ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

in

Sistemi Embedded Riconfigurabili M

Integrazione di dati di disparità sparsi in algoritmi per la visione stereo basati su deep learning

CANDIDATO

Davide Pallotti

RELATORE

Prof. Stefano Mattoccia

CORRELATORI

Dott. Matteo Poggi

Dott. Fabio Tosi

Anno Accademico 2017/18

Sessione II

PREMESSA

Uno dei problemi di interesse nell'ambito della computer vision è quello della visione stereo, ovvero l'estrazione di informazioni di profondità da una scena a partire da una vista sinistra e una vista destra catturate da una coppia di camere. Per comprendere il ruolo che la visione stereo riveste, si pensi ad esempio, tra i molteplici domini di applicazione, ai sistemi di assistenza alla guida e al crescente interesse per i sistemi di guida autonoma, in cui è richiesta la conoscenza in tempo reale della posizione esatta di tutti gli oggetti circostanti. Dispositivi RADAR e i più precisi LiDAR consentono sì di ottenere risultati tendenzialmente più affidabili e accurati quando necessario, ma sono strumenti decisamente più complessi e costosi e rappresentano quindi una soluzione non sempre praticabile o vantaggiosa.

Il problema della visione stereo si riduce a determinare, per ogni punto visibile in un'immagine, la posizione dello stesso punto nell'altra immagine. La soluzione non è banale, e presenta ulteriori criticità in presenza di occlusioni, aree povere di texture, riflessi, condizioni di luminosità particolari, strutture sottili. Tuttavia, il problema si semplifica nel caso di immagini rettificate, nel qual caso due punti corrispondenti risultano traslati solo lungo l'asse orizzontale. L'entità della traslazione è detta disparità, ed è inversamente proporzionale alla profondità: note le caratteristiche delle due camere, le due grandezze sono quindi equivalenti. Scelta una vista di riferimento e calcolata la disparità per ogni suo punto, la mappa di disparità che ne risulta contiene le informazioni di profondità della scena.

Un ampio spettro di algoritmi sono stati progettati per la costruzione di mappe di disparità, alcuni più improntati alla velocità di esecuzione ma meno precisi, altri più accurati ma anche molto onerosi in termini computazionali. Tra questi spicca ad esempio l'algoritmo SGM, che trova un buon compromesso tra accuratezza ed efficienza, in particolare nella sua implementazione denominata rSGM, in grado di ottenere un frame-rate accettabile anche su CPU general purpose. SGM produce una mappa di disparità minimizzando una funzione di costo costruita a partire da un volume dei costi, che misura la somiglianza degli intorni di punti potenzialmente corrispondenti per numerosi valori di disparità.

L'approccio alla visione stereo, nonché a numerosissimi altri problemi di computer vision, è stato rivoluzionato dal successo delle reti neurali convoluzionali (CNN), che, vagamente ispirate alla struttura della nostra corteccia visiva, si sono rivelate particolarmente abili nello svolgere attività di percezione e riconoscimento

tipicamente intuitive per l'uomo ma estremamente complesse per un calcolatore. La conoscenza della geometria stereo e l'esperienza degli algoritmi stereo tradizionali ha portato numerose CNN per la visione stereo a presentare una struttura generale analoga: le due immagini vengono in una prima fase processate separatamente, e le feature così estratte vengono combinate in un volume dei costi che la seconda parte della rete utilizza poi per produrre la mappa di disparità. Un esempio di CNN per la visione stereo con questa architettura è GC-Net, le cui dimensioni relativamente ridotte, in termini di numero di parametri addestrabili, la rendono particolarmente adatta alla sperimentazione.

Obiettivo di questa tesi è integrare dati di disparità sparsi, suggeriti da una fonte esterna, in un algoritmo stereo, prevalentemente una CNN, con l'intento di migliorarne l'accuratezza. Dati sparsi di disparità possono essere ad esempio prodotti da un sensore di profondità, come un LiDAR a bassa risoluzione, nel qual caso si tratta di dati precisi, o da un secondo algoritmo, che magari presenti punti di forza differenti da quello in esame. L'idea che proponiamo è quella di utilizzare i dati noti associati a punti sparsi per andare a modulare opportunamente i valori corrispondenti a quegli stessi punti nel volume dei costi, nella speranza che l'algoritmo riesca a generalizzare le informazioni all'intera mappa di disparità.

In una prima fase sperimenteremo estensivamente questo approccio sulla CNN di nostra scelta, GC-Net. Dapprima faremo uso di disparità estratte dalla ground truth, campionata casualmente, aspettandoci sicuramente un miglioramento: ciò può apparire artificioso, ma ci permetterà di verificare l'applicabilità dell'approccio, e può in ogni caso simulare l'impiego di un sensore di profondità. Dopodiché estrarremo invece le disparità dagli output di SGM e rSGM, ancora in maniera casuale, chiedendoci se ciò sia già sufficiente per ottenere un primo piccolo miglioramento rispetto alla sola GC-Net.

In un secondo momento saggeremo l'applicabilità di questo stesso metodo a un algoritmo tradizionale, rSGM, agendo ancora sul suo volume dei costi. Questa volta utilizzeremo soltanto la ground truth come fonte di disparità da campionare, ma oltre alla modulazione proveremo anche l'effetto di sostituire direttamente i costi con valori opportuni.

Infine riprenderemo l'idea di fornire a GC-Net l'aiuto di rSGM, cosicché possa limitare i propri errori nelle aree in cui presenta maggiori indecisioni. Anziché campionare l'output di rSGM casualmente, però, adotteremo un approccio meno naïve, scegliendo soltanto i punti che risultano più promettenti rispetto a una misura di confidenza calcolata con una seconda rete neurale, CCNN.

INDICE

Premessa **3**

Indice **5**

1 **Introduzione** **7**

1.1 Computer vision e visione stereo 7

1.2 Machine learning e reti neurali convoluzionali 13

2 **GC-Net con punti a disparità nota** **21**

2.1 CNN per la visione stereo: DispNet, GC-Net 21

2.2 GC-Net in dettaglio 24

2.3 Dataset per training e testing 27

2.4 Modifica di GC-Net 29

2.5 Training e risultati con disparità estratte dalla ground truth 32

2.6 Training e risultati con disparità estratte da SGM e rSGM 38

3 **rSGM con punti a disparità nota** **41**

3.1 Algoritmi tradizionali per la visione stereo: SGM e rSGM 41

3.2 Modifica di rSGM con sostituzione dei costi 43

3.3 Test e risultati con sostituzione dei costi 44

3.4 Modifica di rSGM con modulazione dei costi 46

3.5 Test e risultati con modulazione dei costi 47

4 **GC-Net con punti a disparità con confidenza elevata** **53**

4.1 Una misura di confidenza per le mappe di disparità: CCNN 53

4.2 Integrazione di CCNN in GC-Net modificata 55

4.3 Training e risultati 57

5 **Conclusioni** **63**

6 **Bibliografia** **65**

1 INTRODUZIONE

1.1 Computer vision e visione stereo

Per *computer vision* si intende la disciplina che si occupa di consentire a un elaboratore di estrarre informazioni da immagini, sequenze di immagini o video digitali e di acquisirne una comprensione di alto livello. Le applicazioni sono innumerevoli: in ambito industriale per il controllo dei processi automatizzati e il rilevamento di difetti; nell'automotive per l'assistenza alla guida e la guida autonoma; nei sistemi di controllo del traffico, ad esempio per la lettura delle targhe delle automobili; nella sorveglianza per la rilevazione del movimento e il tracciamento di oggetti; nei sistemi di riconoscimento facciale o delle impronte digitali; in ambito medico per assistere operazioni chirurgiche di precisione; per l'organizzazione di collezioni multimediali e la ricerca basata sul contenuto; più in generale per il riconoscimento e la classificazione di oggetti; e l'elenco potrebbe continuare a lungo.

È possibile rintracciare in un processo di computer vision tre fasi principali: l'acquisizione delle immagini; l'estrazione di feature di basso livello da parti dell'immagine tramite algoritmi e tecniche matematiche; l'analisi e la comprensione di alto livello delle immagini sfruttando le feature estratte al passo precedente, da cui eventualmente possa scaturire una decisione. Mentre la seconda e la terza fase dipendono fortemente dall'applicazione di interesse, la fase di acquisizione, per quanto possa fare uso di dispositivi differenti e più o meno avanzati, è riconducibile

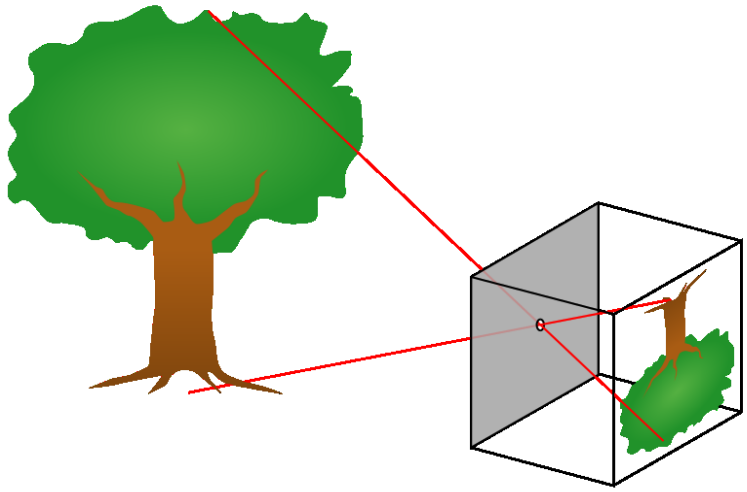

Figura 1.1 – Schema di una pinhole camera.

alla semplice astrazione della *pinhole camera*, o camera oscura, che, seppur difficilmente utilizzabile per catturare immagini, ci permette di definire la geometria di base dell'acquisizione di immagini.

Una pinhole camera è di fatto costituita da una scatola chiusa con un piccolo foro su una faccia attraverso il quale passano raggi luminosi che vanno a proiettare un'immagine, capovolta e rovesciata, sul lato interno della faccia opposta. Geometricamente, quindi, ogni punto dell'immagine è ottenuto intersecando con il piano di proiezione la retta che passa per il corrispondente punto della scena e per il foro.

Sia ora C il foro della camera oscura, o centro ottico; c la sua proiezione sul piano dell'immagine; f la distanza tra il centro ottico e il piano dell'immagine, ovvero la distanza fra C e c, detta distanza focale. Definiamo il *camera reference frame* (CRF) come il sistema di riferimento cartesiano con origine nel centro ottico C in cui l'asse z è perpendicolare al piano dell'immagine e di conseguenza i cui assi x e y sono ad esso paralleli. Volendo possiamo inoltre ipotizzare, per eliminare fin da subito il problema del capovolgimento dell'immagine, che il piano dell'immagine si trovi davanti al centro ottico, sempre a distanza f. Adesso, un punto $M(x, y, z)$ nel camera reference frame è proiettato sul punto dell'immagine di coordinate $m(u, v)$, con

$$u = \frac{f}{z}x, \qquad v = \frac{f}{z}y,$$

dove gli assi u e v sono paralleli a x e y rispettivamente.

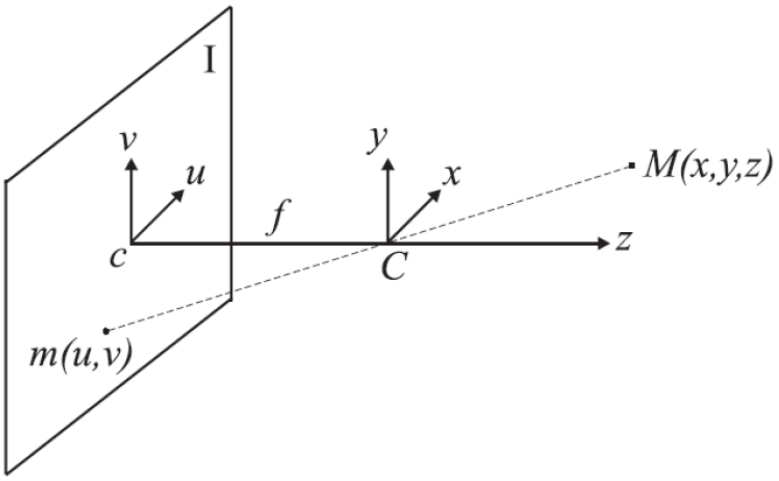

Figura 1.2 – Il Camera Reference Frame e le coordinate dell'immagine.

È evidente che nel passaggio dalle tre alle due dimensioni si ha una perdita di informazione: se ad ogni punto della scena corrisponde un solo punto nelle coordinate dell'immagine, ad ogni punto m dell'immagine corrispondono infiniti punti della scena, tutti quelli che giacciono sulla retta passante per m e per il centro ottico. Di conseguenza un solo punto di vista di una scena non è sufficiente per ricostruirne la struttura tridimensionale. Il cervello umano è in grado di stimare la distanza di oggetti anche nel caso monoculare, sfruttando indizi come linee di prospettiva, ombre, pattern e oggetti noti. Ma nel caso generale il problema resta ambiguo, perfino per l'occhio umano, se si pensa ad esempio alle illusioni ottiche di profondità.

Un particolare ambito della computer vision è proprio quello della visione stereo, che consiste nel ricostruire le informazioni tridimensionali dei punti visibili di una scena avendo a disposizione non una ma due immagini della scena, ottenute da due camere distinte, che producono una vista sinistra e una vista destra. In generale, l'idea è quella di individuare nelle due viste le due proiezioni di uno stesso punto della scena: dalle posizioni delle due proiezioni, e nota la collocazione delle due camere, è possibile risalire alla distanza del punto da ogni camera. Ma il problema di individuare punti corrispondenti nella due viste è tutt'altro che banale.

La geometria che regola la visione stereo è la geometria epipolare. Si faccia riferimento alla Figura 1.3. Siano O_L e O_R i centri ottici rispettivamente della camera sinistra e destra e siano e_L ed e_R, detti epipoli, rispettivamente la proiezione di O_R sul piano dell'immagine sinistra e la proiezione di O_L sul piano dell'immagine destra (in altre parole, gli epipoli sono ottenuti intersecando la retta che passa per i

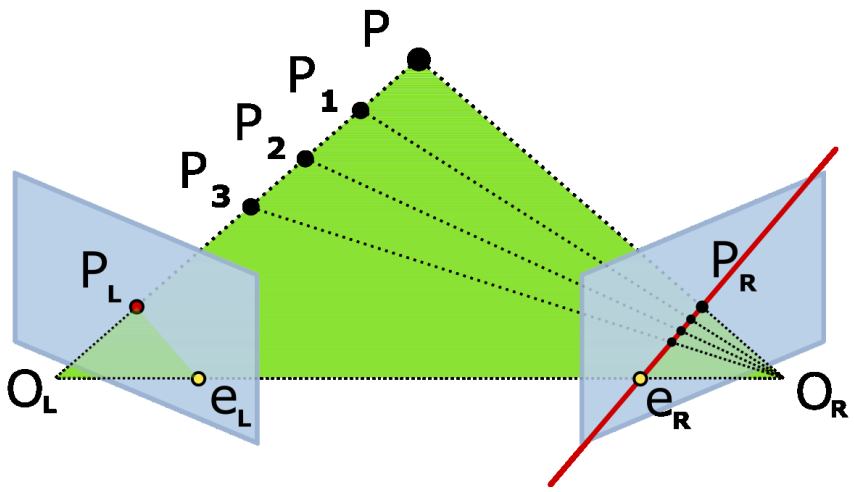

Figura 1.3 – Geometria epipolare e vincolo epipolare.

due centri ottici con i due piani dell'immagine). I punti O_L, e_L, e_R, O_R giacciono tutti sulla stessa retta tridimensionale.

Sia ora P un punto della scena, sia P_L la sua proiezione sul piano dell'immagine sinistra e sia P_R la sua proiezione sul piano dell'immagine destra. Siano inoltre $P_1, P_2, \ldots$ altri punti appartenenti alla retta che passa per P e O_L: la proiezione di tali punti sull'immagine sinistra è sempre P_L, mentre le loro proiezioni sull'immagine destra giacciono tutte sulla retta passante per P_R ed e_R. Questa proprietà è detta vincolo epipolare. Una retta come questa, passante per un epipolo, è detta retta epipolare.

Non solo: dati due punti X e Y che, proiettati sull'immagine sinistra, giacciano entrambi su una stessa retta epipolare (passante per X_L, Y_L ed e_L), si ha che anche le loro proiezioni X_R e Y_R sull'immagine destra giacciono su una stessa retta epipolare (nell'immagine destra). Due rette epipolari così definite sono dette rette epipolari coniugate. Le due proiezioni corrispondenti sinistra e destra di uno stesso punto della scena giacciono su rette epipolari coniugate.

Il vincolo epipolare consente dunque di ridurre il problema di individuare punti corrispondenti nelle due viste dalle due dimensioni a una dimensione, perché è possibile limitare la ricerca lungo rette epipolari coniugate. Ma resta il problema di determinare la retta epipolare coniugata a quella nell'altra vista. Ciò è immediato nella situazione ideale in cui rette epipolari coniugate siano collineari e parallele all'asse orizzontale. Questo accade nel caso in cui le camere abbiano la stessa distanza focale, gli assi dei due CRF siano paralleli e gli assi x collineari, e i due piani dell'immagine siano complanari. Sotto queste condizioni, schematizzate in Figura 1.4, i due CRF differiscono solo per una traslazione lungo l'asse x di lunghezza b, la distanza tra i due centri ottici, detta *baseline*. Anche se in un sistema reale è impossibile ricreare esattamente queste condizioni, è sempre possibile trasformare geometricamente le immagini come se fossero state acquisite in questo setup ideale: tale procedimento è detto rettificazione, e se le camere non introducono distorsione una trasformazione lineare risulta sufficiente.

In questo setup, inoltre, note le coordinate nelle viste destra e sinistra di due punti corrispondenti, è immediato risalire alla posizione del punto nella scena, e in particolare alla sua distanza. Infatti, essendo

$$\begin{cases} u_L = x_L \cdot \dfrac{f}{z} \\ u_R = x_R \cdot \dfrac{f}{z} \end{cases}$$

si ottiene

$$u_L - u_R = (x_L - x_R) \cdot \frac{f}{z} = b \cdot \frac{f}{z},$$

da cui

$$z = \frac{b \cdot f}{u_L - u_R} = \frac{b \cdot f}{d},$$

dove $d = u_L - u_R$ è detta disparità. Essa rappresenta la differenza tra le coordinate orizzontali di due punti corrispondenti nelle due viste ed è tanto maggiore quanto più vicino è il punto della scena. Se sono note la distanza tra i centri ottici e la distanza focale, conoscere la disparità equivale quindi a conoscere la profondità del punto nella scena. La coordinata y è banale essendo $y_L \cdot \frac{f}{z} = v_L = v_R = y_R \cdot \frac{f}{z}$.

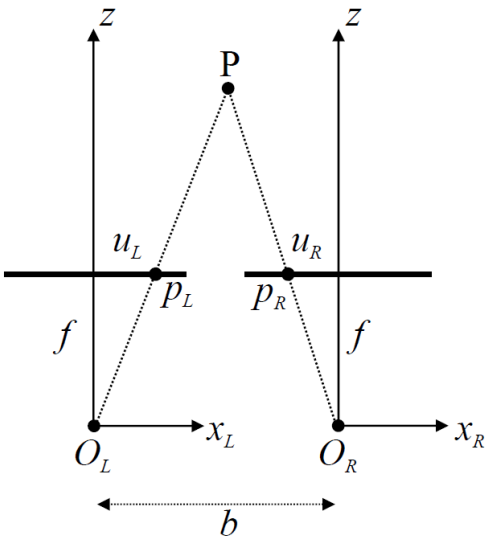

Figura 1.4 – Setup ideale che produce una coppia di immagini rettificate.

Per ricostruire la struttura tridimensionale della scena resta dunque da risolvere il problema di determinare, scelta una vista di riferimento, ad esempio quella sinistra, la disparità di ogni suo punto, ricercando il punto corrispondente nell'altra vista alla stessa coordinata verticale v. L'idea di base è che gli intorni di due punti corrispondenti sono simili nelle due immagini, ma ciò è complicato da numerosi fattori: a causa di occlusioni dovute ad oggetti più vicini non tutti i punti in una vista hanno un corrispondente nell'altra vista; nelle zone di discontinuità di profondità, in particolare lungo i contorni degli oggetti, gli intorni di due punti corrispondenti possono essere piuttosto diversi; lo stesso vale, sebbene per motivi differenti, in condizioni di luce particolari o in presenza di riflessi; in zone a bassa

texture o con pattern ripetitivi, al contrario, molteplici punti possono avere intorni simili a quello di un punto dato.

Gli algoritmi classici per il calcolo della disparità possono in generale seguire due approcci diversi a seconda che si ricerchi maggiore accuratezza o maggiore velocità di esecuzione. I metodi locali si basano solo sull'intorno dei singoli punti, ottenendo una bassa accuratezza lungo i bordi o in aree con poca texture ma garantendo maggiore efficienza. I metodi globali partono dagli intorni dei punti per risolvere un problema di minimizzazione di costo (ad esempio penalizzando le discontinuità nella disparità) che prenda viceversa in considerazione tutti i punti dell'immagine, fornendo una maggiore accuratezza al prezzo di una complessità computazione più alta, talvolta intrattabile. Altri metodi, come SGM (Semi-Global Matching), che vedremo nel capitolo 3, seguono un approccio tendenzialmente globale, ma riducendo lo spazio di ricerca a sufficienza per ottenere velocità di esecuzione accettabili senza sacrificare troppo l'accuratezza.

L'insieme dei valori di disparità calcolati su un'immagine è detta mappa di disparità. Una mappa di disparità è solitamente rappresentata a sua volta come un'immagine, in cui a disparità diverse corrispondono colori diversi secondo una mappa di colori che ad esempio passi da colori più caldi a più freddi o da colori più chiari a più scuri al diminuire della disparità.

Figura 1.5 – Un'immagine sinistra e la relativa mappa di disparità LR tratte dal dataset Middlebury (la vista destra è omessa). In questa mappa di colori il blu scuro (non presente) indicherebbe una disparità nulla, mentre all'estremo opposto il colore rosso una disparità massima di 127. Le aree nere indicano l'assenza di informazioni di disparità.

Si noti che, nel caso stereo, le mappe di disparità possibili sono due, a seconda che si prenda come riferimento la vista sinistra (mappa LR) o la vista destra (mappa RL). Spesso solo una della due mappe viene generata, tendenzialmente quella LR, ma disporre di entrambe consente ad esempio di applicare il *left-right consistency*

check, che invalida disparità inconsistenti tra le due mappe. Nello specifico, se secondo la mappa LR p_R è il corrispondente di p_L, allora secondo la mappa RL p_L deve essere il corrispondente di p_R, viceversa si ha un'inconsistenza.

1.2 Machine learning e reti neurali convoluzionali

I notevoli progressi nel campo del machine learning, inteso come insieme di tecniche che consentono a un elaboratore di modificare il proprio comportamento e migliorare le proprie capacità nello svolgere un determinato compito "imparando" da grandi quantità di dati anziché essere programmato esplicitamente con regole statiche, come è invece il caso degli algoritmi tradizionali, hanno modificato radicalmente l'approccio a diversi problemi di computer vision.

In effetti, l'analisi di immagini è uno degli ambiti in cui il machine learning brilla maggiormente nel confronto con algoritmi basati su regole, tanto da essere spesso uno dei punti di ingresso al suo apprendimento. Si pensi al problema del riconoscimento di un oggetto in un'immagine: se un infante è in grado di svolgere un tale compito senza alcuna difficoltà, progettare un algoritmo tradizionale per fare lo stesso è tutt'altro che semplice. Ciò è dovuto al fatto che un'immagine è già in sé una codifica estremamente complessa, variegata, rumorosa e di bassissimo livello dei concetti che rappresenta, e specificare un insieme di regole che descrivano un determinato pattern nel caso generale richiede tecniche matematiche avanzate, ove disponibili, e di accettare comunque un certo grado di inaccuratezza.

Se un sistema basato sul machine learning impara a svolgere il proprio compito basandosi sui dati, ciò significa che un tale sistema dovrà attraversare una fase potenzialmente lunga di training su un dataset preferibilmente ampio prima di raggiungere l'accuratezza desiderata. In generale, più complesso è il modello da addestrare, in termini del numero di parametri da apprendere e affinare, più lungo sarà il training e maggiori dati richiederà. Tra le modalità di addestramento distinguiamo due grandi categorie, l'apprendimento supervisionato e l'apprendimento non supervisionato. Nel primo caso ai dati utilizzati per il training è associata la *ground truth*, ovvero l'output atteso, e l'obiettivo è far imparare al sistema un'approssimazione della relazione tra input e output. Nel secondo caso gli output "corretti" non sono noti, e il sistema deve essere in grado di estrarre autonomamente una struttura dai dati che riceve in input.

Tra le varie tecniche di machine learning note, le reti neurali artificiali sono sicuramente tra quelle di maggior interesse, per la loro capacità di adattarsi con successo a problemi di natura diversa. Le reti neurali si ispirano vagamente alla struttura dell'encefalo animale e non è un caso che si rivelino particolarmente

efficaci nel risolvere problemi così complessi per un algoritmo tradizionale ma apparentemente così semplici per il nostro cervello, quali appunto l'elaborazione di immagini. In effetti, le reti neurali sono ormai diventate il metodo standard per la risoluzione di molti problemi di computer vision. Abbiamo già citato il riconoscimento di oggetti, ma si rifletta ora sul nostro problema della visione stereo e del matching di punti corrispondenti di due viste distinte: se il nostro cervello combina costantemente le due immagini osservate dai nostri occhi in un'unica immagine sovrapponendo aree omologhe in maniera del tutto naturale e senza un apparente sforzo, possiamo pensare di poter ottenere buoni risultati anche con una rete neurale artificiale opportunamente architettata.

Una rete neurale è costituita da una combinazione anche molto complessa di semplici componenti di base, i neuroni. Un neurone artificiale riceve in ingresso n input $x_1, \dots, x_n$ ed è modellato come la combinazione tra una trasformazione affine dei suoi input e una funzione di attivazione f non lineare. Precisamente, un neurone produce in output

$$f\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

dove $w_1, \dots, w_n$ sono detti pesi e b è detto *bias*. I neuroni sono combinati tra loro portando l'output di ognuno all'input di altri (o in uscita alla rete). La funzione di attivazione, non lineare, è fondamentale affinché la rete possa produrre un output che non sia una semplice combinazione lineare degli input. Funzioni di attivazione molto comuni, la cui scelta dipende comunque solitamente dalla tipologia e dall'ambito di utilizzo della rete, sono ad esempio la *rectified linear unit* (ReLU)

$$f(x) = \max(0, x)$$

e la funzione sigmoidea

$$\sigma(x) = \frac{1}{1 + e^{-t}}.$$

A partire da questi componenti di base, le reti neurali possono assumere strutture estremamente complesse e diverse fra loro, e alcune di queste si sono rivelate più efficaci di altre a seconda del problema da risolvere. Nella struttura più classica i neuroni sono organizzati in una successione di strati, o *layer*, e gli output di tutti i neuroni di un layer sono portati in input a tutti i neuroni del layer successivo, come esemplificato in Figura 1.6. I neuroni del primo strato ricevono in ingresso gli input della rete, i neuroni dell'ultimo producono in uscita l'output della rete. Layer di questo tipo sono detti *fully connected*.

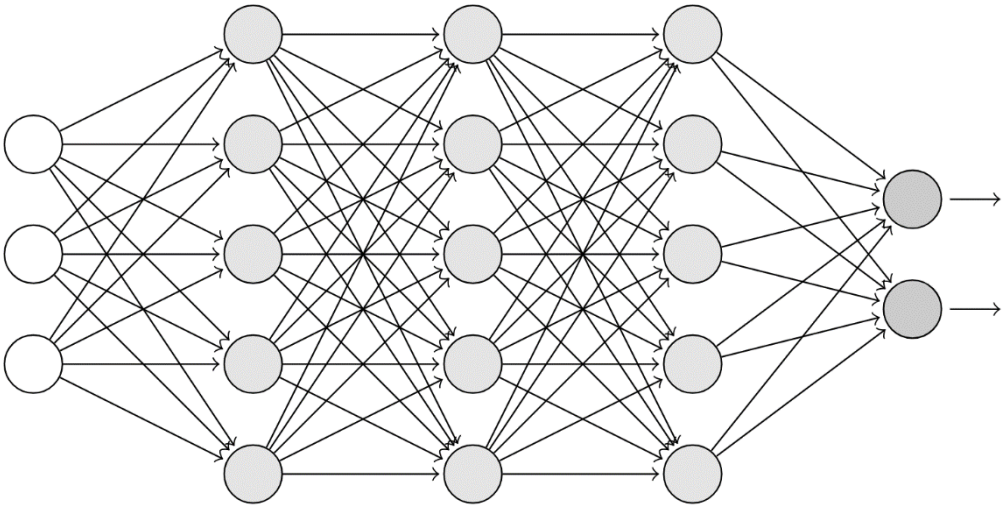

Figura 1.6 – Schema di rete neurale fully connected.

Questa tipologia di rete ci permette di comprendere come avviene l'addestramento di una rete neurale nel caso supervisionato. Addestrare una rete significa modificare opportunamente i pesi e il bias di ogni neurone affinché l'output della rete approssimi nella maniera migliore possibile l'output atteso a fronte di ogni possibile input. Questo procedimento può essere pensato come un problema di ottimizzazione, in cui si vuole minimizzare una funzione di costo $C\big(y(x), t(x)\big)$ definita a partire dall'output y della rete e dalla ground truth t: un esempio tra i tanti di funzione di costo è l'errore quadratico medio $MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2$. Naturalmente lo spazio di ricerca è estremamente grande, perché ogni parametro della rete rappresenta una diversa dimensione, perciò l'idea generale di un algoritmo di training è quella di partire da un punto iniziale, corrispondente a un'inizializzazione casuale dei parametri della rete (che segua comunque opportuni criteri per la media e la deviazione standard), e muoversi ad ogni passo verso punti a costo minore, seguendo l'opposto del gradiente della funzione di costo $-\nabla C$ per una distanza che dipende dalla *learning rate* η scelta: questa tecnica è detta discesa del gradiente (*gradient descent*). Le implementazioni della discesa del gradiente sono numerose, ciascuna della quali cerca di bilanciare la velocità di avvicinamento alla soluzione, la possibilità di sfuggire a minimi locali e la capacità di convergere effettivamente a una soluzione. In generale, però, il problema si riduce a calcolare, per ogni parametro v_k della rete, la derivata parziale della funzione di costo rispetto a quel parametro, $\frac{\partial C}{\partial v_k}$, e ad aggiornare il parametro di conseguenza. Nel caso più semplice v_k sarà aggiornato secondo

$$v_k \leftarrow v_k - \eta \frac{\partial C}{\partial v_k}.$$

Il calcolo del gradiente avviene con il cosiddetto algoritmo di *backpropagation*, in cui l'errore calcolato sull'uscita è propagato dagli ultimi ai primi livelli e utilizzato per determinare le derivate parziali sfruttando i risultati già calcolati ai livelli successivi (ma processati prima).

Le reti neurali fully connected sono concettualmente semplici, ma il numero di parametri da addestrare cresce velocemente con il numero di layer e soprattutto con il numero di neuroni in ognuno, essendo il numero di connessioni tra due layer consecutivi pari al prodotto delle loro dimensioni. Consideriamo ancora il caso di nostro interesse dell'analisi di immagini. Il solo numero di input è pari al numero di pixel dell'immagine, eventualmente moltiplicato per il numero di canali. Se supponiamo ora di avere anche pochi layer fully connected con numero di neuroni dello stesso ordine di grandezza dei pixel in ingresso, il totale dei parametri della rete cresce con il quadrato del numero di pixel e diventa immediatamente intrattabile, sia in termini di memoria richiesta per rappresentare la rete, sia di numero di esempi di input necessari per addestrarla.

D'altra parte, però, una rete per l'analisi di immagini in cui ogni neurone di un layer sia connesso a tutti i neuroni del layer precedente e successivo è probabilmente più complessa del necessario, perché non tiene in considerazione la struttura spaziale delle immagini, trattando ad esempio pixel vicini e lontani allo stesso modo. L'idea delle reti neurali convoluzionali (*Convolutional Neural Network*, CNN) è quella di privilegiare le relazioni tra pixel vicini, peraltro in maniera indipendente dalla loro posizione nell'immagine, con l'effetto collaterale di ridurre drasticamente il numero di parametri della rete.

Una rete neurale convoluzionale mantiene l'organizzazione in layer successivi, ma tra questi figurano un certo numero di layer convoluzionali, che applicano una o più convoluzioni all'input e passano il risultato al layer seguente. La definizione generale dell'operazione di convoluzione è

$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau \,.$$

In caso di funzioni definite sull'insieme degli interi, questa diventa

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m] \,.$$

Se f è non nulla solo in $\{-M, -M+1, \dots, M-1, M\}$ si può usare la somma finita

$$(f * g)[n] = \sum_{m=-M}^{M} f[m]g[n-m] .$$

È possibile estendere la definizione a più dimensioni. Ad esempio, in due dimensioni, e cambiando nomi alle funzioni e alle variabili,

$$(K * I)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} K[m,n]I[i-m, j-n] .$$

Questa è l'operazione che un layer convoluzionale applica a un'immagine o al layer precedente I. K è detto *kernel* o filtro e il risultato *feature map*. Nella pratica, l'operazione consiste nel far scorrere il filtro sulla matrice in input e per ogni posizione produrre in uscita la somma dei prodotti membro a membro tra il filtro e la sottomatrice sottostante. Lo stesso concetto può essere esteso a tre o più dimensioni.

Ma perché proprio questa operazione? Le reti neurali convoluzionali prendono spunto da come la corteccia visiva del nostro cervello percepisce le immagini secondo una struttura gerarchica e un approccio divide-et-impera. I neuroni che ricevono gli stimoli visivi non vedono l'intera immagine, ma reagiscono a stimoli provenienti solo dalle parti dell'immagine nel proprio *receptive field*. Questi neuroni riconoscono quindi pattern locali e di basso livello. I neuroni nei livelli successivi sono a loro volta connessi solo ai neuroni del livello precedente che si trovano nel proprio receptive field e combinano feature di basso livello per riconoscere pattern sempre più complessi e di più alto livello.

L'applicazione di un filtro convoluzionale in una posizione dell'input simula il receptive field locale del neurone al livello successivo che riceve il risultato. Inoltre, poiché i pesi con cui un neurone è connesso ai neuroni del layer precedente nel proprio receptive field sono condivisi tra tutti i neuroni dello stesso layer, tutti i neuroni di uno stesso layer rilevano la stessa feature, ma in posizioni differenti. Le connessioni tra neuroni di livelli consecutivi sono quindi molto più sparse e coinvolgono molti meno parametri rispetto a livelli fully connected.

Non è necessario che l'applicazione di un kernel avvenga in ogni posizione dell'input di un layer convoluzionale: è possibile far scorrere il kernel saltando una o più posizioni, nel qual caso la dimensione del livello successivo sarà una frazione di quella corrente. Il numero di posizioni di cui viene spostato un filtro lungo le diverse dimensioni prima di essere riapplicato è detto *stride*. Se si vuole poi rendere

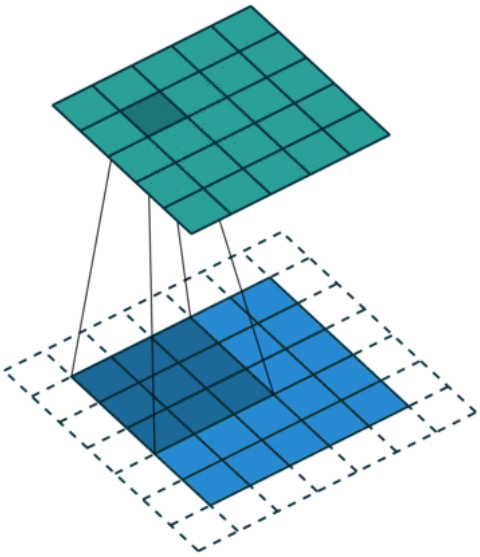

Figura 1.7 – Rappresentazione grafica dell'operazione di convoluzione, con stride unitario e padding.

possibile l'applicazione del kernel anche nelle posizioni più vicine ai bordi dell'input, è necessario aggiungere esternamente un apposito *padding*, solitamente costituito da valori nulli: ciò consente, ed è il caso più comune, di ottenere in uscita feature map di dimensioni che siano le stesse, o un sottomultiplo esatto se lo stride non è unitario, delle feature map in ingresso.

Per come descritto finora, un layer convoluzionale bidimensionale riceve in ingresso una matrice $h \times w$ e produce un'altra matrice di dimensioni $h' \times w'$ che dipendono dallo stride e dal padding. Nella realtà questo tipo di layer ha come input una molteplicità k di feature map e produce a sua volta k' feature map. Ciò richiede di applicare all'input k' differenti kernel, ciascuno dei quali rileverà un pattern differente, e ciascuno dei quali, concettualmente bidimensionale, ha in realtà tre dimensioni (che salgono poi a quattro se si considera la dimensione del batch). La terza dimensione è pari al numero di feature map in ingresso k, il che significa anche che il filtro non viene fatto scorrere lungo la dimensione delle feature map. Ad esempio, un layer che applichi kernel 3×3 su un input di 8 feature map producendo in output 16 feature map fa in realtà uso di 16 kernel di dimensione $3 \times 3 \times 8$. Analogamente, un kernel che venga applicato a una molteplicità di feature map tridimensionali ha in realtà quattro dimensioni (più quella del batch).

Se i livelli convoluzionali sono i costituenti fondamentali di una CNN, in essa figurano spesso anche layer fully connected e layer di *pooling*. Un layer di pooling effettua un subsample del layer precedente ed è solitamente utilizzato immediatamente dopo un layer convoluzionale (o sull'input) per semplificarne

l'informazione in una feature map di dimensione inferiore. Ciò avviene applicando un'operazione di sintesi, quale il massimo o la media, secondo una finestra scorrevole con stride uguale alle sue dimensioni, in modo che non vi siano sovrapposizioni tra le aree coinvolte. A differenza della convoluzione, il pooling è applicato separatamente a ogni feature map, perciò ne lascia invariato il numero. Inoltre non coinvolge alcun parametro addestrabile: la sua funzione è solo quella di ridurre la dimensione dell'input e di conseguenza quella dei livelli successivi.

2 GC-NET CON PUNTI A DISPARITÀ NOTA

2.1 CNN per la visione stereo: DispNet, GC-Net

Date una vista sinistra e una vista destra di una scena, preferibilmente rettificate, il problema della visione stereo, come abbiamo visto, consiste nel determinare, per ogni pixel di un'immagine, il corrispondente nell'altra immagine e di conseguenza la sua disparità, legata da un rapporto di proporzionalità inversa alla profondità del punto nella scena. Algoritmi stereo tradizionali come SGM consentono di ottenere buoni risultati, ma mostrano particolari difficoltà in aree prive di texture o che presentano pattern ripetitivi, in presenza di riflessi, in condizioni di luminosità scarsa o eccessiva. Ciò che manca a un algoritmo tradizionale è una comprensione della struttura complessiva della scena e il riconoscimento dei singoli oggetti. Viceversa, le reti neurali convoluzionali si dimostrano particolarmente efficaci nel cogliere la semantica di un problema quando addestrate su dataset adeguatamente ampi e vari, e ciò ne suggerisce l'applicazione alla visione stereo, in cui la comprensione di informazioni di contesto può certamente aiutare nella ricerca di punti corrispondenti.

In principio, una rete neurale convoluzionale generica di dimensioni sufficienti che riceva in ingresso le due viste di una scena dovrebbe essere già in grado, quando opportunamente addestrata, di arrivare a processare gli input in modo da estrarre informazioni di disparità.

È questo il caso, ad esempio, di *DispNet* [1] nella sua versione più semplice. *DispNetSimple* riceve in input la concatenazione delle due viste lungo la dimensione dei canali dell'immagine e presenta una struttura di tipo encoder-decoder. In questa architettura una sequenza di layer convoluzionali che riducono progressivamente le dimensioni delle feature map (ma non il loro numero) tramite stride maggiori di 1 (fase di contrazione, encoder) è seguita da una sequenza di layer che, all'opposto, aumentano la dimensione delle feature map fino a tornare alla dimensione originale (fase di espansione, decoder). Nel caso di DispNet, l'encoder riduce le dimensioni delle feature map di 64 volte. I layer del decoder applicano un'operazione di convoluzione trasposta, detta comunemente e impropriamente deconvoluzione o *upconvolution*, che ha appunto l'effetto di estendere le feature map. Questa struttura si dimostra efficace perché riduce notevolmente lo sforzo per addestrare la rete,

permettendo allo stesso tempo l'aggregazione di informazioni su ampie aree delle immagini in input. Alcuni layer della fase di contrazione sono connessi ai corrispondenti layer della fase di espansione tramite connessioni che consistono nel concatenare l'output di un layer di deconvoluzione con le feature map del layer di contrazione omologo: la concatenazione delle feature map è poi passata alla deconvoluzione successiva, come di consueto. In questo modo non c'è alcun collo di bottiglia nella rete, in quanto le informazioni possono passare anche dall'encoder al decoder tramite queste connessioni a lungo raggio.

È interessante notare come l'architettura di DispNetSimple ricalchi quella di *FlowNetSimple* [2], una rete finalizzata alla risoluzione del problema dell'*optical flow* (flusso ottico), che, dati due frame successivi di una scena con oggetti in movimento, ha lo scopo di assegnare a ogni pixel di un frame un *motion vector* (vettore di movimento) che punta alla posizione del pixel corrispondente nel frame successivo. In effetti possiamo pensare il problema della visione stereo come un caso più semplice del flusso ottico, in cui la scena è immobile mentre è la camera a muoversi orizzontalmente: se le due viste sono rettificate, nel passaggio da quella sinistra a quella destra ogni vettore di movimento è orizzontale e punta verso sinistra.

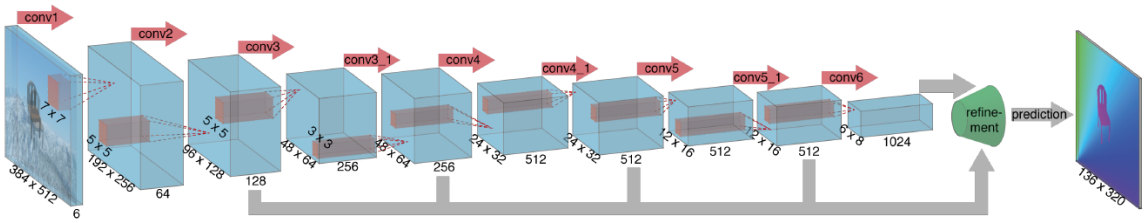

Figura 2.1 – Architettura di FlowNetSimple.
Il componente indicato come "refinement" realizza la fase di espansione.

Non possiamo tuttavia essere certi che gli algoritmi e i dataset di training che abbiamo a disposizione portino mai una rete di questo tipo a raggiungere i migliori risultati possibili. Può allora essere più opportuno progettare una rete con un'architettura meno generica che tenga conto del problema da risolvere e in particolare delle nostre conoscenze della geometria stereo.

Le stesse FlowNet e DispNet che dalla prima prende spunto presentano anche una versione più articolata, rispettivamente *FlowNetCorr* e *DispNetCorr*, che tiene conto della struttura del problema della corrispondenza. L'idea è quella di processare separatamente le due immagini in una prima parte della rete con due sottoreti identiche, in modo da estrarre da esse rappresentazioni utili per le fasi successive, e solo in seguito combinarle con un criterio che aiuti la rete ad effettuare il matching. In FlowNetCorr l'unione delle rappresentazioni estratte separatamente dalle due

immagini avviene introducendo un *correlation layer* che riceve una misura moltiplicativa della similarità degli intorni di coppie di punti provenienti rispettivamente dalle due immagini. In particolare, per ogni pixel x_1 della prima immagine, e per ognuno dei D^2 pixel x_2 nel quadrato di lato $D = 2d + 1$ centrato su x_1, viene calcolata la correlazione $c(x_1, x_2)$ dei loro intorni di lato $K = 2k + 1$. $c(x_1, x_2)$ è definita come la somma dei prodotti scalari tra feature map in posizioni corrispondenti nei due intorni,

$$c(x_1, x_2) = \sum_{o \in [-k,k] \times [-k,k]} \langle f_1(x_1 + o), f_2(x_2 + o) \rangle ,$$

ed è pertanto uno scalare. Il correlation layer ha quindi concettualmente quattro dimensioni, ma per ogni coppia di punti le D^2 correlazioni sono organizzate linearmente in D^2 canali, con il risultato finale di sole tre dimensioni. DispNetCorr segue un approccio simile, processando separatamente le due viste e costruendo il correlation layer, con la differenza che la correlazione può seguire la sola direzione orizzontale e quindi considerare un numero maggiore di possibili disparità.

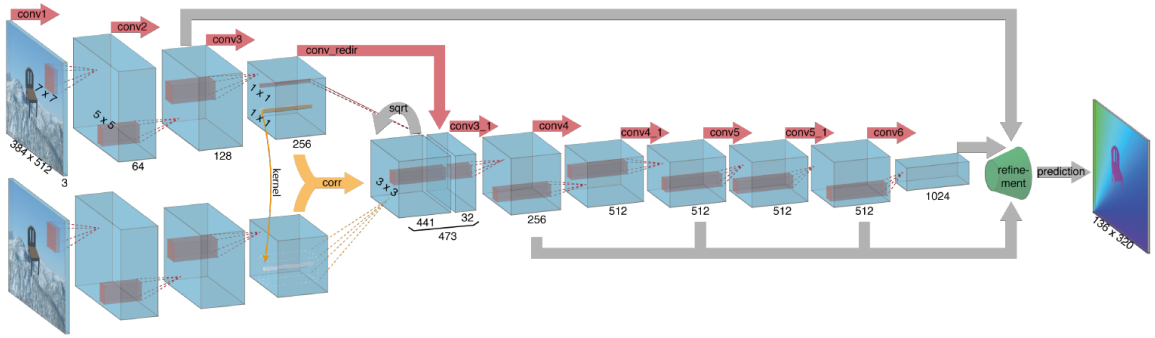

Figura 2.2 – Architettura di FlowNetCorr.

Anche l'architettura di GC-Net (*Geometry and Context Network*) [4] è progettata tenendo conto di intuizioni ben note e di risultati pregressi sulla geometria stereo. In alcune parti, in effetti, non si distanzia molto da DispNetCorr, elaborando le due viste separatamente per produrre due diverse rappresentazioni, composte da *unary features*, combinandole e comparandole per molteplici valori di disparità, e utilizzando un encoder-decoder per effettuare la predizione delle disparità corrette. Ma presenta anche importanti differenze: al posto di un correlation layer, GC-Net produce un volume dei costi, in cui le rappresentazioni ottenute dalla prima fase, opportunamente traslate, sono concatenate per ogni possibile valore di disparità. Inoltre, l'idea dell'encoder-decoder è quella di non produrre direttamente la mappa di disparità, ma un costo del matching per ogni coppia di punti e per ogni disparità. Allora a valle dell'encoder-decoder sarà presente una componente che effettua

un'operazione di argmin lungo la direzione delle disparità: si tratta in particolare di una *soft argmin*, che ha il vantaggio di essere differenziabile e di poter produrre disparità frazionarie. Più dettagli tra poco.

2.2 GC-Net in dettaglio

GC-Net sarà oggetto della maggioranza di questa tesi, perciò ne vediamo dettagliatamente la struttura. Come accennato, la rete può essere scomposta in quattro componenti: l'estrazione delle unary features da ciascuna delle due viste, la costruzione del volume dei costi, un encoder-decoder che indicheremo come fase di *regularization*, e la soft argmin.

Nella prima parte è estratta una rappresentazione interna delle due viste separatamente, tramite due sequenze parallele di layer convoluzionali. Le due sottoreti condividono i parametri, in modo da imparare in maniera più efficace feature corrispondenti. Ciascuna sottorete applica prima di tutto un kernel 5×5 con stride 2 per dimezzare la dimensione dell'input. A questo seguono otto *residual blocks*, ciascuno costituito da due layer che applicano F filtri 3×3 con stride 1. Ognuno di questi layer fa uso di funzione di attivazione ReLU e di *batch normalization*. Infine un ultimo layer applica ancora un filtro 3×3 con stride 1 ma senza funzione di attivazione né batch normalization.

Di interesse è l'impiego di residual block, introdotti da ResNet [5], una rete sviluppata da Microsoft Research che al momento della sua presentazione impose nuovi standard di accuratezza nel riconoscimento di immagini. Un residual block è una successione di layer convoluzionali al cui output finale $F(x)$ è sommato l'input iniziale x, producendo in uscita $F(x) + x$. Dunque, chiamando $H(x) = F(x) + x$ la funzione che vogliamo il blocco impari, i layer convoluzionali devono in realtà imparare $F(x) = H(x) - x$, cioè la differenza, o residuo, tra l'input e l'output desiderato. L'intuizione è che i residual block sono più facili da addestrare rispetto

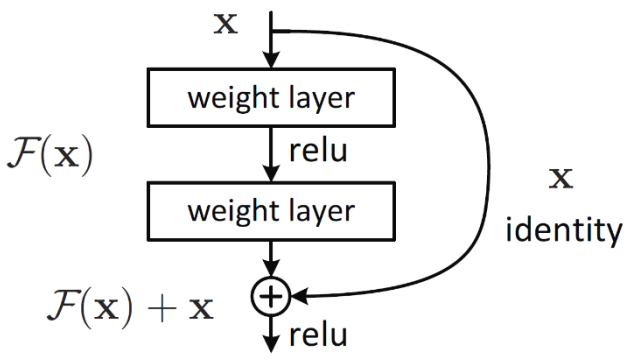

Figura 2.3 – Schema di un residual block costituito da due layer convoluzionali.

a layer convoluzionali classici e pertanto facilitano il training di reti particolarmente profonde.

Riprendiamo GC-Net: nella seconda fase le unary feature sono combinate nel volume dei costi, che sarà utilizzato per il calcolo del costo del matching di pixel per i diversi valori di disparità. Per ogni valore ammesso di disparità d da 0 a un massimo $D_{\max} = D - 1$ fissato a priori, le $F = 32$ unary feature sinistre sono concatenate con le $F = 32$ unary feature destre corrispondenti per quel valore di disparità, cioè traslate a destra di d pixel. Si ottiene un volume di dimensione $H \times W \times D \times 2F$.

Nella costruzione del volume dei costi, un'alternativa alla concatenazione delle unary feature è quella di effettuarne il prodotto scalare. Questo approccio riduce la dimensionalità del volume dei costi a $H \times W \times D \times 1$, che può essere pensata effettivamente tridimensionale, abbassando il tempo di training di circa il 25 % senza sacrificare significativamente l'accuratezza. Faremo riferimento a questa versione di GC-Net come "versione con prodotto scalare", mentre a quella originaria come "versione con concatenazione".

La terza parte della rete, o *regularization*, è la più complessa. Essa applica al volume dei costi un encoder-decoder con diverse connessioni in avanti e produce, per ogni disparità e ogni coppia di punti, il costo del matching. Le dimensioni delle feature map sono ridotte di 16 volte, che insieme al dimezzamento iniziale corrisponde a una riduzione dell'input di 32 volte. Ogni layer applica una convoluzione o deconvoluzione 3D con filtri $3 \times 3 \times 3$ e fa uso di ReLU e batch normalization. Per tornare alla risoluzione dell'input è posto a valle un ulteriore layer di deconvoluzione con stride 2, senza funzione di attivazione né batch normalization, che produce una sola feature map di dimensioni $H \times W \times D$.

Nell'ultima fase, come è tipico degli algoritmi stereo, si utilizza una argmin sul volume dei costi lungo la dimensione della disparità per individuare, per ogni pixel della vista sinistra, la disparità con costo minore. Tuttavia una semplice argmin ha due problemi: non è in grado di produrre disparità con precisione inferiore al pixel, e non è differenziabile, quindi è inadatta alla backpropagation. Per superare queste limitazioni, al suo posto è definita una "soft argmin", realizzata come segue. Prima di tutto passiamo da un volume dei costi a un volume di probabilità negando, per ogni disparità d, ogni costo c_d. In seguito normalizziamo lungo la dimensione delle

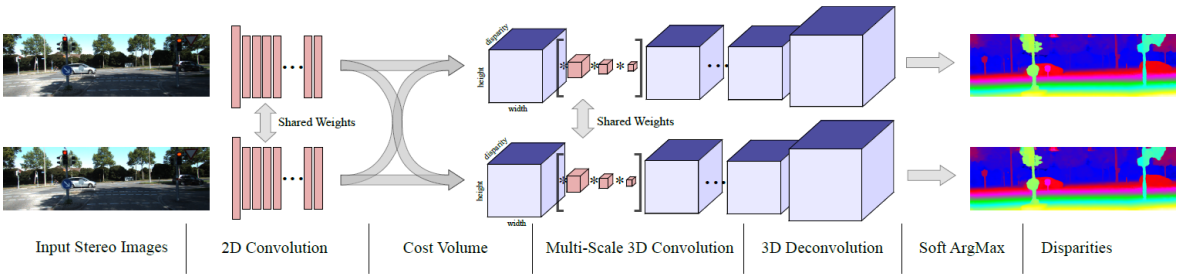

Figura 2.4 – Architettura di GC-Net.

disparità con un'operazione di softmax, $\sigma(\cdot)$. Infine calcoliamo la somma di ogni disparità d pesata dalla sua probabilità normalizzata $\sigma(-c_d)$. In sintesi:

$$\underset{0 \le d < D}{\text{soft arg min}}\, c_d = \sum_{d=0}^{D-1} d \cdot \sigma(-c_d)$$

Questa operazione è ora differenziabile, ma ha un problema: il suo output è influenzato da tutti i costi e può differire da quello di una semplice argmin in caso di distribuzioni multimodali. Ci affidiamo alla capacità della rete di imparare a produrre distribuzioni di probabilità unimodali.

La struttura dettagliata della rete per quanto concerne i layer convoluzionali è sintetizzata nella seguente tabella.

Nome	*Input*	*Kernel*	*Stride*	*Altro*	*Output*
input	Input				$H \times W \times C$
Unary features (due sottoreti)					
conv1	input	5×5	2		$½H \times ½W \times F$
conv2	conv1	3×3	1		$½H \times ½W \times F$
conv3	conv2	3×3	1	+ conv1	$½H \times ½W \times F$
conv4	conv1	3×3	1		$½H \times ½W \times F$
conv5	conv4	3×3	1	+ conv3	$½H \times ½W \times F$
⋮	⋮	⋮	⋮	⋮	⋮
conv18	conv17	3×3	1	no ReLU/BN	$½H \times ½W \times F$
Volume dei costi					
volume	conv18	Volume dei costi			$½H \times ½W \times ½D \times 2F$
Regularization					
conv19	volume	$3 \times 3 \times 3$	1		$½H \times ½W \times ½D \times F$
conv20	conv19	$3 \times 3 \times 3$	1	skip	$½H \times ½W \times ½D \times F$
conv21	volume	$3 \times 3 \times 3$	2		$¼H \times ¼W \times ¼D \times 2F$
conv22	conv21	$3 \times 3 \times 3$	1		$¼H \times ¼W \times ¼D \times 2F$
conv23	conv22	$3 \times 3 \times 3$	1	skip	$¼H \times ¼W \times ¼D \times 2F$
conv24	conv21	$3 \times 3 \times 3$	2		$⅛H \times ⅛W \times ⅛D \times 2F$
conv25	conv24	$3 \times 3 \times 3$	1		$⅛H \times ⅛W \times ⅛D \times 2F$

conv26	conv25	$3 \times 3 \times 3$	1	skip	$\frac{1}{8}H \times \frac{1}{8}W \times \frac{1}{8}D \times 2F$
conv27	conv24	$3 \times 3 \times 3$	2		$(H \times W \times D)/16 \times 2F$
conv28	conv27	$3 \times 3 \times 3$	1		$(H \times W \times D)/16 \times 2F$
conv29	conv28	$3 \times 3 \times 3$	1	skip	$(H \times W \times D)/16 \times 2F$
conv30	conv27	$3 \times 3 \times 3$	2		$(H \times W \times D)/32 \times 4F$
conv31	conv30	$3 \times 3 \times 3$	1		$(H \times W \times D)/32 \times 4F$
conv32	conv31	$3 \times 3 \times 3$	1		$(H \times W \times D)/32 \times 4F$
conv33	conv32	$3 \times 3 \times 3$	2	+ conv29	$(H \times W \times D)/16 \times 2F$
conv34	conv33	$3 \times 3 \times 3$	2	+ conv26	$\frac{1}{8}H \times \frac{1}{8}W \times \frac{1}{8}D \times 2F$
conv35	conv34	$3 \times 3 \times 3$	2	+ conv23	$\frac{1}{4}H \times \frac{1}{4}W \times \frac{1}{4}D \times 2F$
conv36	conv35	$3 \times 3 \times 3$	2	+ conv20	$\frac{1}{2}H \times \frac{1}{2}W \times \frac{1}{2}D \times F$
conv37	conv36	$3 \times 3 \times 3$	2	no ReLU/BN	$H \times W \times D \times 1$
Soft argmin					
output	conv37	Soft argmin			$H \times W$

Tabella 2.1 – Architettura di GC-Net.

Se $F = 32$, nella versione con concatenazione GC-Net ha 2 847 201 parametri addestrabili, nella versione con prodotto scalare 2 683 905.

Accenniamo infine ora al fatto che è comune, per queste reti stereo, utilizzare per il training una funzione di costo che è la semplice media, calcolata su tutti i pixel, dell'errore assoluto tra la disparità attesa d_{ij} e quella predetta $\hat{d}_{ij}$:

$$Loss(\hat{d}, d) = \frac{1}{N} \sum_{\substack{1 \le i \le H \\ 1 \le j \le W}}^{1} \left| d_{ij} - \hat{d}_{ij} \right| .$$

2.3 Dataset per training e testing

Addestrare una rete neurale convoluzionale con milioni di parametri a risolvere un problema della complessità della visione stereo, nonché valutarne le prestazioni, richiede un dataset adeguatamente ampio, abbastanza realistico e che presenti sufficiente variabilità.

Proprio a questo scopo gli autori di DispNet hanno generato al computer un dataset sintetico, corredato di informazioni di disparità e di flusso di scena, che include decine di migliaia di frame che ritraggono un ampio numero di oggetti che seguono traiettorie complesse. Il dataset, che prende il nome di *SceneFlow* [DispNet], è in realtà costituito da tre sottoparti: FlyingThings3D, Driving, Monkaa.

FlyingThings3D è la parte più consistente, e comprende scene con vari oggetti quotidiani che fluttuano lungo traiettorie 3D su uno sfondo ricco di texture. Da ogni scena sono estratti dieci frame consecutivi. Il dataset è evidentemente non realistico,

e fa invece leva sulla propria dimensione. In effetti, FlyingThings3D è a sua volta diviso in una parte di training (in seguito FlyingThings3D-TRAIN), che include ben 22390 coppie di frame (ciascuna nella vista destra e sinistra) e che useremo per il training, e una di testing (in seguito FlyingThings3D-TEST), che consta di 4370 coppie di frame. Ogni frame ha risoluzione 960x540.

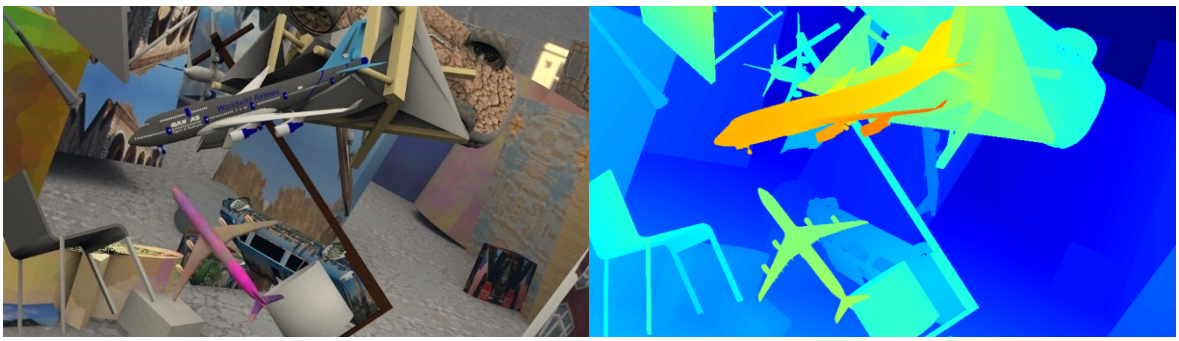

Figura 2.5 - Un frame sinistro e la relativa mappa di disparità estratti dal dataset FlyingThings3D-TRAIN.

Per quanto riguarda le altre due parti di SceneFlow, *Monkaa* è basato sugli asset del cortometraggio animato Monkaa, mentre *Driving* è più realistico, includendo scene di guida stradale riprese dal punto di vista di un'automobile, e nasce sulla falsariga dal dataset KITTI.

Il dataset KITTI è un benchmark costituito da frame di scene stradali catturate dal vivo guidando in una cittadina di medie dimensioni, in aree rurali e in autostrada. L'acquisizione delle scene è avvenuta con due coppie di camere ad alta risoluzione (una coppia a colori e una in bianco e nero) e un sensore laser LiDAR Velodyne montati su un'auto, calibrati e sincronizzati. Se il dataset contiene dati reali, il metodo di acquisizione fa sì che la ground truth sia sparsa (al 50 % circa), limitata alle parti statiche e sufficientemente vicine della scena, e arrivi solo fino a una certa altezza. Una prima versione del dataset risale al 2012 [6], ed è stata poi estesa nel 2015 [7] con l'inserimento nella scena di modelli 3D delle auto in movimento per far sì che la ground truth includesse anche questi oggetti. Le coppie stereo di frame per cui è disponibile la ground truth sono 194 nella versione 2012 e 200 nella versione 2015, e la risoluzione è di circa 1240x376.

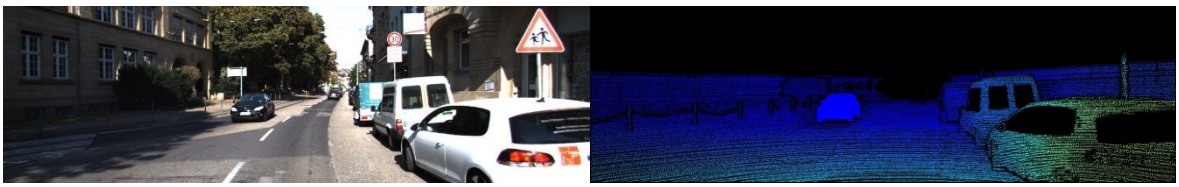

Figura 2.6 – Vista sinistra e disparità estratte da KITTI 2015.

Citiamo inoltre il dataset Middlebury [8], un benchmark che contiene scene reali ma limitate ad ambienti interni e controllati e ha dimensioni estremamente ridotte in termini odierni. Un esempio tratto da questo dataset è stato già mostrato nella Figura 1.5. Si noti che, nel calcolo dei risultati complessivi sull'intero dataset, il sistema di valutazione online di Middlebury attribuisce un peso inferiore ad alcune coppie di immagini, per compensarne la maggiore difficoltà: quando adotteremo questo criterio ci riferiremo ai risultati come "Middlebury pesato".

Infine, anche se per il training faremo uso di un ampio dataset, FlyingThings3D-TRAIN, una strategia per migliorare ulteriormente la capacità di generalizzazione delle reti neurali è la *data augmentation*, che consiste nell'estendere il dataset apportando varie alterazioni ai dati di training. Per gli scopi della visione stereo è ad esempio possibile modificare luminosità, contrasto, gamma e colore delle immagini a disposizione.

2.4 Modifica di GC-Net

Un primo obiettivo di questa tesi è ora adattare una rete neurale convoluzionale per la visione stereo al fine di accettare dati sparsi di disparità provenienti da qualche fonte esterna, una sorta di mappa dei suggerimenti che speriamo possa aiutare la rete a migliorare la propria accuratezza.

Dati di disparità disponibili a priori possono ad esempio essere prodotti da un sensore 3D come un LiDAR, da cui la rete riceverebbe dati sparsi ma precisi. In questo caso l'idea è quella di integrare un sensore LiDAR relativamente economico che produca dati di profondità piuttosto sparsi con una coppia di camere nel tentativo di avvicinare la precisione di un tale setup a quella di un LiDAR in grado di produrre mappe di profondità più dense e dettagliate ma complessivamente molto più costoso.

Viceversa è pensabile utilizzare dati di disparità provenienti da un differente algoritmo per la visione stereo, ad esempio tradizionale. Ovviamente i dati ottenuti non sono sempre corretti, ma l'idea è quella di integrare un tale algoritmo nella nostra rete con la speranza che, dove questa mostra più difficoltà, il primo abbia un'accuratezza superiore e le possa venire in aiuto.

Per quanto riguarda la scelta della rete, la decisione è ricaduta su GC-Net, e in particolare sulla versione con prodotto scalare, perché il numero relativamente contenuto di parametri rispetto ad altre reti fa sì che richieda un tempo più ridotto per l'addestramento, circa due giorni e mezzo su una GPU NVIDIA Titan X per 150 000 immagini di training, e sia meno avida di memoria.

Resta da definire come integrare dati sparsi in GC-Net. Prima di tutto aggiungiamo due input alla rete, ciascuno di dimensione $H \times W \times 1$: una mappa H potenzialmente sparsa delle disparità note; una relativa matrice V di validità, costituita da zeri e uno, che specifica quali punti di H devono essere effettivamente utilizzati.

La nostra proposta è quella di andare ad agire sul volume dei costi, alterando il risultato della combinazione delle unary feature. Un'idea può essere quella, per ogni punto di cui è suggerita la disparità, di azzerare ogni elemento corrispondente a una disparità differente, lasciando inalterato l'unico relativo alla disparità fornita. Riteniamo questo approccio un po' brutale, pertanto agiamo invece in questo modo: per ogni punto di cui è suggerita la disparità, cioè per cui $v_{ij} = 1$, moltiplichiamo gli elementi lungo la dimensione delle disparità d per una funzione Gaussiana centrata sul valore di disparità fornito h_{ij}. Così facendo, il valore corrispondente alla disparità $d = h_{ij}$ sarà moltiplicato per il massimo della Gaussiana, mentre gli altri per valori sempre più vicini a zero allontanandosi da h_{ij}. In particolare, moltiplichiamo per

$$k \cdot e^{-\frac{(d-h_{ij})^2}{2c^2}}$$

dove k è il massimo e c determina la larghezza.

Per rendere l'operazione applicabile indifferentemente anche a tutti gli altri punti, scegliamo di moltiplicare per

$$(1 - v_{ij}) + v_{ij} \cdot k \cdot e^{-\frac{(d-h_{ij})^2}{2c^2}}$$

che per $v_{ij} = 0$ vale proprio 1 e quindi non ha alcun effetto.

L'idea generale è che, anche se alteriamo il volume dei costi solo in corrispondenza dei punti, sparsi, per cui la disparità è fornita, ci attendiamo che la rete utilizzi ed estenda queste informazioni per migliorare la propria accuratezza negli intorni di questi punti.

Dobbiamo però fare attenzione al fatto che la rete opera su feature map che hanno dimensioni dimezzate rispetto all'input e che anche il volume dei costi è costruito di conseguenza. La nostra mappa di disparità e la relativa maschera di validità devono perciò essere opportunamente adattate. La procedura più corretta sarebbe la seguente. Per la maschera di validità potremmo utilizzare un semplice max pooling con una finestra di lato 2, con l'effetto che per ogni finestra il valore prodotto nella maschera ridotta sia 1 solo in presenza di almeno un 1. Per la mappa

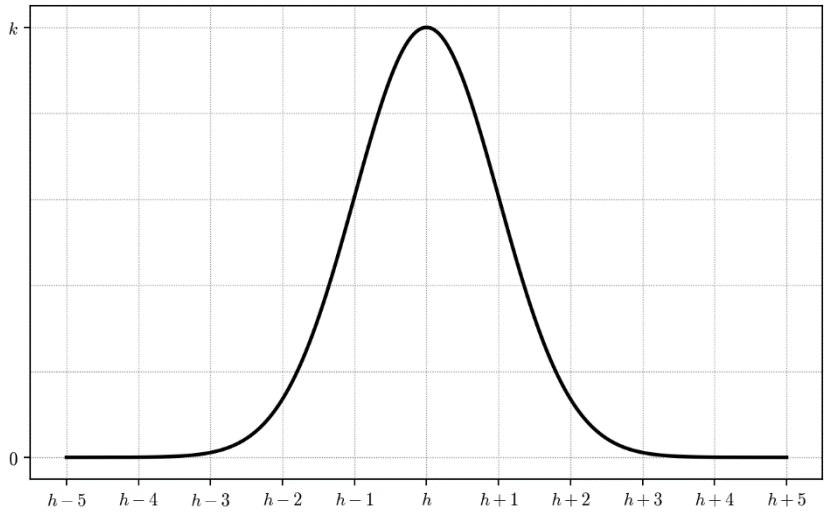

Figura 2.7 – Moltiplicatore del volume dei costi lungo la dimensione della disparità per $v_{ij} = 1$, con $c = 1$.

di disparità, invece, dovremmo estrarre da ogni finestra di lato 2 la media delle disparità, considerando però solo quelle valide (o, se nessuna è valida, produrre un valore qualsiasi, ad esempio 0, che in ogni caso non sarà utilizzato). Invece utilizziamo un'approssimazione, applicando un max pooling con finestra di lato 2 a entrambe: dopotutto, anche se in una finestra dovesse esservi più di una disparità valida, queste differirebbero significativamente solo di rado e anche in quel caso non introdurremmo un errore rilevante.

Di importanza decisamente maggiore è invece il fatto che, poiché le dimensioni delle feature map sono dimezzate, altrettanto lo sono le distanze tra punti corrispondenti: occorre pertanto dimezzare anche i valori di disparità che forniamo dall'esterno prima di utilizzarli per modulare il volume dei costi, al fine di posizionare correttamente la nostra Gaussiana lungo la dimensione delle disparità.

Osserviamo poi che la nostra formula ammetterebbe qualsiasi $v_{ij} \in [0,1]$, ad esempio valori di confidenza. In realtà non ne faremo questo tipo di uso, dato che non è certamente l'unico modo possibile né il migliore.

Notiamo infine che l'effetto del nostro prodotto è quello di amplificare il costo del matching in corrispondenza della disparità suggerita, e di ridurlo altrove: poiché parliamo di un costo, ci aspetteremmo di dover agire in maniera opposta. Questo non è un problema, perché la rete può viceversa imparare senza alcuna difficoltà a interpretare il volume dei costi come una misura di similarità. Di più: poiché i costi sono calcolati come prodotto scalare, è del tutto probabile che ciò avvenga già in maniera naturale, essendo il prodotto scalare (normalizzato) di due vettori tanto maggiore quanto più le loro direzioni sono simili.

2.5 Training e risultati con disparità estratte dalla ground truth

Per il primo esperimento forniamo alla rete disparità sparse ma esatte, ottenute campionando la ground truth. L'idea è quella di verificare prima di tutto che il nostro approccio abbia l'effetto desiderato, nonché di simulare disparità ottenute con un sensore 3D che fornisca dati abbastanza sparsi. In particolare, campioniamo la ground truth in maniera causale e uniforme con probabilità del 5 %, cioè un punto ogni venti.

Implementiamo e modifichiamo GC-Net utilizzando l'API Python di TensorFlow [9], una libreria open-source per il machine learning e il calcolo numerico ad alte prestazioni, particolarmente adatta all'ambito delle reti neurali e utilizzata per numerose attività di elaborazione di immagini e riconoscimento del linguaggio. TensorFlow supporta molteplici architetture hardware differenti, tra cui l'architettura CUDA per il calcolo parallelo su GPU NVIDIA: il nostro ambiente di training e testing di reti neurali convoluzionali sarà infatti sempre una GPU NVIDIA Titan X. Particolarmente utile per controllare i progressi della rete durante il training sarà anche TensorBoard, una suite di strumenti studiati per il debug e l'ottimizzazione di programmi TensorFlow che permette, tra le varie funzionalità, di visualizzare, all'avanzare del training, grafici delle metriche di interesse ed esempi di output.

Configuriamo la rete con $F = 32$, come da implementazione originaria, e $D = 192$ (per cui $D_{\max} = D - 1 = 191$). Per quanto riguarda la Gaussiana con cui alteriamo il volume dei costi, scegliamo $c = 1$ e $k = 10$. La scelta di questi valori è piuttosto arbitraria, ma un $k > 1$ è motivato dall'idea di aiutare ulteriormente la rete a differenziare tra i punti per cui è fornita una disparità dall'esterno e gli altri.

Per il training utilizziamo $H = 256$ e $W = 512$. Addestriamo la rete per 150 000 iterazioni con learning rate costante 0.001 sul dataset FlyingThings3D-TRAIN, ritagliando casualmente le immagini di dimensione originaria 960×540 per portarle a 512×256, e impieghiamo anche la tecnica della data augmentation. In particolare, con probabilità del 50 % applichiamo una correzione di gamma con un valore tra 0.8 e 1.2, modifichiamo la luminosità di un fattore tra 0.5 e 2.0 e alteriamo il colore di un fattore tra 0.8 e 1.2. La funzione di loss scelta è quella mostrata nel paragrafo 2.2, cioè l'errore assoluto medio. Controlliamo l'avanzamento del training validando periodicamente la rete su un sottoinsieme di FlyingThings3D-TEST di 437 coppie di immagini, ottenuto considerando solo il primo frame di ogni scena, e ritagliando centralmente ancora alla risoluzione 512×256.

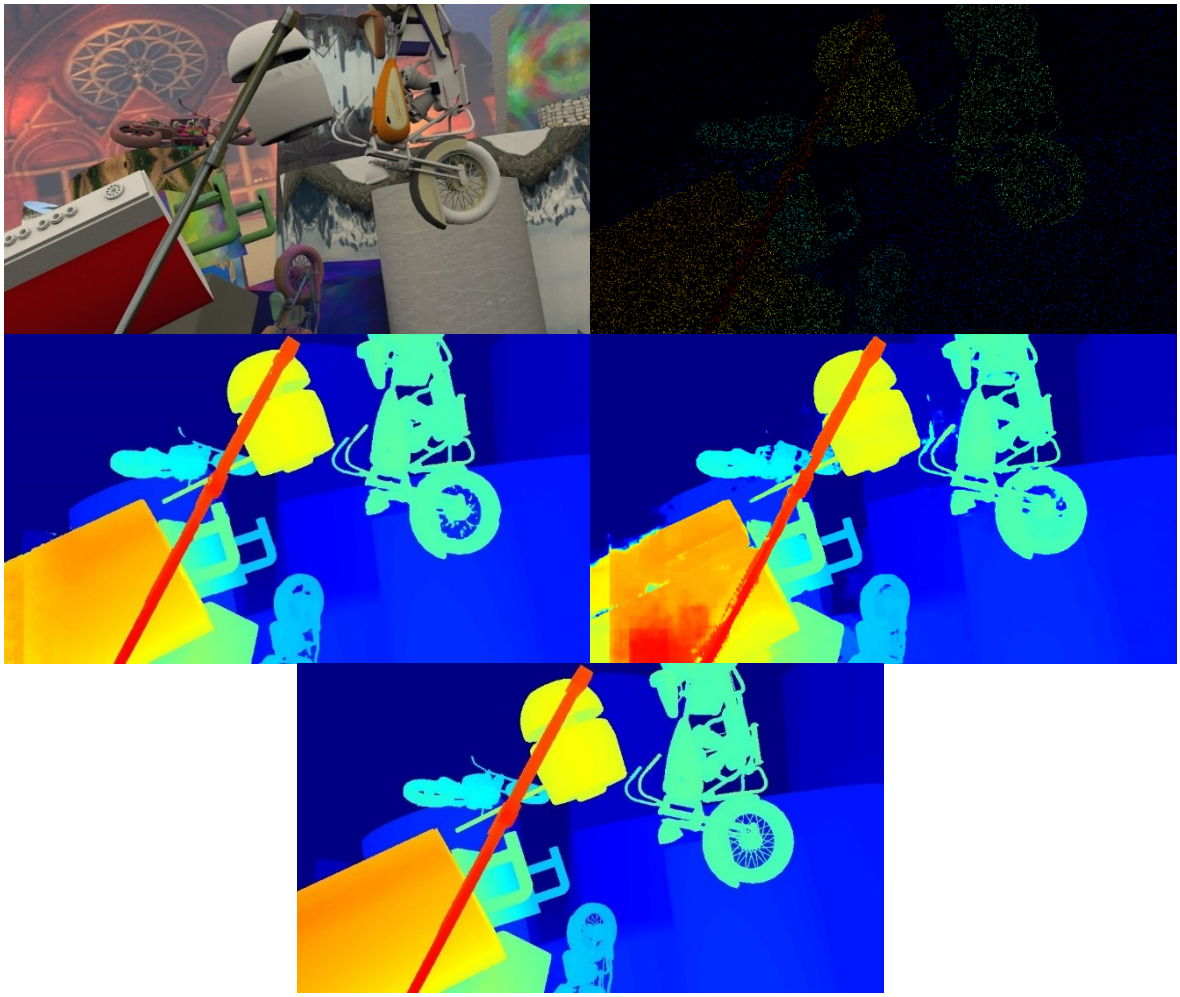

Figura 2.8 – Confronto su un input di FlyingThings3D-TEST tra l'output della rete con ground truth campionata al 5 %, l'output di GC-Net senza dati di disparità e la ground truth. In alto a sinistra: vista sinistra; in alto a destra: campionamento della ground truth; al centro a sinistra: output della rete con suggerimenti; al centro a destra: output di GC-Net; in basso: ground truth.

Infine testiamo su FlyingThing3D-TEST intero, su KITTI 2015, su KITTI 2012 e su Middlebury, mantenendo la risoluzione originaria delle immagini e campionando ancora la ground truth al 5 %. Si noti che, poiché la ground truth di KITTI è già sparsa, il risultato del campionamento avrà in quel caso densità inferiore al 5 %.

Seguono i risultati. Qui EPE indica l'*End-Point Error*, cioè l'errore assoluto medio (lo stesso utilizzato come funzione di loss), mentre D1 indica la percentuale di punti per cui l'errore di disparità è superiore a una soglia fissata, in questo caso 3. Tra parentesi sono sempre riportati, ove disponibili, i risultati ottenuti da GC-Net originaria, anch'essa nella versione con prodotto scalare. Escludiamo inoltre dal

computo finale i casi di input per cui almeno il 25 % delle disparità nella ground truth supera 300: si tratta di ventuno coppie di frame in FlyingThings3D-TEST.

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	0.998 (2.14)	2.408 (6.51)
KITTI 2015	1.062 (3.14)	3.697 (23.82)
KITTI 2012	0.551 (2.91)	2.075 (22.75)
Middlebury (pesato)	0.473	2.401

Tabella 2.2 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dalla ground truth campionando al 5 % e testata con suggerimenti estratti ancora dalla ground truth campionando al 5 %.

Registriamo quindi un notevole miglioramento, atteso, su FlyingThings3D: la Figura 2.8 mette a confronto un output con il rispettivo output ottenuto con GC-Net senza dati di disparità. Ma un miglioramento ancora più consistente si rileva sui due dataset KITTI. Osservando gli output, ad esempio la Figura 2.9, notiamo che ciò è in gran parte dovuto a capacità nettamente migliori nel predire la disparità della strada, un'area viceversa non facile da trattare in quanto ampia e povera di texture. D'altra parte, però, la rete fornisce ora un output evidentemente errato nella zona superiore dell'immagine, dove i dati di disparità sono assenti: ciò non influisce sui risultati perché la ground truth da cui i punti sparsi sono estratti è qui assente, ma testimonia che così addestrata la rete ha imparato a fare eccessivo affidamento sulle disparità che le vengono fornite dall'esterno. Concludiamo quindi che dati di disparità esatti seppur piuttosto sparsi possono sì migliorare notevolmente l'accuratezza di una rete per la visione stereo, tuttavia sarà necessario un regime di addestramento diverso se si prevede di fornire dati con una distribuzione differente.

Allo scopo di osservare fino a che punto possiamo spingere l'abilità della rete nel nostro setup quando valutata esclusivamente su KITTI 2015, uno dei benchmark di riferimento quando si considera la visione stereo nell'ambito dell'assistenza alla guida, effettuiamo un fine-tuning della rete utilizzando per il training il dataset più simile che abbiamo a disposizione, ovvero KITTI 2012. Precisamente, a partire dalla rete già addestrata su FlyingThings3D-TRAIN per 150 000 iterazioni, eseguiamo 50 000 ulteriori iterazioni di training su KITTI 2012 con la stessa learning rate

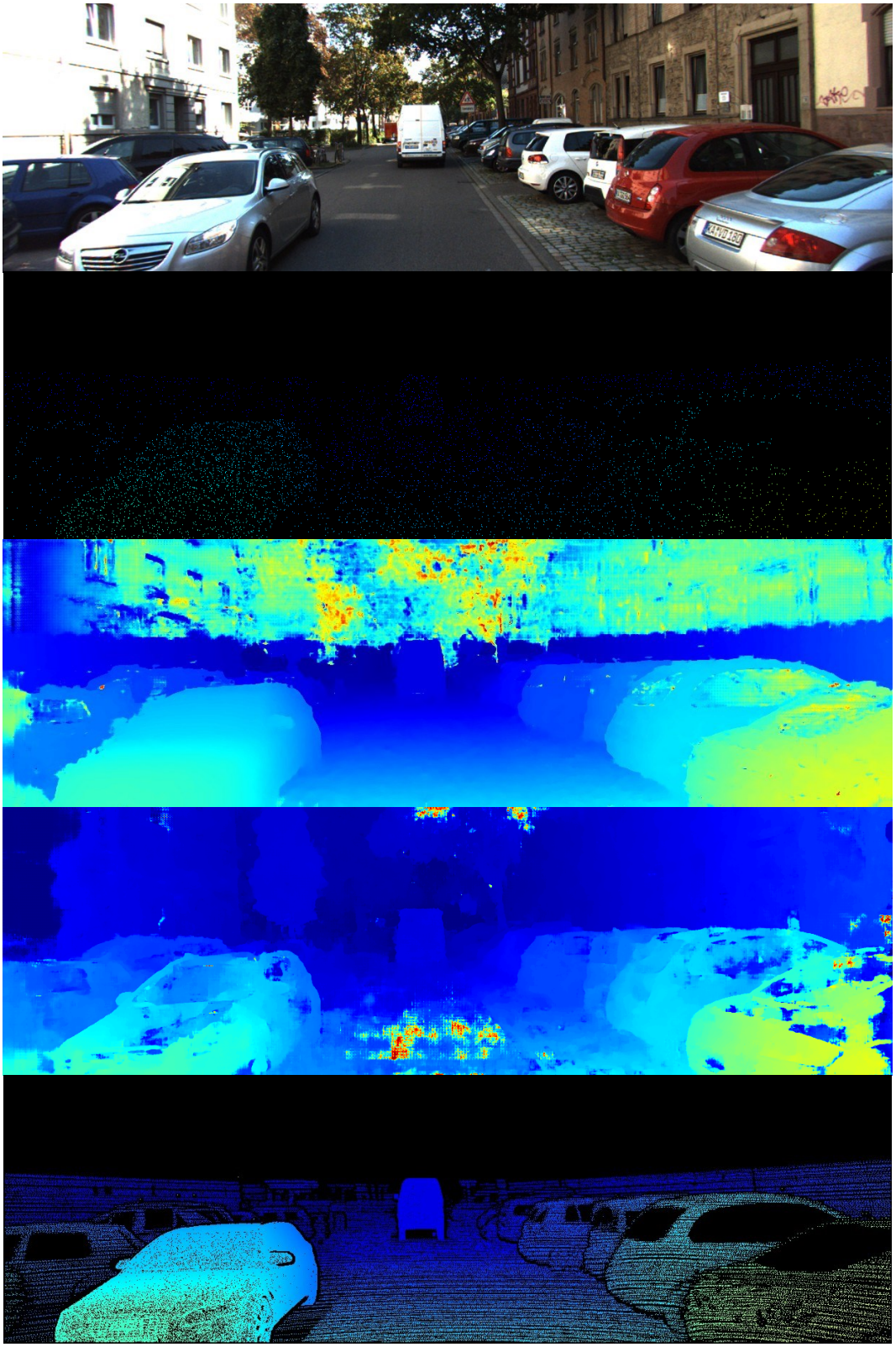

Figura 2.9 – Confronto su un input di KITTI 2015 tra l'output della rete con ground truth campionata al 5 %, l'output di GC-Net senza dati di disparità e la ground truth. Dall'alto verso il basso: vista sinistra; campionamento della ground truth; output della rete con dati di disparità; output di GC-Net; ground truth.

di 0.001. Infine testiamo su KITTI 2015, sempre campionando la ground truth al 5 %. Otteniamo un errore davvero contenuto:

	EPE	*D1 % (>3)*
KITTI 2015	0.365	0.931

Tabella 2.3 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dalla ground truth campionando al 5 % e successivamente fine-tuned su KITTI 2012, e testata con suggerimenti estratti ancora dalla ground truth campionando al 5 %.

Per avere però un'idea più precisa del grado di aiuto che disparità sparse, seppur ancora distribuite uniformemente su tutta l'immagine, forniscono alla rete nella costruzione della mappa di disparità, ripetiamo per intero l'addestramento e il testing, ma campionando la ground truth con probabilità 1/400, cioè dello 0.25 %. I risultati sono i seguenti:

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	1.302 (2.14)	3.694 (6.51)
KITTI 2015	2.479 (3.14)	14.694 (23.82)
KITTI 2012	1.674 (2.91)	9.890 (22.75)
Middlebury (pesato)	1.108	6.110

Tabella 2.4 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dalla ground truth campionando allo 0.25 % e testata con suggerimenti estratti ancora dalla ground truth campionando allo 0.25 %.

Questi valori rispettano le attese, essendo intermedi rispetto al caso precedente con la ground truth campionata al 5 % e GC-Net originaria. Possiamo concludere che anche dati di disparità molto sparsi ma esatti forniscono a GC-Net un aiuto consistente.

Come esperimento vogliamo ora osservare il comportamento di una rete addestrata campionando la ground truth ad una certa densità quando riceve in ingresso dati esatti molto più sparsi o molto meno sparsi. Iniziamo testando la rete addestrata con una ground truth campionata al 5 % fornendole invece la ground truth campionata allo 0.25 %. L'esito è il seguente.

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	21.709 (2.14)	58.717 (6.51)
KITTI 2015	32.310 (3.14)	80.102 (23.82)
KITTI 2012	27.575 (2.91)	78.346 (22.75)
Middlebury (pesato)	18.689	58.257

Tabella 2.5 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dalla ground truth campionando al 5 % e testata con suggerimenti estratti dalla ground truth campionando invece allo 0.25 %.

I risultati sono pessimi. Come già ipotizzato osservando le aree prive di ground truth delle immagini di KITTI, la rete addestrata con dati al 5 % impara a fare eccessivo affidamento sulle disparità che riceve dall'esterno, probabilmente limitandosi ad estendere queste informazioni al loro intorno, e tende ad ignorare le corrispondenze che può ricavare autonomamente dalle due viste. D'altra parte si tratta di disparità esatte, e la loro densità è sufficiente per far sì che questo approccio risulti vincente nel minimizzare l'errore sulle immagini di training.

All'opposto, testiamo la rete addestrata con un campionamento allo 0.25 % suggerendole una ground truth campionata al 5 %:

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	1.610 (2.14)	4.380 (6.51)
KITTI 2015	0.614 (3.14)	1.839 (23.82)
KITTI 2012	0.528 (2.91)	1.852 (22.75)
Middlebury (pesato)	1.030	4.595

Tabella 2.6 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dalla ground truth campionando allo 0.25 % e testata con suggerimenti estratti dalla ground truth campionando invece allo 5 %.

Qui i risultati sono molto interessanti. Ci potremmo attendere prestazioni intermedie tra quelle della rete addestrata e testata con ground truth al 5 % e quelle di questa rete addestrata e testata campionando allo 0.25 %. In effetti, ciò si verifica solo sul dataset Middlebury. Su FlyingThings3D-TEST, invece, questa rete si comporta meglio quando i dati sparsi forniti sono allo 0.25 %, percentuale con cui è stata addestrata, rispetto al caso del 5 %: evidentemente, il miglioramento che dati più densi dovrebbero garantire è più che annullato dal fatto che durante il training la rete non è stata abituata a ricevere suggerimenti con una densità così elevata.

Il caso più notevole riguarda però i dataset KITTI, per cui la rete addestrata con ground truth campionata allo 0.25 % e testata con ground truth al 5 % si comporta meglio della rete addestrata e testata campionando al 5 %; la differenza è particolarmente notevole su KITTI 2015. È necessario ricordare che la ground truth di KITTI è già sparsa, dunque il nostro criterio di campionamento produce una densità effettiva inferiore rispetto alla percentuale riportata. Di fronte a una densità reale intermedia rispetto alle due possibilità che abbiamo utilizzato per il training, risulta chiaro come la rete testata con dati esatti più densi rispetto a quelli incontrati durante l'addestramento si comporti meglio nel confronto con il caso opposto, in cui cioè la fase di test procede con dati più sparsi rispetto al training. Ciò ha ancora più senso se consideriamo nuovamente come la rete addestrata al 5 % faccia

affidamento quasi esclusivamente sui dati esterni, ignorando per buona parte le immagini: l'errore causato dal limitarsi a estendere al loro intorno le disparità già note aumenta rapidamente nel momento in cui esse iniziano a diventare più sparse.

2.6 Training e risultati con disparità estratte da SGM e rSGM

Visti i risultati positivi ottenuti campionando la ground truth, proviamo ora a prelevare dati di disparità dall'output dell'algoritmo tradizionale SGM, e di quello di una sua versione più improntata all'efficienza computazionale, rSGM, in modo da realizzare una prima integrazione tra un algoritmo tradizionale e una rete neurale per la visione stereo. In questa fase ci limitiamo a campionare l'output di SGM con le stesse modalità utilizzate per la ground truth, cioè in maniera casuale e uniforme. Si noti che la mappa di disparità prodotta da SGM presenta, soprattutto in corrispondenza di occlusioni, aree prive di informazione, che risultano da una fase di validazione dell'output che utilizza, ad esempio, il left-right consistency check.

Manteniamo gli stessi parametri della rete e le stesse modalità di training e di testing utilizzate con la ground truth. Per semplificare il procedimento, ed evitare di eseguire più di una volta lo stesso algoritmo sulla stessa coppia di frame, precalcoliamo tutti gli output di SGM e rSGM e li conserviamo su disco. In un primo esperimento campioniamo l'output di SGM al 5 %, sia per il training che per il testing.

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	2.179 (2.14)	6.705 (6.51)
KITTI 2015	3.432 (3.14)	18.643 (23.82)
KITTI 2012	3.031 (2.91)	18.871 (22.75)

Tabella 2.7 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dall'output di SGM campionando al 5 % e testata con suggerimenti estratti ancora dall'output di SGM campionando al 5 %.

L'errore medio è peggiore rispetto a GC-Net su tutti i dataset: apparentemente la rete non riesce nel complesso a bilanciare pienamente gli errori introdotti dalle disparità di SGM con i benefici che queste portano laddove corrette. Nonostante ciò, però, è notevole osservare come la percentuale di punti errati sui dataset KITTI venga effettivamente ridotta. La Figura 2.10 mostra quanto accade: anche tralasciando la parte superiore dell'immagine, che non influisce sui risultati in quando la ground truth è assente, è evidente il miglioramento a livello della strada, area in cui GC-Net ha tipicamente più difficoltà rispetto a SGM.

Procediamo invece con l'output di rSGM campionato ancora al 5 %, sia per il training che per il testing. Riportiamo che le mappe di disparità prodotte da rSGM

tendono ad essere più complete di quelle di SGM, cioè a presentare meno aree prive di informazione, ma sono presumibilmente lievemente meno accurate.

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	2.224 (2.14)	7.149 (6.51)
KITTI 2015	9.535 (3.14)	26.660 (23.82)
KITTI 2012	8.280 (2.91)	26.182 (22.75)
Middlebury (pesato)	4.016	15.535

Tabella 2.8 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dall'output di rSGM campionando al 5 % e testata con suggerimenti estratti ancora dall'output di rSGM campionando al 5 %.

I risultati su FlyingThings3D-TEST sono in linea con le attese, non discostandosi molto dal caso precedente. L'errore medio su KITTI è invece sorprendentemente elevato. Il motivo non è ben chiaro, ma non è da ricercare nella differenza di accuratezza tra SGM e rSGM, certamente non così ampia da giustificare risultati tanto diversi. Un'ipotesi è che la maggiore completezza delle mappe di disparità rispetto a SGM, e di conseguenza il minore bilanciamento tra aree con e senza suggerimenti durante il training, renda più difficile per la rete generalizzare quanto appreso su FlyingThings3D a dataset così diversi, quali i due KITTI, quando supportata da disparità esterne non esatte.

Infine proviamo anche con l'output di rSGM ma campionato allo 0.25 %, sia per il training che per il testing.

	EPE	*D1 % (>3)*
FlyingThings3D-TEST	2.605 (2.14)	7.246 (6.51)
KITTI 2015	3.338 (3.14)	27.476 (23.82)
KITTI 2012	3.013 (2.91)	24.589 (22.75)
Middlebury (pesato)	2.558	14.823

Tabella 2.9 – Risultati ottenuti da GC-Net addestrata con suggerimenti estratti dall'output di rSGM campionando allo 0.25 % e testata con suggerimenti estratti ancora dall'output di rSGM campionando allo 0.25 %.

Si conferma, seppur in maniera meno marcata, la tendenza già registrata campionando rSGM al 5 %: sia l'errore medio sia il numero di punti errati sono peggiori rispetto a quanto GC-Net sia in grado di ottenere autonomamente.

In questo paragrafo abbiamo però fornito a GC-Net dati di disparità estratti da SGM e rSGM in maniera casuale, includendo quindi sicuramente una buona percentuale di punti corretti ma inevitabilmente anche una parte di punti errati. Nel capitolo 4 esamineremo la possibilità di scegliere in maniera più astuta le disparità da suggerire alla rete, cercando di minimizzare quelle errate. Prima, però, ci

domandiamo se il nostro approccio di fornire dati sparsi a una rete stereo possa essere applicato anche a un algoritmo tradizionale.

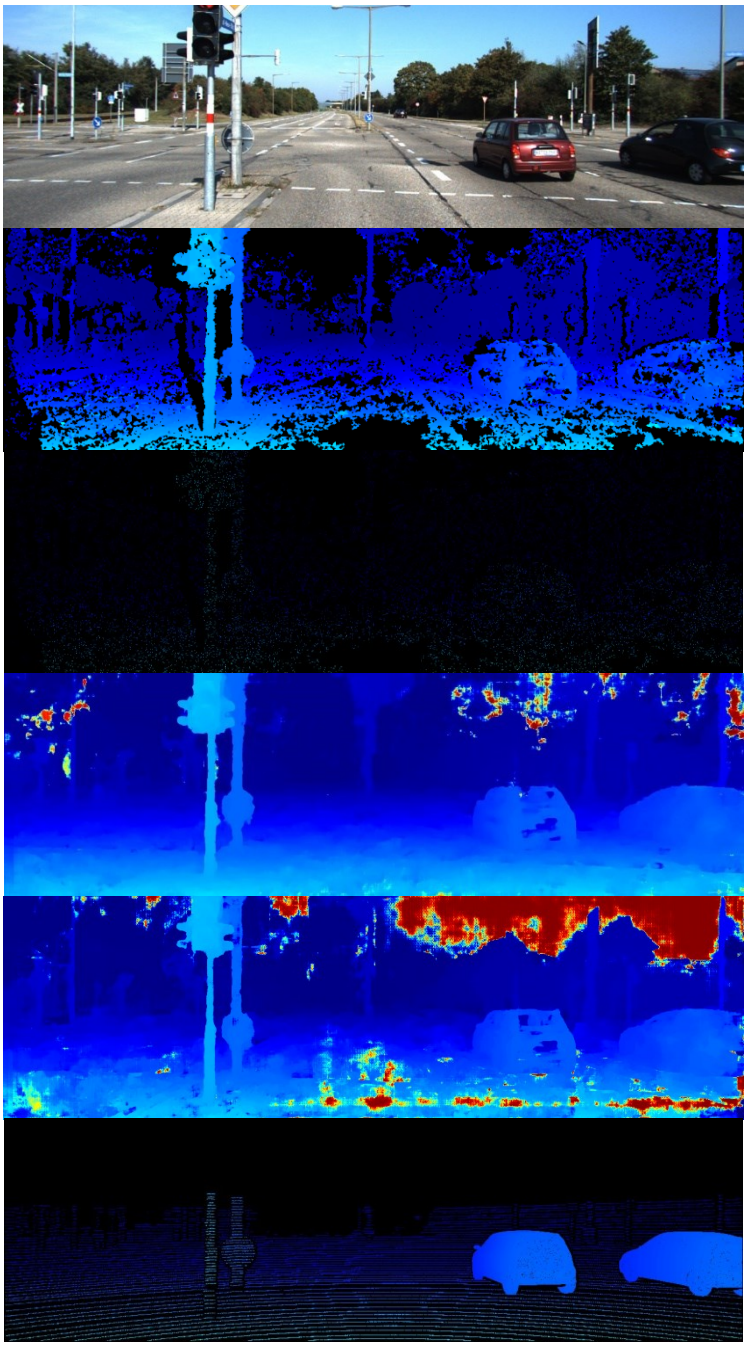

Figura 2.10 – Confronto su un input di KITTI 2015 tra l'output della rete con output di SGM campionato al 5 %, l'output di GC-Net senza dati di disparità e la ground truth. Dall'alto verso il basso: vista sinistra; output di SGM; campionamento di SGM; output della rete con dati di disparità; output di GC-Net; ground truth.

3 rSGM CON PUNTI A DISPARITÀ NOTA

3.1 Algoritmi tradizionali per la visione stereo: SGM e rSGM

Il problema della visione stereo ha iniziato ad essere investigato da ben prima della diffusione e del successo delle reti neurali. Numerosi algoritmi per la risoluzione del problema del matching di punti corrispondenti tra immagini stereo e per la stima della disparità sono stati ideati e studiati: questi algoritmi si basano in generale su caratteristiche visuali di basso livello e sulla minimizzazione di una qualche funzione di costo nell'effettuarne il matching. Nonostante le difficoltà in scenari reali che presentino aree povere di texture, pattern ripetitivi, riflessi, condizioni di luce particolari, discreti risultati sono stati ottenuti.

Una tassonomia degli algoritmi stereo è fornita dallo studio di Scharstein e Szeliski [8], che ne individuano quattro blocchi costitutivi fondamentali ricorrenti: calcolo del costo di matching (*matching cost computation*), aggregazione dei costi (*cost aggregation*), calcolo della disparità (*disparity computation*) e raffinamento della disparità (*disparity refinement*).

Il calcolo del costo di matching produce una misura della dissimilarità tra intorni di pixel potenzialmente corrispondenti, solitamente basata sulle intensità dei pixel nelle finestre messe a confronto.

L'aggregazione dei costi riunisce i costi di matching all'interno di aree più o meno ampie dell'immagine, ad esempio sommando i costi all'interno di una finestra o effettuandone una somma pesata basata sulla distanza dal centro della finestra.

Il calcolo della disparità fornisce una prima approssimazione della mappa di disparità ed è effettuato in maniera differente dagli algoritmi locali e globali. Gli algoritmi locali calcolano la disparità di un punto considerando solo i pixel presenti nell'intorno di quel punto. Il cuore di questi algoritmi sono le prime due fasi, mentre il calcolo della disparità avviene banalmente scegliendo la disparità che minimizza il costo associato, secondo un approccio *winner-takes-all*. Gli algoritmi globali, invece, fanno precise assunzioni riguardo alla regolarità della mappa di disparità e risolvono un problema di ottimizzazione, cercando di minimizzare una funzione che tiene in considerazione tutti i pixel dell'immagine. Solitamente, gli algoritmi

globali non effettuano l'aggregazione dei costi, ma ricercano una mappa di disparità d che minimizzi un'energia globale del tipo

$$E(d) = E_{\text{data}}(d) + \lambda E_{\text{smooth}}(d)\,,$$

dove, senza entrare in eccessivi dettagli, il termine $E_{\text{data}}(d)$ misura quanto la mappa di disparità sia in accordo con i costi di matching, mentre $E_{\text{smooth}}(d)$ codifica le assunzioni di regolarità della mappa di disparità, penalizzando in generale le discontinuità.

La fase di raffinamento della disparità, infine, ha ad esempio lo scopo di produrre disparità con precisione inferiore al pixel, di rimuovere incongruenze con il left-right consistency check, di eliminare i picchi, di stimare la disparità nelle aree che corrispondono a occlusioni.

Gli algoritmi stereo tradizionali con la migliore accuratezza sono tendenzialmente algoritmi globali, ma hanno un costo computazionale molto elevato, talvolta intrattabile, perché nel caso generale minimizzare la funzione di energia che definiscono è un problema NP-hard. Gli algoritmi semi-globali semplificano la funzione di costo a sufficienza affinché non dipenda da tutti i pixel contemporaneamente e possa essere minimizzata utilizzando la programmazione dinamica, risultando in una complessità polinomiale senza sacrificare in maniera significativa l'accuratezza.

L'algoritmo SGM [10], Semi-Global Matching, è appunto un algoritmo semi-globale per la visione stereo che approssima una funzione di disparità globale bidimensionale combinando funzioni definite in una sola dimensione. Il calcolo del costo di matching utilizza l'informazione mutua, robusta di fronte a variazioni di luce, definita a partire dai concetti di entropia ed entropia congiunta. Glissando sui dettagli, il volume dei costi così costruito è utilizzato per calcolare, per ogni pixel e per ogni possibile disparità, un costo aggregato ottenuto combinando costi di matching lungo percorsi monodimensionali. Le discontinuità nella disparità sono penalizzate con una costante P_1 per le variazioni di un pixel e con una costante P_2 per le discontinuità di due o più pixel. A questo punto la mappa di disparità può essere prodotta come negli algoritmi stereo locali scegliendo per ogni pixel quella cui corrisponde il minor costo. L'algoritmo è in grado di produrre le mappe di disparità relative a entrambe le immagini, perciò può applicare il left-right consistency check, cui si aggiungono altri raffinamenti come la rimozione dei picchi, cioè di piccole aree con disparità molto differente rispetto alle zone circostanti.

Velocità di esecuzione idonee per l'utilizzo in tempo reale possono essere raggiunte soltanto implementando SGM su FPGA o GPU. rSGM [11] è un'implementazione in C++ di SGM che si pone l'obiettivo di raggiungere frame rate elevati su CPU multi-core general purpose. A differenza di SGM, i costi di matching sono calcolati come distanza di Hamming tra trasformate Census [13] con finestre 5×5 o 9×7, che si comportano meglio rispetto all'informazione mutua in scene all'aperto. La trasformata Census associa ad ogni pixel nell'intorno di un pixel di riferimento un valore 1 se l'intensità luminosa è maggiore o uguale a quella di riferimento, un valore 0 se è minore. Il calcolo delle trasformate Census è realizzato in maniera efficiente con istruzioni SIMD (*Single Instruction, Multiple Data*). L'immagine è poi suddivisa in bande orizzontali che possano essere processate in parallelo, con una sovrapposizione sufficiente per prevenire la riduzione dell'accuratezza in corrispondenza dei bordi superiore e inferiore delle bande.

L'algoritmo rSGM è parametrizzato da alcune variabili che ne controllano in particolare l'efficienza di esecuzione (eventualmente influenzando la precisione), quali numero di thread, numero di bande, disparità massima e dimensione della finestra Census. Il codice di rSGM [12] mette a disposizione sei diverse modalità preconfigurate, tra le quali faremo in seguito riferimento, in quanto mostrano un'accuratezza maggiore, alla "modalità 2", che ammette una disparità massima di 128 e usa una finestra Census 5×5, e alla "modalità 5", che ammette ancora una disparità massima di 128 e usa una finestra Census 9×7.

3.2 Modifica di rSGM con sostituzione dei costi

Vogliamo ora provare ad agire su rSGM in maniera simile a quanto fatto con GC-Net al fine di far accettare all'algoritmo dati sparsi di disparità che ne migliorino la precisione. Nonostante i due paradigmi siano completamente diversi, rSGM e GC-Net condividono in effetti l'idea del volume dei costi, costruito a partire da feature estratte dalle immagini in input messe a confronto per i possibili valori di disparità. Come abbiamo operato sul volume dei costi di GC-Net, lo stesso faremo con rSGM. Ma se con GC-Net abbiamo scelto di amplificare i valori nei pressi della disparità corretta basandoci sull'assunto che la parte restante della rete sarebbe stata in grado di interpretare arbitrariamente il volume dei costi, in rSGM il significato dei costi di matching è ben definito dal procedimento di aggregazione dei costi che segue. È allora immediato notare che per i punti di cui è fornita la disparità sia necessario ridurre il costo in corrispondenza del valore corretto e aumentare gli altri costi.

Sperimentiamo due approcci differenti. In un primo momento, poiché l'algoritmo di aggregazione dei costi utilizzato non ha troppo spazio per generalizzare

all'intorno di un punto la disparità fornita per quel punto, cerchiamo di dare il massimo aiuto a rSGM agendo nella maniera più invasiva possibile sul volume dei costi: azzeriamo il costo corrispondente alla disparità corretta, fissiamo a un valore costante ed elevato gli altri costi. Per la scelta di questa costante partiamo dal massimo della distanza Hamming tra trasformate Census, cioè $5 \times 5 - 1 = 24$ per finestre 5×5 e $9 \times 7 - 1 = 62$ per finestre 9×7, e testiamo quattro multipli, moltiplicando il massimo per 1, per 2, per 10, e per 100, in seguito denominati semplicemente "moltiplicatori". Quindi per una finestra Census 5×5 con moltiplicatore 10 la costante scelta sarà ad esempio 240.

Riportiamo che occorre prestare attenzione al fatto che con alcune configurazioni, in particolare nelle modalità 3 e 4, il volume dei costi è costruito considerando ogni possibile valore di disparità tra 0 e 63, ma un valore ogni due o quattro tra 64 e 127. Nel caso la disparità fornita esternamente non rientri tra le possibilità, scegliamo di azzerare il costo corrispondente al valore di disparità più vicino, o i costi corrispondenti ai due più vicini in caso di parità.

3.3 Test e risultati con sostituzione dei costi

Testiamo rSGM così modificato fornendo punti di disparità nota estratti dalla ground truth con la stessa modalità utilizzata con GC-Net, cioè campionando in maniera casuale e uniforme, prima con probabilità del 5 %, cioè un punto ogni venti, poi con probabilità dello 0.25 %, ovvero un punto ogni quattrocento.

Utilizziamo come dataset KITTI 2015 e 2012, Middlebury pesato e FlyingThings3D-TEST, limitandoci per quest'ultimo, al fine di contenere il tempo totale di esecuzione, solo al primo dei dieci frame di ogni scena. Come per GC-Net, escludiamo i casi in cui la ground truth supera 300 in almeno il 25 % dei punti, cioè tre coppie di frame nel sottoinsieme considerato di FlyingThings3D-TEST.

Per quanto riguarda le metriche per valutare l'algoritmo, utilizziamo ancora l'errore assoluto medio, EPE, e la percentuale delle disparità che si discostano di più di 3 pixel dalla ground truth, D1. Tuttavia l'output di rSGM presenta aree prive di informazione perché invalidate. Per il calcolo dell'EPE dobbiamo necessariamente limitarci ai punti con disparità valida, a meno di falsare notevolmente i risultati. Per i punti errati calcoliamo invece due versioni: una che come l'EPE si limita a considerare i punti con disparità valida, semplicemente D1, e una seconda, D1*, che considera tutti i punti e assegna disparità nulla a quelli non validi: in questo caso tutti i punti non validi nell'output ma con ground truth maggiore di 3 pixel saranno considerati errati. Ovviamente i punti per cui non è disponibile la ground truth non sono mai inclusi nel computo. Ciò che importa è in ogni caso il confronto con i

risultati ottenuti dall'algoritmo originario, per cui andiamo a calcolare le stesse metriche.

Di seguito sono elencati i risultati campionando al 5 % ed eseguendo rSGM nella modalità 2. Nella prima colonna è indicato il moltiplicatore. L'ultima riga mostra i valori di riferimento senza dati sparsi.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	3.49	6.40	18.35	0.99	2.95	9.24	1.05	2.99	10.66	1.53	5.31	16.90
x2	3.07	4.87	16.56	0.91	2.38	8.25	0.94	2.40	9.33	1.32	4.20	15.52
x10	2.85	4.09	15.83	0.88	2.18	7.97	0.88	2.14	8.98	1.21	3.59	14.93
x100	2.85	4.09	15.83	0.88	2.18	7.99	0.88	2.15	8.97	1.21	3.60	14.95
	4.36	*10.00*	*23.38*	*1.19*	*4.35*	*11.72*	*1.30*	*4.48*	*14.27*	*2.01*	*8.77*	*22.01*

Tabella 3.1 – Risultati ottenuti da rSGM in modalità 2 con suggerimenti estratti dalla ground truth campionando al 5 % e sostituzione dei costi.

La Figura 3.2, più avanti, mostra un esempio di output confrontato con l'output originario di rSGM.

Questi invece i risultati campionando al 5 % ma eseguendo rSGM in modalità 5.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	3.53	6.42	19.19	0.89	2.50	7.93	0.84	2.26	8.66	1.74	6.60	19.34
x2	3.21	5.30	17.96	0.85	2.18	7.52	0.80	1.98	8.22	1.55	5.51	18.13
x10	3.09	4.90	17.34	0.85	2.13	7.45	0.80	1.94	8.17	1.48	5.09	17.46
x100	3.09	4.90	17.35	0.85	2.12	7.47	0.80	1.93	8.18	1.47	5.07	17.46
	4.28	*9.88*	*23.54*	*1.09*	*3.91*	*10.02*	*1.04*	*3.52*	*11.29*	*2.18*	*9.96*	*23.76*

Tabella 3.2 – Risultati ottenuti da rSGM in modalità 5 con suggerimenti estratti dalla ground truth campionando al 5 % e sostituzione dei costi.

Il miglioramento registrato è superiore alla percentuale con cui è stata campionata la ground truth, pertanto il metodo è efficace. Notiamo che un moltiplicatore superiore a 10 non modifica i risultati.

Ora ripetiamo campionando la ground truth con probabilità dello 0.25 %. Questi i risultati in modalità 2.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	4.47	10.47	24.81	1.18	4.25	11.58	1.28	4.37	14.04	1.98	8.47	21.64
x2	4.41	10.21	24.49	1.17	4.17	11.48	1.27	4.27	13.88	1.94	8.19	21.33
x10	4.36	10.04	24.34	1.17	4.14	11.46	1.26	4.24	13.84	1.91	8.04	21.18
x100	4.36	10.04	24.34	1.17	4.15	11.45	1.26	4.24	13.84	1.92	8.08	21.21
	4.36	*10.00*	*23.38*	*1.19*	*4.35*	*11.72*	*1.30*	*4.48*	*14.27*	*2.01*	*8.77*	*22.01*

Tabella 3.3 – Risultati ottenuti da rSGM in modalità 2 con suggerimenti estratti dalla ground truth campionando allo 0.25 % e sostituzione dei costi.

Questi in modalità 5.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	4.32	9.64	23.54	1.07	3.80	9.90	1.03	3.40	11.11	2.14	9.58	23.43
x2	4.28	9.47	23.38	1.07	3.75	9.86	1.02	3.35	11.06	2.12	9.44	23.27
x10	4.26	9.41	23.27	1.07	3.74	9.85	1.02	3.35	11.06	2.10	9.36	23.10
x100	4.26	9.41	23.28	1.07	3.74	9.85	1.02	3.34	11.05	2.10	9.37	23.12
	4.28	*9.88*	*23.54*	*1.09*	*3.91*	*10.02*	*1.04*	*3.52*	*11.29*	*2.18*	*9.96*	*23.76*

Tabella 3.4 – Risultati ottenuti da rSGM in modalità 5 con suggerimenti estratti dalla ground truth campionando allo 0.25 % e sostituzione dei costi.

In questo caso il miglioramento, ove presente, è solo marginale, seppur superiore allo 0.25 %. Rispetto a GC-Net, d'altra parte, un algoritmo tradizionale come rSGM ha difficoltà a generalizzare dati esterni così sparsi ad aree abbastanza ampie, non essendo stato pensato con questo fine. Nel caso di FlyingThings3D dobbiamo talvolta perfino registrare un peggioramento.

3.4 Modifica di rSGM con modulazione dei costi

Se con GC-Net avevamo modificato il volume dei costi effettuando una modulazione dei costi di matching utilizzando una funzione Gaussiana centrata sul valore di disparità corretto, una soluzione non troppo radicale e in un certo senso elegante, nel caso di rSGM abbiamo invece considerato i dettagli dell'algoritmo e optato per un approccio più drastico, sostituendo completamente i costi dei punti suggeriti con i valori più convenienti. Vogliamo ora invece provare ad applicare il metodo della modulazione anche a rSGM, per mostrarne la generalità.

Ricordiamo però ancora una volta che se abbiamo supposto i costi di GC-Net rappresentare, al contrario del nome, misure di similarità tra feature map, lasciando alla rete il compito di adattarsi di conseguenza, questo non è il caso di rSGM, dove i costi, distanze di Hamming tra trasformate Census, sono utilizzati come tali. Allora la funzione modulante che andremo a scegliere dovrà essere prossima a zero in corrispondenza della disparità fornita e portarsi a valori elevati allontanandosi da essa.

Per mantenere il più possibile il parallelismo con la funzione Gaussiana utilizzata con GC-Net, scegliamo semplicemente la differenza tra una costante elevata e una Gaussiana dallo stesso massimo, centrata sulla disparità desiderata. Nello specifico, per i punti per cui è fornito un valore di disparità h_{ij}, moltiplichiamo i costi lungo la dimensione della disparità d per

$$k - k \cdot e^{-\frac{(d-h_{ij})^2}{2c^2}} = k \cdot \left(1 - e^{-\frac{(d-h_{ij})^2}{2c^2}}\right),$$

che assume appunto il valore 0 per $d = h_{ij}$ e si avvicina velocemente a k allontanandosi da h_{ij}. La Figura 3.1 ne mostra il grafico.

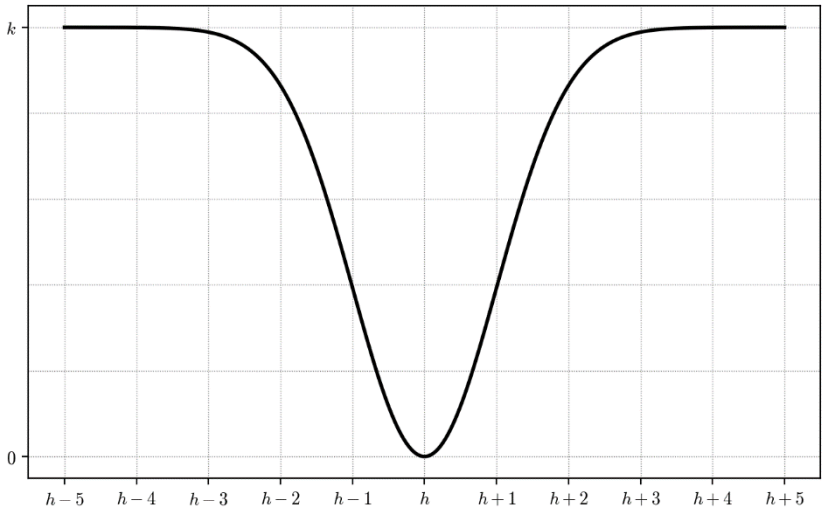

Figura 3.1 – Moltiplicatore del volume dei costi di rSGM, con $c = 1$.

3.5 Test e risultati con modulazione dei costi

Testiamo ora l'algoritmo nelle stesse modalità dell'esperimento precedente: forniamo disparità ottenute dal campionamento casuale e uniforme della ground truth con probabilità del 5 % e dello 0.25 %.

Per la scelta dei parametri della funzione modulante poniamo $c = 1$, mentre proviamo quattro diversi valori di k, ancora 1, 2, 10 e 100.

Di seguito sono riportati i risultati con la ground truth campionata al 5 % ed eseguendo rSGM in modalità 2. La prima colonna riporta il valore di k.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	4.09	8.73	21.53	1.13	3.93	10.96	1.21	3.95	12.90	1.85	7.51	20.26
x2	3.86	7.84	20.29	1.09	3.62	10.42	1.15	3.60	12.00	1.72	6.68	19.06
x10	3.15	5.33	17.24	0.95	2.74	8.94	0.98	2.64	9.92	1.37	4.64	16.33
x100	2.86	4.35	16.35	0.91	2.45	8.51	0.92	2.37	9.48	1.25	3.88	15.51
	4.36	*10.00*	*23.38*	*1.19*	*4.35*	*11.72*	*1.30*	*4.48*	*14.27*	*2.01*	*8.77*	*22.01*

Tabella 3.5 – Risultati ottenuti da rSGM in modalità 2 con suggerimenti estratti dalla ground truth campionando al 5 % e modulazione dei costi.

La Figura 3.2 mostra un confronto tra l'output di rSGM e l'output ottenuto con l'aiuto della ground truth, sia con l'approccio della sostituzione dei costi sia con quello della modulazione. Un'osservazione attenta è necessaria per notare le differenze, ma un buon punto di partenza è la strada, come è tipico nelle immagini di KITTI. In particolare appare presto evidente la differenza nel numero di disparità invalidate.

Questi invece i risultati campionando al 5 % ma eseguendo in modalità 5.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	4.11	8.89	22.11	1.04	3.59	9.54	0.99	3.18	10.46	2.07	9.08	22.48
x2	3.95	8.21	21.20	1.01	3.34	9.21	0.95	2.95	9.97	1.99	8.51	21.70
x10	3.42	6.20	18.83	0.92	2.67	8.24	0.86	2.36	8.89	1.68	6.62	19.29
x100	3.07	5.10	17.89	0.88	2.36	7.91	0.83	2.15	8.60	1.51	5.51	18.08
	4.28	*9.88*	*23.54*	*1.09*	*3.91*	*10.02*	*1.04*	*3.52*	*11.29*	*2.18*	*9.96*	*23.76*

Tabella 3.6 – Risultati ottenuti da rSGM in modalità 5 con suggerimenti estratti dalla ground truth campionando al 5 % e modulazione dei costi.

Al crescere del valore di k i risultati diventano sempre più paragonabili al caso precedente con costi fissati. D'altra parte per k elevati l'effetto sul volume dei costi si avvicina all'operazione di sostituzione dei costi.

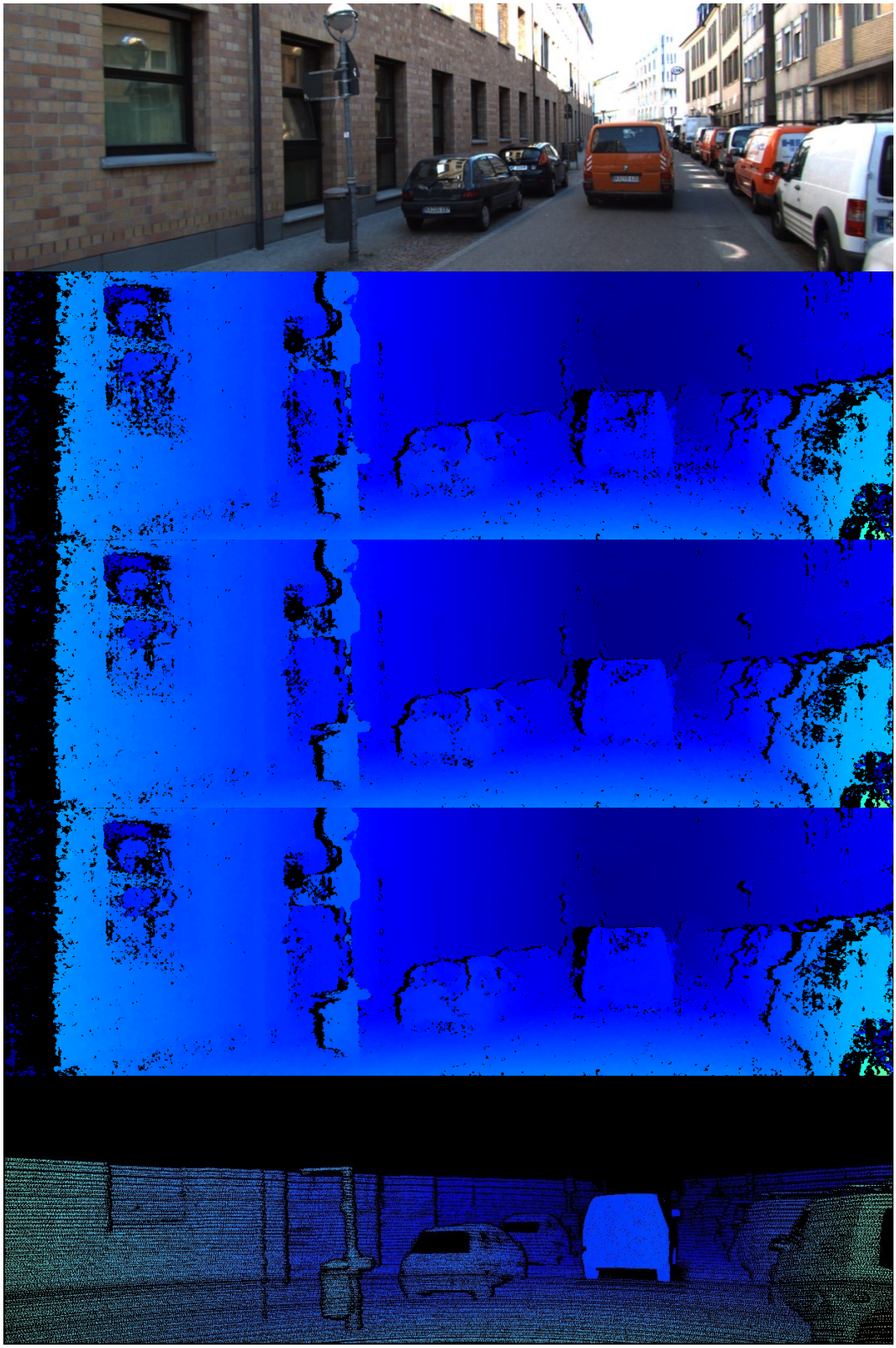

Figura 3.2 – L'output di rSGM in modalità 2 a confronto su un input di KITTI 2015 con gli output con ground truth campionata al 5 % e rispettivamente sostituzione e modulazione dei costi. Dall'alto verso il basso: vista sinistra; output di rSGM; output con sostituzione dei costi; output con modulazione dei costi; ground truth.

Passiamo invece a campionare la ground truth allo 0.25 %. Questi i risultati in modalità 2.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	4.51	10.66	25.08	1.19	4.32	11.68	1.29	4.45	14.19	2.00	8.69	21.91
x2	4.49	10.59	24.96	1.19	4.31	11.66	1.29	4.42	14.14	1.99	8.62	21.84
x10	4.40	10.23	24.54	1.18	4.23	11.55	1.27	4.33	13.96	1.95	8.34	21.53
x100	4.36	10.09	24.41	1.17	4.19	11.51	1.26	4.29	13.89	1.93	8.20	21.38
	4.36	*10.00*	*23.38*	*1.19*	*4.35*	*11.72*	*1.30*	*4.48*	*14.27*	*2.01*	*8.77*	*22.01*

Tabella 3.7 – Risultati ottenuti da rSGM in modalità 2 con suggerimenti estratti dalla ground truth campionando allo 0.25 % e modulazione dei costi.

E questi i risultati in modalità 5.

	FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
x1	4.36	9.87	23.76	1.09	3.89	10.00	1.04	3.50	11.25	2.17	9.91	23.68
x2	4.35	9.81	23.68	1.08	3.87	9.97	1.04	3.48	11.21	2.17	9.86	23.62
x10	4.29	9.55	23.40	1.08	3.81	9.91	1.03	3.42	11.11	2.14	9.67	23.41
x100	4.25	9.43	23.32	1.07	3.78	9.88	1.02	3.40	11.08	2.11	9.45	23.21
	4.28	*9.88*	*23.54*	*1.09*	*3.91*	*10.02*	*1.04*	*3.52*	*11.29*	*2.18*	*9.96*	*23.76*

Tabella 3.8 – Risultati ottenuti da rSGM in modalità 5 con suggerimenti estratti dalla ground truth campionando allo 0.25 % e modulazione dei costi.

I miglioramenti con una percentuale di campionamento così bassa sono sempre marginali, ma del resto non potevamo aspettarci un miglioramento rispetto al caso con costi sostituiti. Notiamo anzi che le prestazioni su FlyingThings3D della modalità 2 sono peggiori di quelle di riferimento per qualsiasi valore di k.

Riportiamo qui una sintesi dei risultati, scegliendo il valore del moltiplicatore o di k più efficace. Per la modalità 2:

		FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
		EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
$\frac{1}{20}$	sost	2.85	4.09	15.83	0.88	2.18	7.97	0.88	2.14	8.98	1.21	3.59	14.93
	gauss	2.86	4.35	16.35	0.91	2.45	8.51	0.92	2.37	9.48	1.25	3.88	15.51
$\frac{1}{400}$	sost	4.36	10.04	24.34	1.17	4.14	11.46	1.26	4.24	13.84	1.91	8.04	21.18
	gauss	4.36	10.09	24.41	1.17	4.19	11.51	1.26	4.29	13.89	1.93	8.20	21.38
		4.36	*10.00*	*23.38*	*1.19*	*4.35*	*11.72*	*1.30*	*4.48*	*14.27*	*2.01*	*8.77*	*22.01*

Tabella 3.9 – Confronto dei risultati ottenuti da rSGM in modalità 2 con suggerimenti estratti dalla ground truth campionando al 5 % o allo 0.25 % e sostituzione o modulazione dei costi.

Mentre in modalità 5:

		FlyingThings3D			KITTI 2015			KITTI 2012			Middlebury		
		EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*	EPE	D1	D1*
$\frac{1}{20}$	sost	3.09	4.90	17.34	0.85	2.13	7.45	0.80	1.94	8.17	1.48	5.09	17.46
	gauss	3.07	5.10	17.89	0.88	2.36	7.91	0.83	2.15	8.60	1.51	5.51	18.08
$\frac{1}{400}$	sost	4.26	9.41	23.27	1.07	3.74	9.85	1.02	3.35	11.06	2.10	9.36	23.10
	gauss	4.25	9.43	23.32	1.07	3.78	9.88	1.02	3.40	11.08	2.11	9.45	23.21
		4.28	*9.88*	*23.54*	*1.09*	*3.91*	*10.02*	*1.04*	*3.52*	*11.29*	*2.18*	*9.96*	*23.76*

Tabella 3.10 – Confronto dei risultati ottenuti da rSGM in modalità 5 con suggerimenti estratti dalla ground truth campionando al 5 % o allo 0.25 % e sostituzione o modulazione dei costi.

Possiamo quindi affermare di essere riusciti a integrare con successo valori sparsi ma esatti di disparità in algoritmi stereo, sia reti neurali convoluzionali sia algoritmi tradizionali.

4 GC-NET CON PUNTI A DISPARITÀ CON CONFIDENZA ELEVATA

4.1 Una misura di confidenza per le mappe di disparità: CCNN

Abbiamo in precedenza accennato che molti algoritmi stereo, tipicamente quelli tradizionali, effettuano un'ultima fase di raffinamento della mappa di disparità, uno dei cui scopi è quello di invalidare o correggere le disparità di punti o aree che presentano errori evidenti. Tra i metodi utilizzati rientrano ad esempio il left-right consistency check e la rimozione dei picchi, che elimina piccole aree in cui la disparità differisce nettamente da quella dei punti circostanti.

Un diverso approccio per riconoscere disparità errate è quello di calcolare, per ogni punto nella mappa di disparità, una misura di confidenza, che codifichi il grado di certezza o incertezza di quel punto. Una volta ottenuta la mappa delle confidenze si può procedere a invalidare le disparità di tutti quei punti che non raggiungono una soglia di confidenza fissata.

Molti dei metodi proposti per generare una mappa di confidenza utilizzano, oltre alla mappa di disparità finale, anche risultati intermedi calcolati dallo specifico algoritmo, quali il volume dei costi, alla ricerca di determinati pattern che siano indicativi di un probabile errore [14]. Anche in questo ambito l'utilizzo di tecniche di machine learning ha consentito di migliorare in maniera significativa i risultati precedenti: ad esempio Park & Yoon impiegano molteplici misure di confidenza combinate in una foresta causale (*random forest*) [15].

La necessità di disporre di risultati intermedi fa però sì che questi metodi siano applicabili solamente a sistemi aperti, e non siano invece utilizzabili su sensori di disparità che, ad esempio perché proprietari, non forniscono il volume dei costi. La proposta di CCNN (*Confidence Convolutional Neural Network*) si prefigge gli obiettivi di superare questa limitazione e al contempo di migliorare l'accuratezza dei metodi precedenti utilizzando una rete neurale convoluzionale che produca una mappa di confidenza a partire solamente da una mappa di disparità. L'osservazione chiave di partenza è che determinati pattern ricorrenti nelle mappe di disparità sono quasi sempre sufficienti per distinguere valori corretti da valori errati: una rete neurale convoluzionale può imparare a riconoscere efficacemente simili pattern e conseguentemente a restituire opportuni valori di confidenza. [16] [17]

CCNN stima la confidenza del singolo pixel basandosi sul pattern formato dai valori di disparità in una finestra quadrata $N \times N$ centrata su quel pixel. La generazione della mappa di confidenza avviene considerando ogni punto della mappa di disparità separatamente: per ogni punto, i valori di disparità nel suo intorno $N \times N$ vengono normalizzati tra 0 e 1 e forniti a CCNN, e l'output inserito nella mappa di confidenza finale.

L'architettura di CCNN è relativamente poco profonda rispetto a reti neurali utilizzate per task di più alto livello. L'input è un unico canale di dimensione $N \times N$. Una prima parte della rete è costituita da $\frac{N-1}{2}$ layer convoluzionali, ciascuno dei quali applica F filtri 3×3 ed è seguito dalla funzione di attivazione ReLU. Poiché lo stride è unitario e non è impiegata alcuna forma di padding, ognuno di questi layer riduce la dimensione del proprio input di due pixel per lato: l'output dell'ultimo di questi livelli ha quindi dimensione $1 \times 1 \times F$, cioè è un insieme di F feature map che constano di un solo elemento ciascuna.

Questi F valori vengono passati alla seconda parte della rete, in cui si susseguono due layer fully connected di L neuroni ciascuno con funzione di attivazione ReLU, e un ultimo livello fully connected con un solo neurone e funzione di attivazione sigmoidale, che produce il valore finale di confidenza, tra 0 e 1. Utilizzando una metodologia ricorrente nel caso delle reti convoluzionali, i layer fully connected sono in realtà implementati in maniera equivalente come livelli convoluzionali che applicano L kernel 1×1. La struttura esatta della rete è sintetizzata nella seguente tabella ed è schematizzata in Figura 4.1.

Nome	*Input*	*Kernel*	*Stride*	*Attivazione*	*Output*
input	Input				$N \times N \times 1$
conv1	input	3×3	1	ReLU	$(N-2) \times (N-2) \times F$
conv2	conv1	3×3	1	ReLU	$(N-4) \times (N-4) \times F$
⋮	⋮	⋮	⋮	⋮	⋮
conv$\frac{N-1}{2}$	conv$\frac{N-3}{2}$	3×3	1	nessuna	$1 \times 1 \times F$
fc1	conv$\frac{N-1}{2}$	1×1	1	ReLU	$1 \times 1 \times L$
fc2	fc1	1×1	1	ReLU	$1 \times 1 \times L$
output	fc2	1×1	1	sigmoide	$1 \times 1 \times 1$

Tabella 4.1 – Architettura di CCNN.

L'utilizzo di layer convoluzionali al posto di layer fully connected ha l'importante vantaggio di permettere l'utilizzo della rete per produrre una mappa di confidenza completa in un unico passaggio, fornendo in input la mappa di disparità intera con un padding di $\frac{N-1}{2}$ intorno: in questo modo, infatti, le dimensioni $w \times h$ dell'input

prima del padding sono portate in output. Processare un'unica mappa $w \times h$ piuttosto che $w \times h$ finestre $N \times N$ consente di ridurre drasticamente il tempo necessario per produrre una mappa di confidenza, poiché numerosi risultati intermedi sono riutilizzati.

Dalla valutazione del comportamento della rete è emerso che l'impiego di finestre $N \times N = 9 \times 9$, e dunque di quattro layer convoluzionali, di $F = 64$ filtri per ognuno di questi layer, e di $L = 100$ neuroni per ogni layer fully connected è sufficiente per massimizzarne l'efficacia. Sono dunque questi i valori dei parametri cui faremo riferimento.

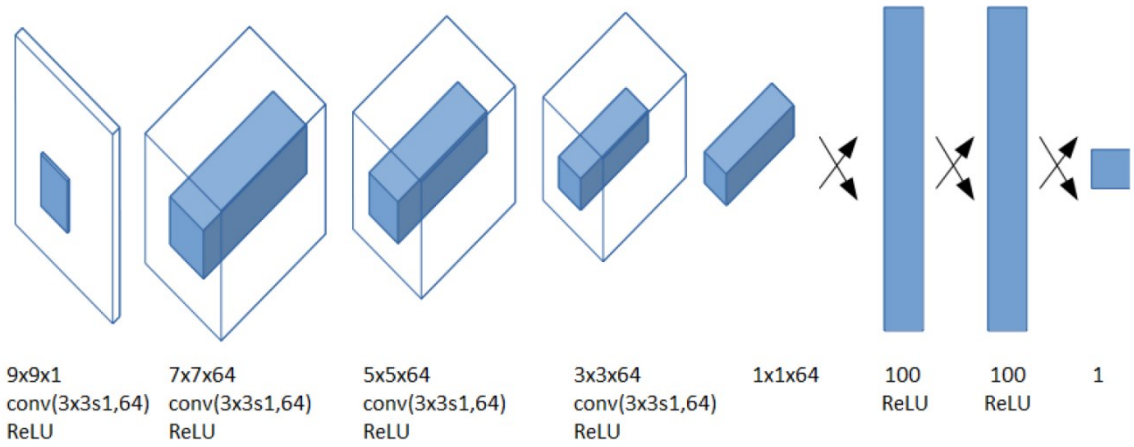

Figura 4.1 – Architettura di CCNN.

4.2 Integrazione di CCNN in GC-Net modificata

Nel paragrafo 2.6 abbiamo sperimentato l'approccio di supportare GC-Net con dati di disparità sparsi prelevati non più dalla ground truth, bensì dall'output di SGM e di rSGM, in un primo tentativo di combinare i punti di forza di una CNN con quelli di un algoritmo tradizionale e di sopperire viceversa alle reciproche debolezze. Sfortunatamente, il risultato generale è stato un peggioramento dell'errore medio su tutti i dataset di test, nonché un aumento della percentuale di errori, fatta eccezione solamente per i dataset KITTI utilizzando SGM. In effetti i suggerimenti forniti da SGM e rSGM, affetti dagli errori prodotti dai due algoritmi, erano campionati in maniera casuale ed uniforme, pertanto valori corretti, lievemente imprecisi o completamente errati venivano scelti con la stessa probabilità. Poiché i suggerimenti sono iniettati direttamente nel volume dei costi, GC-Net non ha modo di distinguere tra corretti ed errati, se non utilizzando i risultati di disparità che la rete sarebbe in grado di produrre autonomamente. Ora vogliamo invece indagare la possibilità di estrarre disparità da rSGM scegliendo i punti con maggiore probabilità di essere corretti, basandoci su una misura di confidenza.

Calcoliamo le confidenze utilizzando CCNN: data la mappa di disparità prodotta da rSGM, ne ricaviamo la mappa di confidenza. Dopodiché procediamo come segue. Escludiamo a priori tutti i punti con confidenza inferiore a una determinata soglia. Tra quelli rimasti, forniamo a GC-Net i punti che, facendo scorrere in ogni posizione possibile una finestra di dimensione fissa, hanno all'interno della finestra la confidenza massima (si noti che, in questo modo, applicazioni ravvicinate della finestra risultano spesso nella scelta dello stesso punto). In caso di più massimi di confidenza uguali, ed è solitamente il caso di aree con confidenza satura a 1, scegliamo casualmente. L'effetto di estrarre il massimo da una finestra scorrevole è quello di produrre ancora dati di disparità sparsi. Il risultato finale è che i punti scelti sono quelli con confidenza superiore alla soglia e massima in almeno una finestra che li contiene.

Per integrare le due reti, una possibilità è quella di addestrare separatamente CCNN, e utilizzarla per produrre di volta in volta i suggerimenti per GC-Net. Un'altra possibilità, che è quella che utilizzeremo, è quella di unire effettivamente le due reti, che avranno a questo punto una funzione di loss comune e saranno addestrate contestualmente, con una fase unica di backpropagation. In particolare,

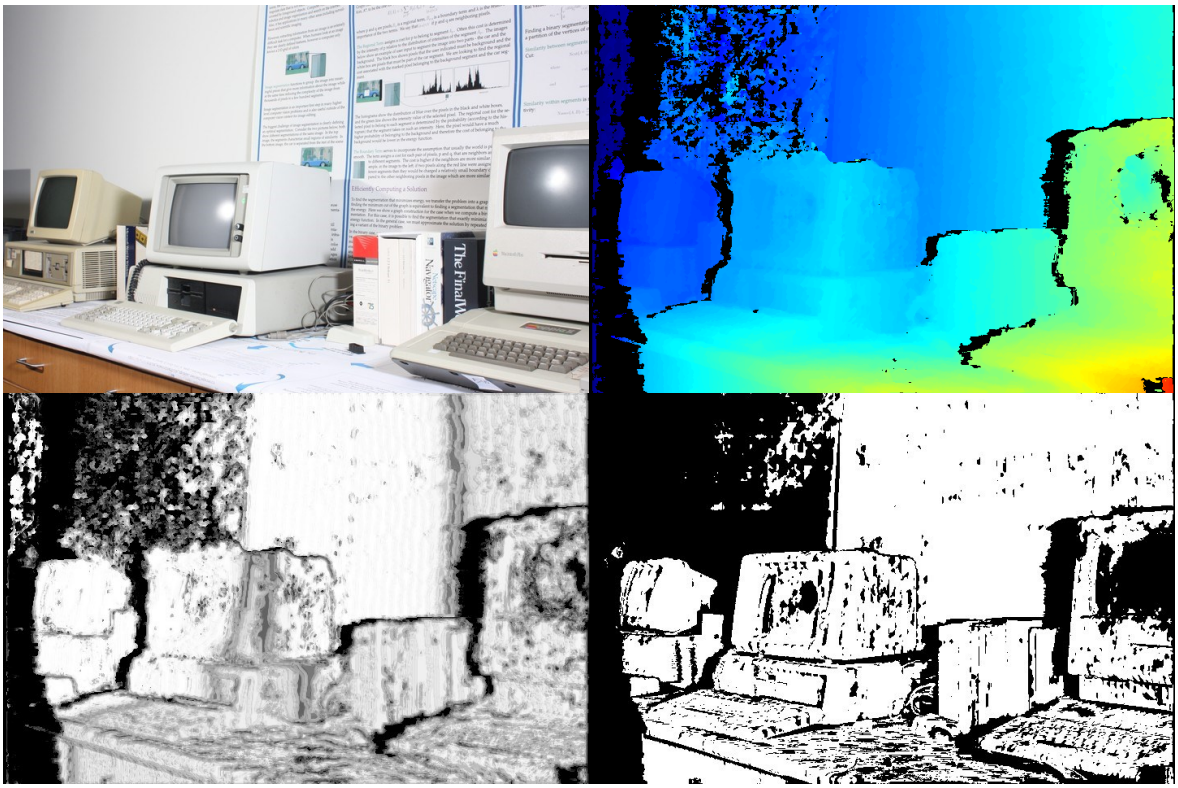

Figura 4.2 – Output e ground truth di CCNN per un input di Middlebury.
In alto a sinistra: immagine sinistra; in alto a destra: output di rSGM (e input non normalizzato di CCNN); in basso a sinistra: output di CCNN; in basso a destra: ground truth della confidenza.

la funzione di loss complessiva sarà semplicemente la somma della loss di CCNN e della loss di GC-Net: se poi si desidera una learning rate differente per le due reti, è sufficiente moltiplicare uno dei due termini per una costante. In ogni caso, poiché CCNN converge estremamente più in fretta di GC-Net, quest'ultima ha poi tutto il tempo di adattarsi all'output della prima.

Per quanto riguarda la funzione di loss di CCNN, l'output della rete è un valore tra 0 e 1 prodotto da una funzione di attivazione sigmoidale, ma dobbiamo costruire la relativa ground truth. Poiché idealmente la confidenza di un punto dovrebbe essere 0 o 1 a seconda rispettivamente che la sua disparità sia errata o corretta, la ground truth risulta essere una mappa di zeri e uno costruita secondo lo stesso criterio. Un esempio di mappa di confidenza è mostrato in Figura 4.2. Più precisamente consideriamo corrette, quindi affidabili, le disparità che differiscono dalla disparità reale di non più di un pixel. CCNN è così di fatto un classificatore binario, dunque ha senso utilizzare come funzione di loss la cross-entropy binaria, ovvero

$$Loss_{XE}(y, t) = -(t \log(y) + (1 - t) \log(1 - y)) ,$$

dove y è l'output e t la ground truth e dove si definisce $0 \cdot \log 0 = 0$.

4.3 Training e risultati

Procediamo con l'addestramento e la valutazione della nuova rete, domandandoci se fornire disparità sparse ad alta confidenza prelevate dall'output di rSGM sia sufficiente per migliorare la precisione di GC-Net.

Insieme alla misura dell'accuratezza della rete, utilizzando i soliti indicatori EPE e D1 con soglia di tre pixel, vogliamo anche monitorare il comportamento di CCNN nel selezionare punti affidabili. Una volta fissata una soglia di confidenza, CCNN è a tutti gli effetti un classificatore binario, dunque valgono i concetti di vero positivo (*true positive*, TP), vero negativo (*true negative*, TN), falso positivo (*false positive*, FP), falso negativo (*false negative*, FN), e di tutte le numerose metriche che ne derivano. Tra queste ne scegliamo tre che riteniamo significative per il nostro problema: il tasso di errore (*error rate*)

$$ERR = \frac{FP + FN}{TP + TN + FP + FN};$$

la *false discovery rate*

$$FDR = \frac{FP}{TP + FP},$$

che misura quanti, tra i punti supposti affidabili, sono in realtà non affidabili, cioè la percentuale di suggerimenti errati; la *false negative rate*

$$FNR = \frac{FN}{TP + FN},$$

che misura quanti punti, tra quelli realmente affidabili, sono stati considerati non affidabili, cioè la percentuale di punti affidabili lasciati indietro.

Manteniamo i parametri predefiniti delle reti coinvolte. Per GC-Net abbiamo numero di disparità $D = 192$ e numero di unary feature $F = 32$. Per CCNN abbiamo dimensione dell'intorno per la valutazione della confidenza su un punto $N \times N = 9 \times 9$, numero di feature map dei livelli convoluzionali $F = 64$ e dimensione dei livelli fully connected $L = 100$. Per la Gaussiana con cui moduliamo il volume dei costi di GC-Net confermiamo $k = 10$ e $c = 1$.

Iniziamo fissando una soglia di confidenza piuttosto alta, 0.9, e optando per una finestra per la scelta del massimo di confidenza di dimensione 5×5: ciò equivale, nelle zone ad alta confidenza, a scegliere in media il 4 % dei punti, percentuale solo lievemente inferiore a quella utilizzata per la maggior parte degli esperimenti nel capitolo 2. Addestriamo come al solito su FlyingThings3D-TRAIN per 150 000 iterazioni con learning rate 0.001. Le disparità suggerite sono sempre estratte da rSGM. I risultati sui soliti dataset sono i seguenti. Tra parentesi è sempre indicato il riferimento di GC-Net originaria.

	EPE	*D1 % (>3)*	*FDR %*	*FNR %*
FlyingThings3D-TEST	2.075 (2.14)	6.816 (6.51)	3.95	20.73
KITTI 2015	2.041 (3.14)	9.843 (23.82)	21.87	20.42
KITTI 2012	1.791 (2.91)	10.323 (22.75)	10.35	23.20
Middlebury (pesato)	2.482	12.971	8.73	19.37

Tabella 4.2 – Risultati ottenuti da GC-Net addestrata e testata con suggerimenti estratti dall'output di rSGM scegliendo i punti con confidenza calcolata da CCNN superiore alla soglia di 0.9 e massima in almeno una finestra 5×5 *che li contiene.*

I risultati su FlyingThings3D non si discostano molto dal riferimento di GC-Net: l'EPE è sì marginalmente migliore, ma al contrario la percentuale di punti errati è peggiore. Poiché la false detection rate è molto bassa e anche la false negative rate piuttosto buona, ciò significa che su questo dataset GC-Net non può trarre grandi benefici da rSGM.

È su KITTI 2015 e KITTI 2012 che otteniamo invece risultati molto soddisfacenti. La Figura 4.3 mostra il confronto tra l'output della nostra rete e quello di GC-Net su un input di KITTI 2015, oltre che gli output intermedi come quello di rSGM e di CCNN:

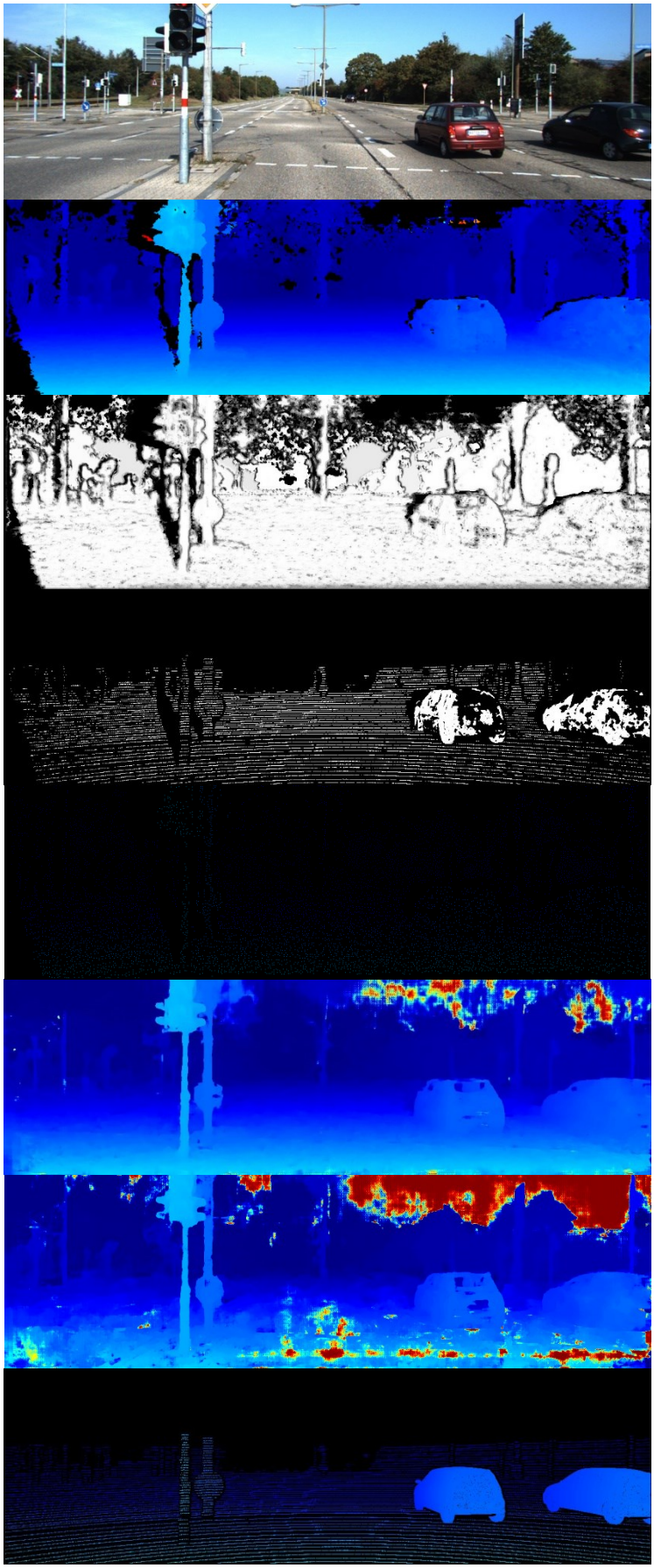

Figura 4.3 – Output intermedi e finale su un input di KITTI 2015 della rete ottenuta integrando GC-Net e CCNN e confronto con l'output della sola GC-Net.

Dall'alto verso il basso:
vista sinistra;
output di rSGM;
output di CCNN;
ground truth della confidenza;
disparità fornite alla rete;
output della rete;
output di GC-Net;
ground truth.

le differenze sono lampanti, in particolare sulle aree più complesse come la strada e il cielo, che GC-Net non ha imparato a riconoscere con precisione dall'addestramento sul molto diverso FlyingThings3D-TRAIN e dove viceversa rSGM tende a commettere meno errori (anche se il cielo, ricordiamo, non influenza il risultato, poiché la ground truth è assente). Può anche essere interessante osservare come la mappa di disparità nella Figura 4.3 sia visibilmente più precisa e uniforme rispetto a quella mostrata in Figura 2.10, prodotta sulla stessa coppia di immagini ma dove le disparità sparse fornite alla rete sono state estratte campionando SGM in maniera casuale.

Visti i buoni risultati, vogliamo ora ottimizzare l'accuratezza sul solo KITTI 2015, effettuando 50 000 ulteriori iterazioni di fine-tuning su KITTI 2012, nelle stesse modalità descritte nel paragrafo 2.5. Otteniamo infine i seguenti risultati, che rispettano le nostre attese.

	EPE	*D1 % (>3)*	*FDR %*	*FNR %*
KITTI 2015	1.459	3.636	16.93	40.66

Tabella 4.3 – Risultati ottenuti da GC-Net addestrata, fine-tuned su KITTI 2012 e testata con suggerimenti estratti dall'output di rSGM scegliendo i punti con confidenza calcolata da CCNN superiore alla soglia di 0.9 e massima in almeno una finestra 5×5 *che li contiene.*

Come ultimo esperimento proviamo ad abbassare la soglia di confidenza da 0.9 a 0.75, ripetendo per intero training e testing con tutti gli altri parametri invariati. I nuovi risultati sono i seguenti:

	EPE	*D1 % (>3)*	*FDR %*	*FNR %*
FlyingThings3D-TEST	2.419 (2.14)	7.522 (6.51)	6.17	16.68
KITTI 2015	2.093 (3.14)	9.693 (23.82)	23.17	24.43
KITTI 2012	2.327 (2.91)	10.850 (22.75)	11.22	25.72
Middlebury (pesato)	2.801	14.010	8.98	16.60

Tabella 4.4 – Risultati ottenuti da GC-Net addestrata e testata con suggerimenti estratti dall'output di rSGM scegliendo i punti con confidenza calcolata da CCNN superiore alla soglia di 0.75 e massima in almeno una finestra 5×5 *che li contiene.*

Possiamo osservare come la precisione della rete sia peggiorata su tutti i dataset, arrivando su FlyingThings3D ad essere anche inferiore rispetto a quella della sola GC-Net. Ciò testimonia quanto già osservato nel paragrafo 2.6: al fine di supportare positivamente una rete stereo con disparità esterne è necessario che queste siano molto accurate, viceversa si ottiene un peggioramento delle prestazioni. Nel caso della nostra combinazione di CCNN e GC-Net, questo significa che ampie aree prive di suggerimenti di disparità, che lascino operare GC-Net in maniera

autonoma, sono da preferire rispetto a iniettare nel volume dei costi disparità con una discreta probabilità di essere errate.

Tutta la nostra trattazione ha scelto come rete di riferimento GC-Net, in quanto sufficientemente semplice da richiedere un tempo di addestramento accettabile per effettuare molteplici tentativi. Ma GC-Net è stato solo un esperimento per verificare se il nostro approccio si possa applicare a una rete già più accurata, quale la molto più complessa PSMNet (*Pyramid Stereo Matching Network*) [18]. Ovviamente una rete con una precisione di partenza migliore avrà dal nostro approccio un margine di miglioramento molto più ridotto, potenzialmente nullo, ma i presupposti sono incoraggianti.

5 CONCLUSIONI

Così come per innumerevoli altri problemi di computer vision e percettivi in generale, anche per il problema della visione stereo le reti neurali convoluzionali si sono ormai affermate come la scelta più accurata e vantaggiosa rispetto all'approccio degli algoritmi tradizionali basati su regole statiche. Ciò nonostante, non si è ancora giunti a un grado di accuratezza pienamente soddisfacente o affidabile per tutti gli ambiti di applicazione: in diversi contesti reali la precisione di un sensore di profondità come un LiDAR è ancora lontana. D'altra parte, però, questi dispositivi risultano particolarmente costosi e non sono comunque in grado di produrre informazioni oltre una certa densità.

Il filo conduttore di questa tesi è stata l'esplorazione della possibilità di supportare con dati di disparità sparsi di origine esterna algoritmi stereo che producano mappe di disparità dense, nel tentativo di migliorarne l'accuratezza. In una prima fase abbiamo utilizzato una rete neurale convoluzionale per la visione stereo, GC-Net, e dati ottenuti campionando casualmente e in maniera sparsa la ground truth, simulando l'utilizzo di un sensore di profondità a bassa risoluzione: ciò ha permesso di verificare il nostro approccio di modulare opportunamente il volume dei costi costruito da GC-Net con i dati sparsi e ha mostrato miglioramenti molto incoraggianti rispetto ai risultati di riferimento.

Visti i successi ottenuti con questo metodo, lo abbiamo sperimentato anche sull'algoritmo tradizionale rSGM, sfruttando il fatto che anch'esso costruisce un volume dei costi per il calcolo delle disparità finali: seppur non così netti come nel caso di GC-Net, i risultati sono stati piuttosto buoni, e hanno confermato la validità del nostro approccio nel fondere dati di disparità sparsi e algoritmi stereo, tradizionali o CNN.

A questo punto ci siamo allontanati dall'utilizzo della ground truth e abbiamo esaminato la possibilità di utilizzare la stessa metodologia per integrare una CNN stereo e un algoritmo tradizionale, con l'idea che questo potesse sopperire ai punti deboli della prima. Limitarsi a campionare casualmente l'output di SGM o rSGM e fornire queste disparità, non esatte, a GC-Net nelle medesime modalità impiegate nel caso della ground truth non ha dato i frutti sperati, causando in generale un peggioramento anche significativo dei risultati.

Del resto, estrarre casualmente informazioni da una mappa non accurata era un inizio ma certo non la strategia ottimale: per far sì che rSGM porti un contributo significativo è ragionevole volersi limitare a prelevare dal suo output le disparità più promettenti, con la maggior probabilità di essere precise. A tale scopo abbiamo allora introdotto un'altra rete neurale convoluzionale, CCNN, che, data una mappa di disparità, produce per ogni punto un valore di confidenza: adesso abbiamo potuto fornire a GC-Net i punti della mappa prodotta da rSGM con confidenza superiore a una determinata soglia e massima nel loro intorno. Se questo metodo non ha avuto un impatto rilevante sui risultati di GC-Net di fronte a dataset generici, ha invece portato un miglioramento dell'accuratezza di più del 35 % su dataset relativi all'ambito di particolare interesse di scenari di guida stradale.

Numerose CNN per la visione stereo sono ad oggi disponibili, diverse più precise di GC-Net, che è stata un utile banco di prova per testare la validità del nostro approccio, visto ad esempio il tempo di addestramento contenuto. Un naturale sviluppo è a questo punto integrare rSGM e CCNN in alcune di queste reti più precise, per verificare se il metodo delle confidenze possa portare benefici anche partendo da una migliore accuratezza di base.

6 BIBLIOGRAFIA

[1] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, T. Brox. **A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation.** *CVPR 2016*. arXiv:1512.02134.

[2] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, T. Brox. **FlowNet: Learning Optical Flow with Convolutional Networks.** *ICCV 2015*. arXiv:1504.06852.

[3] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox. **FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks.** *CVPR 2017*. arXiv:1612.01925.

[4] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, A. Bry. **End-to-End Learning of Geometry and Context for Deep Stereo Regression.** *ICCV 2017*: 66-75. arXiv:1703.04309.

[5] K. He, X. Zhang, S. Ren, J. Sun. **Deep Residual Learning for Image Recognition.** arXiv:1512.03385. 2015.

[6] A. Geiger, P. Lenz, R. Urtasun. **Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite.** *CVPR 2012*.

[7] M. Menze, A. Geiger. **Object Scene Flow for Autonomous Vehicles.** *CVPR 2015*.

[8] D. Scharstein, R. Szeliski. **A taxonomy and evaluation of dense two-frame stereo correspondence algorithms.** *International Journal of Computer Vision*, 47(1/2/3):7-42, April-June 2002.

[9] Martín Abadi et al. **Tensorflow: Large-scale machine learning on heterogeneous distributed systems.** arXiv:1603.04467 (2016).

[10] Heiko Hirschmüller. **Stereo Processing by Semiglobal Matching and Mutual Information.** *IEEE Transactions on Pattern Analysis and machine Intelligence*, vol. 30, no. 2, pp. 328-341, Feb 2008.

[11] R. Spangenberg; T. Langner; S. Adfeldt, R. Rojas. **Large scale Semi-Global Matching on the CPU Intelligent Vehicles Symposium Proceedings.** *2014 IEEE*, 2014, 195-201.

[12] Implementazione di rSGM:
https://github.com/ivankreso/stereo-vision/tree/master/reconstruction/base/rSGM

[13] R. Zabih, J. Woodfill. **Non-parametric Local Transforms for Computing Visual Correspondence.** *Proceedings of European Conference on Computer Vision*, (May):151–158, 1994.

[14] X. Hu, P. Mordohai. **A quantitative evaluation of confidence measures for stereo vision.** *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 2121–2133, 2012.

[15] Min-Gyu Park, Kuk-Jin Yoon. **Leveraging stereo matching with learning-based confidence measures.** In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[16] M. Poggi, S. Mattoccia. **Learning from scratch a confidence measure.** *BMVC 2016*.

[17] Implementazione di CCNN:
https://github.com/fabiotosi92/CCNN-Tensorflow

[18] Jia-Ren Chang, Yong-Sheng Chen. **Pyramid Stereo Matching Network.** arXiv:1803.08669. 2018.