classificata la trasparenza (classe 1 predetta correttamente), confrontandola con quella predetta da no_sem per gli stessi pixel.
Come ultima considerazione è importante sottolineare l'importanza dei dati per la fase di allenamento: Booster è un dataset piccolo per ottenere una conoscenza approfondita della realtà a livello semantico. Una soluzione per migliorare il rendimento della rete potrebbe essere l'estensione del dataset con ulteriori scene. La maggiore difficoltà è l'acquisizione di grandi moli di dati contenenti superfici lambertiane da fornire in

input; quindi, si potrebbe investigare su metodologie tali per cui avere la gt della disparità e della semantica in modo semplice e rapido.

Bibliografia

[1] L.Lipson, Z. Teed e J.Deng, RAFT-Stereo: Multilevel Recurrent Field Transforms for Stereo Matching, 2021

[2] M.Poggi, F.Tosi, K.Bastos, P.Mordohai, S.Mattoccia: On The Synergies between Machine Learning and Binocular Stereo for Depth Estimation from Images: a Survey, 2021

[3] P. Zama Ramirez, F.Tosi, M.Poggi, S.Salti, S.Mattoccia, L.Di Stefano, Open Challenges in Deep Stereo: the Booster Dataset. (2022). Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[4] http://synthia-dataset.net/

[5] N.Mayer, E.Ilg, P.Hausser, P.Fischer, D.Cremers, A.Dosovitskiy, T.Brox, A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation, 2015

[6] S.S.Sajjan, M.Moore, M.Pan, G.Nagaraja, J.Lee, A.Zeng, S.Song, ClearGrasp: 3D Shape Estimation of Trasparent Objects for Manipulation, 2019

[7] L.Zhu, A.Mousavian, Y.Xiang, H.Mazhar, J. van Eenbergen, S.Debnath, D.Fox, RGB-D Local Implicit Function for Depth Completion of Transparent Objects, 2021

[8] X.Liu, R.Jonschkowski, A.Angelova, K.Konolige, KeyPose: Multi-View 3D Labeling and Keypoint Estimation for Trasparent Objects, 2020

[9] H.Fang, H.Fang, S.Xu, C.Lu, TransCG: A Large-Scale Real-World Dataset for Trasparent Object Depth Completion and Grasping, 2022

[10] X.Chen, H.Zhang, Z.Yu, A.Opipari, O.C.Jenkins, ClearPose: Large-scale Transparent Object Dataset and Benchmark, 2022

[11] H.Xu, Y.R.Wang, S.Eppel, A.Aspuru-Guzik, F.Shkurti, A.Garg: Seeing glass: Joint point cloud and depth completion for transparent objects, 2021

[12] X.Liu, R.Jonschkowski, A.Angelova, K.Konolige,: Keypose: Multi-view 3d labeling and keypoint estimation for transparent objects. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020

[13] X.Liu, S.Iwase, K.M.Kitani: Stereobj-1m: Large-scale stereo image dataset for 6d object pose estimation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021

[14] E.Xie, W.Wang, W.Wang, M.Ding, C.Shen, P.Luo: Segmenting transparent objects in the wild. In: European conference on computer vision, 2020

[15]V.S.Victor, P.Neigel, Survey on Semantic Stereo Matching / Semantic Depth Estimation, 2021

[16] H.Zhao, J.Shi, X.Qi, X.Wang, J.Jia, Pyramid Scene Parsing Network, 2017

[17] L.Chen, G.Papandreou, I.Kokkinos, K.Murphy, A.L.Yuille, DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, 2017