ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO

CORSO DI LAUREA

TESI DI LAUREA

In

Sistemi Embedded Riconfigurabili M

CNN per view synthesis da mappe depth

RELATORE:

Rossetto Andrea

Prof. Stefano Mattoccia

CORRELATORI:

Tosi Fabio

Poggi Matteo

Anno Accademico 2017/18

Sessione II

Sommario

1 Introduzione

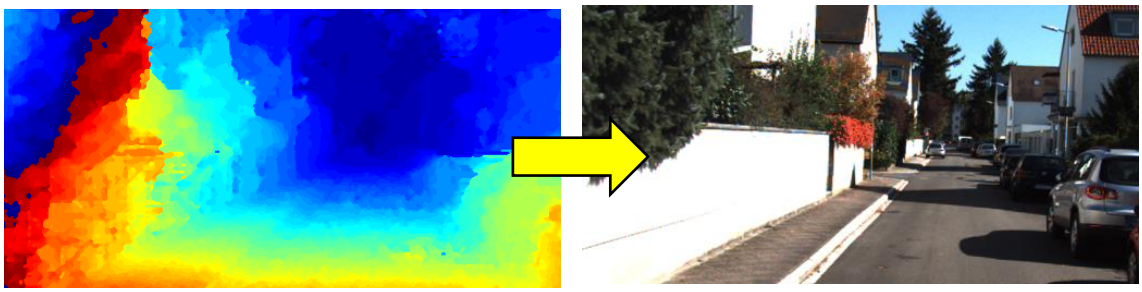
La visione artificiale è da sempre fonte di ricerca, uno degli scopi principali è quello di definire e ricostruire informazioni d'interesse del mondo, impiegando una o più immagini che rappresentano il mondo stesso. Nella visione artificiale possiamo trovare molte branche, tra queste troviamo la stereo visione. La stereo visione mira al ripristino delle informazioni di carattere tridimensionale, le quali vengono inevitabilmente perse durante il processo di acquisizione delle immagini. Nel corso degli anni sono state sviluppate molte tecniche con il fine di migliorare sempre di più la qualità della stima della profondità. Dispositivi come telecamere stereo, sensori ad infrarossi, LIDAR hanno permesso di risolvere molti dei problemi in modo più o meno efficace.

Figura 1: A sinistra un LIDAR, a destra un sistema di telecamere stereo

Questi dispositivi tuttavia spesso risultano essere molto costosi, e molto spesso gli algoritmi utilizzati comportano l'utilizzo di sistemi di elaborazione dedicati. Inoltre, non sempre sono in grado di restituire immagini accurate, il LIDAR ad esempio non è in grado di fornire una rappresentazione densa dello spazio 3D. Tutto questo ha portato allo sviluppo di sistemi in grado di stimare la profondità a partire dalla sua rappresentazione come immagine monoculare. L'utilizzo di immagini monoculari permette di ridurre i costi per l'hardware, ma allo stesso tempo incrementa la difficoltà negli algoritmi di stima della profondità. Nello stesso periodo in cui gli algoritmi hanno iniziato a migliorarsi molto velocemente e sono iniziati a nascerne di nuovi per l'utilizzo di immagini monoculari, il mondo della visione artificiale ha trovato uno strumento in grado di

incrementare le prestazioni e la precisione nell'analizzare le immagini, questo strumento è il **machine learning**. In particolare, vi è un uso molto intenso delle reti neurali profonde.

L'obiettivo di questo lavoro è lo sviluppo di un sistema di machine learning in grado di generare immagini a colori realistiche a partire dalle mappe di disparità[1].



Riuscire a generare delle immagini dalle profondità, senza sapere nulla sugli oggetti presenti nella scena, presenta diverse difficoltà, tra le maggiori vi sono quelle riguardanti i dataset da utilizzare, di come strutturare il sistema affinché funzioni correttamente e quindi la scelta del modello che si adatti al meglio al tipo di task. Al fine di riuscire a ottenere delle buone immagini sono stati testati diversi modelli di rete, alcuni dei quali realizzati partendo da lavori il cui obiettivo era quello di stimare la profondità partendo dalla singola immagine e altri in cui l'obiettivo era completamente differente dal nostro, ma che presentava caratteristiche che potevano tornare utili. Al fine di illustrare al meglio il lavoro svolto, questa tesi è stata strutturata nel seguente modo.

Nel secondo capitolo vi è una panoramica sulle reti neurali classiche, sul deep learning e su alcuni modelli che si possono trovare all'interno del lavoro svolto.

Nel terzo capitolo vi sono illustrati alcuni dei dataset tipicamente utilizzati in task di computer vision e nello specifico il lavoro che ha portato alla realizzazione dei dataset utilizzati.

Nel quarto capitolo viene data una panoramica sui sistemi di deep learning che sono stati utilizzati in questo lavoro, con una spiegazione del loro funzionamento.

Nel quinto capitolo verrà illustrato nel dettaglio il lavoro svolto per la realizzazione del primo sistema.

[1] La mappa di disparità si riferisce alla differenza di pixel o al movimento tra una coppia di immagini stereo.

Nel sesto capitolo viene descritto il modello finale con le relative parti per i test e il training.

Infine, nell'ultimo capitolo vi è la presentazione dei risultati sperimentali delle due versioni del modello finale.

2 Le reti neurali

2.1 *Panoramica*

Una rete neurale artificiale è un modello matematico ispirato dalla versione, semplificata, di una rete neurale biologica. Una rete neurale artificiale può essere realizzata sia da programmi software che da hardware dedicato DSP[2].

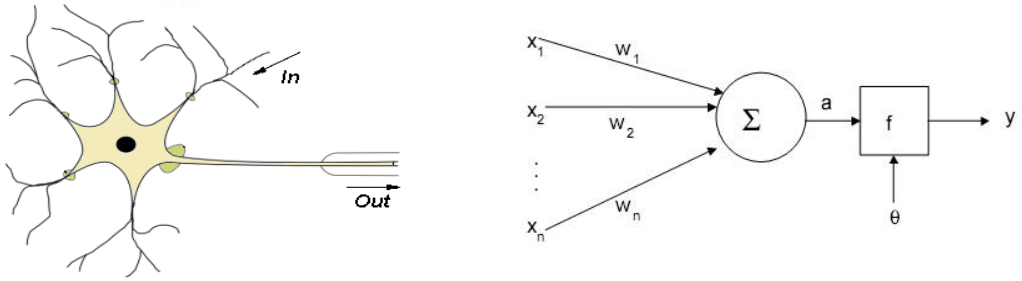


Figura 2: A sinistra il disegno di un neurone biologico caratterizzato da N ingressi e un'uscita. A destra l'equivalente neurone artificiale formato da N ingressi, una funzione di trasferimento e dall'uscita

Una rete neurale è costituita da un gruppo di interconnessioni e da un gruppo di nodi, detti neuroni, l'insieme di queste componenti forma un sistema adattivo che cambia la sua struttura in base alle informazioni che scorrono attraverso la rete durante la fase di apprendimento. Ogni neurone è connesso con molti altri, ed il collegamento può essere di tipo rafforzativo o inibitorio nei confronti dell'attivazione delle unità a cui è connesso. All'interno di un neurone è presente una funzione che combina tra loro i valori di tutti i suoi input ed una detta di attivazione, che rappresenta l'uscita del neurone. La forma generale della funzione presente in un neurone è:

$$y = f(\sum w_i \, x_i + b)$$

Dove w_i sono i pesi assegnati a ciascun input e b un termine di bias. L'insieme dei pesi e del bias rappresenta l'informazione che il neurone apprende in fase di training.

[2] Digital Signal processing, tecnica di elaborazione digitale dei segnali elettrici basata sull'uso di processori dedicati e con un grado elevato di specializzazione

La funzione *f* rappresenta la funzione di attivazione, solitamente consiste in una funzione di soglia e fa in modo che solo i segnali con valori compatibili possano propagarsi al neurone o neuroni successivi.

Tipicamente la funzione di attivazione è una funzione non lineare e solitamente si tratta di una funzione gradino o una sigmoide.

All'interno delle reti neurali si possono individuare principalmente tre parti o livelli:

- L'**input layer** è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete.
- L'**hidden layer**, è il livello che ha in carico il processo vero e proprio di elaborazione dei dati forniti dal primo livello. Questo è l'unico livello che può essere strutturato in più livelli.
- L'**output layer** raccoglie i risultati dei livelli precedenti e li adatta fornendo per ogni possibile classe un valore statistico. Questo valore ho come obiettivo quello di indicare a quale classe può appartenere l'input fornito alla rete.

Affinché la rete fornisca dei risultati corretti è necessario addestrarla, ossia fare in modo che apprenda come deve comportarsi nel momento in cui andrà a risolvere un problema, come ad esempio il riconoscimento di oggetti. Gli algoritmi di apprendimento sono divisi in 3 categorie: **supervisionato**, **non supervisionato** e **per rinforzo**. La scelta di quale usare dipende dal campo di applicazione per cui la rete viene progettata e dalla sua tipologia (*feed-forward o feedback*).

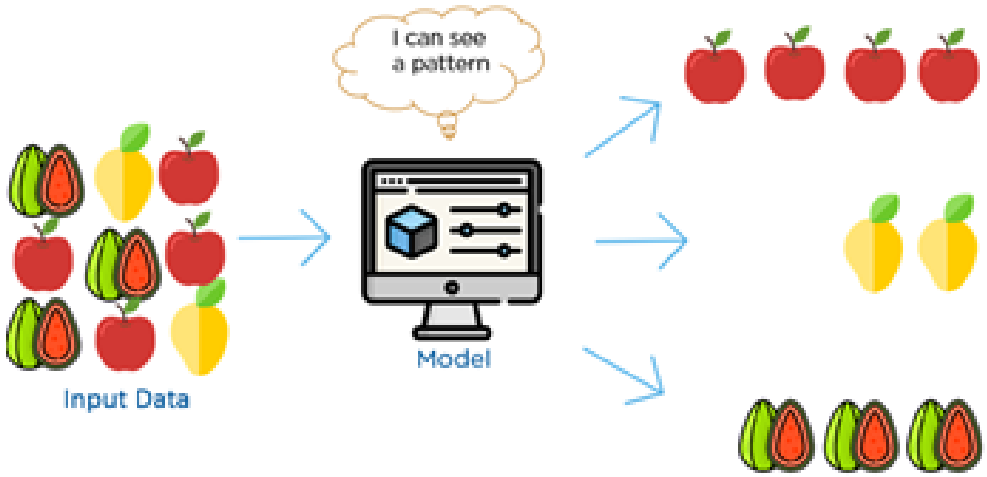
Apprendimento supervisionato

Con apprendimento supervisionato si intende quella metodologia di apprendimento automatico in cui alla macchina vengono forniti degli esempi composti da una coppia di dati, contenenti il dato originale e il risultato atteso. Il compito della macchina è quello di trovare la funzione o modello che colleghi il dato con il risultato atteso, in modo tale che, al presentarsi di un esempio sconosciuto possa ottenere il risultato corretto.

In questo tipo di apprendimento i dati devono essere precedentemente etichettati, ovvero devono essere assegnati a una categoria. L'apprendimento supervisionato è utilizzato principalmente per i problemi di classificazione[3].

Apprendimento non supervisionato

A differenza del precedente, non utilizza dati classificati e etichettati in precedenza. Alla macchina viene chiesto di individuare una regola che raggruppi i casi presentati secondo caratteristiche che ricava dai dati stessi.



In questo caso gli algoritmi cercano una relazione tra i dati per capire se e come essi siano collegati. Una delle applicazioni principali è il clustering, ovvero il raggruppamento dei dati in gruppi omogenei. Per questo motivo è spesso usato nelle discipline quali medicina o biologia e nell'ambito dei Big-Data, quand'è necessario correlare diversi dati ed estrarre informazioni non note.

Apprendimento parzialmente supervisionato

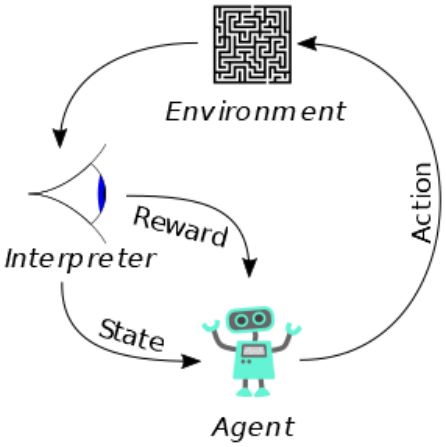
Si basa su dati misti, in cui una minima parte è già etichettata e una larghissima maggioranza è costituita da dati non etichettati. Questo approccio viene utilizzato per migliorare le previsioni fatte dalle macchine sui dati non etichettati. L'approccio è principalmente usato nei problemi di classificazione e di clustering.

[3] Ripartizione e distribuzione in raggruppamenti

Apprendimento per rinforzo

Questo tipo di apprendimento punta a realizzare sistemi in grado di apprendere ed adattarsi ai cambiamenti dell'ambiente in cui sono, attraverso la distribuzione di una ricompensa, che ne valuta le prestazioni.

All'interno dell'algoritmo esistono tre componenti:



- Sistema logico di **esecuzione (A)**, che sulla base dei dati in ingresso restituisce un risultato
- Sistema logico di **valutazione**, che assegna un premio o una penalità al *sistema logico di esecuzione*
- Sistema logico di ottimizzazione, che osserva il comportamento degli altri due sistemi e modifica il modello utilizzato da A per aumentare il premio e ridurre le penalità che assegna il *sistema di valutazione*.

Questo tipo di apprendimento è utilizzato in tutti quei campi in cui è essenziale che la macchina risponda ai cambiamenti dell'ambiente. Per questo è utilizzato in robotica, nella guida senza conducente.

2.2 Il deep learning

Il deep learning è una branca del machine learning basata sull'utilizzo di algoritmi il cui scopo è la modellazione di astrazioni di alto livello. Un'immagine, ad esempio, può essere rappresentata in diversi modi, con un vettore di valori di intensità per ogni pixel o in maniera più astratta come un insieme di bordi, regioni che presentano una particolare forma o caratteristica. Uno degli impieghi principali del deep learning consiste nella creazione di algoritmi di apprendimento specializzati nell'estrapolazione automatica di caratteristiche salienti (**feature**) in un set di dati, da utilizzare in seguito per l'addestramento di sistemi di machine learning. Il contributo apportato è altamente rilevante se si pensa che senza queste tecniche le suddette feature dovrebbero essere prodotte e valutate manualmente e preliminarmente all'addestramento. Il concetto alla base del deep learning è quello di sottoporre i dati di ingresso a numerosi livelli di elaborazione per ottenere delle feature. L'introduzione del deep learning è avvenuta attraverso la definizione delle reti neurali profonde. Il loro principio di funzionamento è lo stesso delle reti neurali classiche, con la differenza nell'elevato numero di livelli nascosti.

Tra le applicazioni di maggior successo troviamo la computer vision, con task che includono la classificazione, la regressione di immagini e l'object detection.

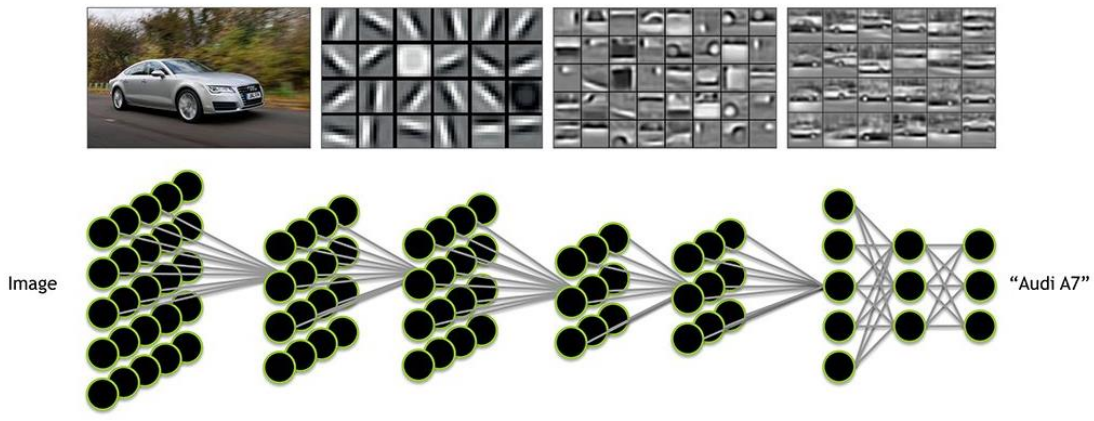
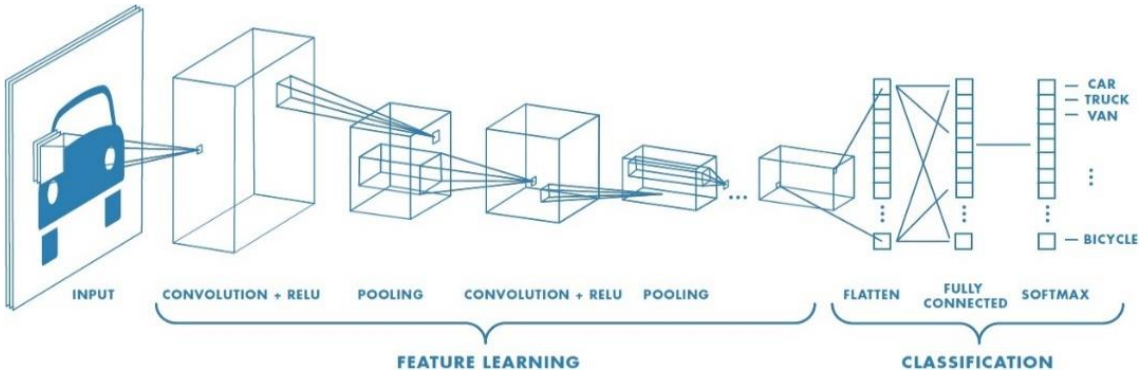


Figura 3: A ogni livello vengono estratte sempre più informazioni

2.3 Le reti neurali convoluzionali

La rete neurale convoluzionale o CNN è un tipo di rete feed-forward[4] in cui il pattern di connettività tra i neuroni è ispirato dall'organizzazione della corteccia visiva animale. Le CNN sono progettate per riconoscere pattern visivi direttamente in immagini rappresentate da pixel e richiedono una quantità di preprocessing molto limitata. Sono in grado di riconoscere pattern estremamente variabili, come ad esempio la scrittura a mano libera e le immagini rappresentanti il mondo reale. In questo tipo di rete è possibile individuare diversi livelli, ognuno dei quali con compiti specifici.



Tipicamente tra i livelli che compongono queste reti possiamo individuare: livelli di **convoluzione**, seguiti da livelli di **pooling** e livelli **completamente connessi**. In alcuni casi i livelli completamente connessi sono sostituiti da livelli di **upsampling**, in questo caso non si parla più di CNN ma di FCN (Fully Convolutional Networks).

Il livello di convoluzione e polling

Il **livello di convoluzione** è una parte fondamenta delle CNN, ed è solitamente presente in più livelli, questo perché maggiore è il loro numero e maggiore è la complessità delle caratteristiche che individuano. La convoluzione è un'operazione matematica che descrive una regola per combinare due funzioni oppure due pezzi distinti di un'informazione. Questi livelli vengono modificati e affinati in modo automatico, così

[4] Le connessioni tra le unità non formano cicli, le informazioni all'interno della rete viaggiano in una sola direzione.

che possano estrarre i dati più rilevanti con una precisione sempre maggiore. Ad esempio, per il riconoscimento di un oggetto in un'immagine potrebbe essere utile filtrare le informazioni riguardo alla forma dell'oggetto stesso, e saltare tutti gli altri attributi, in altri casi è più utile ricavare il più alto numero di dettagli e tralasciare altri tipi di informazioni. A poco a poco che le informazioni vengono filtrate da questi livelli, esse vengono anche semplificate tramite l'operazione di **pooling**: questa operazione permette di identificare se la caratteristica di studio è presente nel livello precedente e rende più grezza l'immagine, mantenendo la caratteristica utilizzata dal livello di convoluzione.

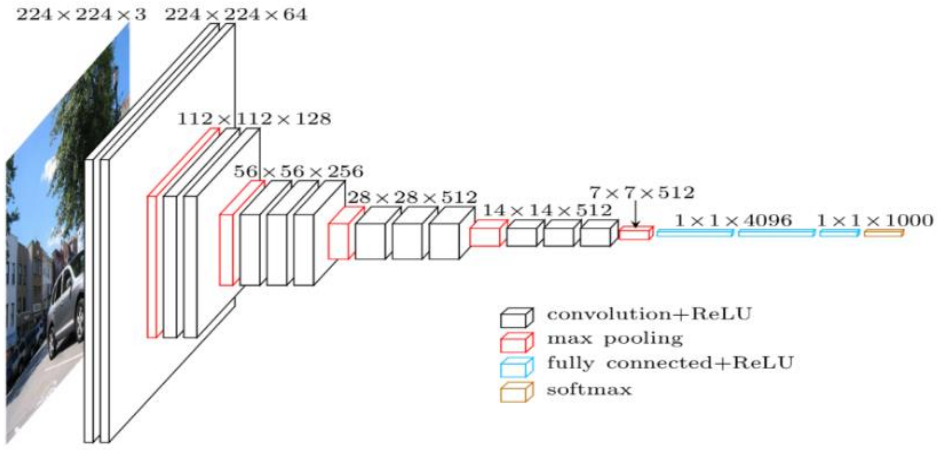


Figura 4: Struttura di una rete convoluzionale

I livelli FC permettono di connettere i neuroni del livello precedente al fine di stabilire le varie classi identificative visualizzate nei precedenti livelli secondo una determinata probabilità. Ogni neurone in uscita rappresenta una possibile risposta finale che il computer potrà fornire.

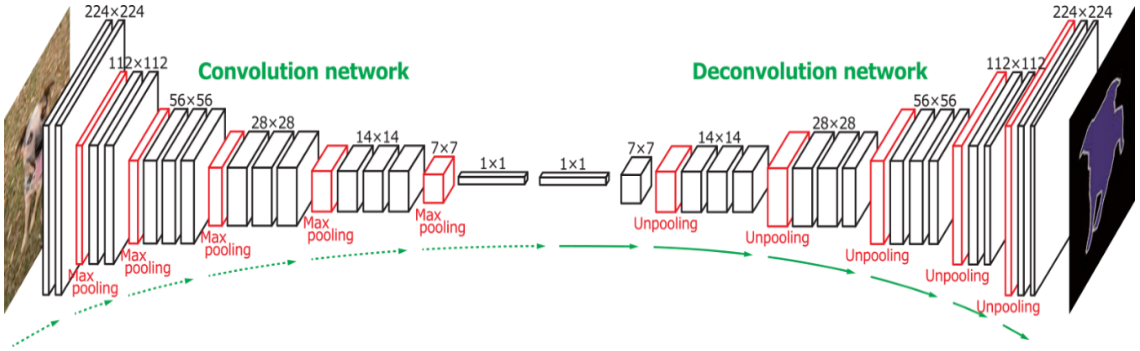


Figura 5: Struttura di una rete fully convolutional

Come detto precedentemente, non tutte le reti convoluzionali possiedono i livelli completamente connessi, alcune presentano livelli di upsampling. Questi livelli si trovano nella parte finale della rete, come i fully connected, tipicamente vengono utilizzati nei task di regressione, come la ricostruzione di immagini, la stima della profondità e la segmentazione. Tipicamente contengono dei livelli di **deconvoluzione** o convoluzione trasposta. Questo tipo di livello agisce in maniera inversa a quello di convoluzione e a quello di polling, così da ottenere la "crescita" delle dimensioni spaziale degli input. In uscita dalla rete si ottengono immagini di dimensioni comparabili a quelle di input.

3 Dataset e Framework

In questo capitolo vengono descritti i principali dataset impiegati nell'ambito della visione artificiale.

Il dataset KITTI[5] è una collezione di coppie di immagini generate a 1240x376 da due telecamere poste sul tettuccio di un'automobile distanti circa 54 centimetri l'una dall'altra. Le immagini sono state acquisite in varie condizioni meteo durante le ore del giorno nelle zone circostanti alla città di Karlsruhe in Germania, inoltre sono presenti anche le immagini di profondità, acquisite tramite LIDAR, che assegna un valore di disparità a circa il 30% dei pixel dell'immagine.
Il dataset raw contiene 48500 immagini e le corrispondenti groundtruth, acquisite con una frequenza di 10Hz.

Figura 6: Immagine sinistra (In alto) e destra del dataset KITTI

Oltre a KITTI esiste una variante sintetica, chiamata virtual kitti[6], questo dataset contiene 21260 immagini a colori e le corrispondenti mappe di profondità.

[5] http://www.cvlibs.net/datasets/kitti/
[6] http://www.europe.naverlabs.com/Research/Computer-Vision/Proxy-Virtual-Worlds

A differenza di KITTI, in virtual kitti non sono presenti immagini stereo e le immagini di profondità, essendo generate in modo virtuale, coprono il 100% dei pixel dell'immagine, in questo caso quindi si hanno delle mappe di disparità dense.

Altri dataset sintetici che solitamente vengono utilizzati nella visione artificiali sono quelli basati su immagini catturate dai videogiochi, come ad esempio GTA V.

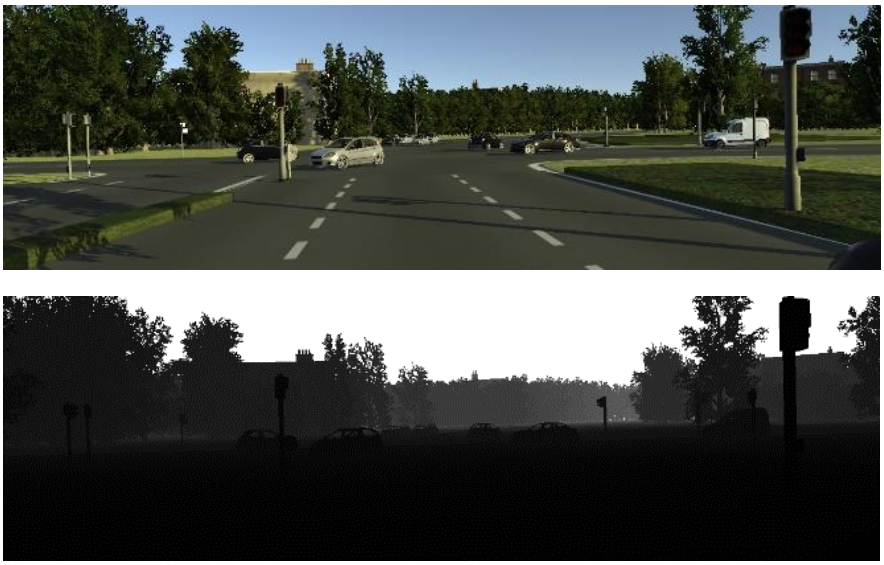


Figura 7: Immagine a colori e relativa groundtruth di virtual kitti

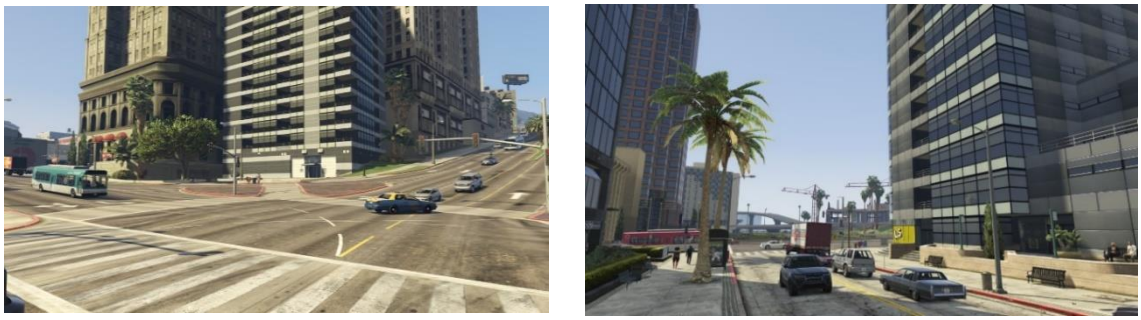


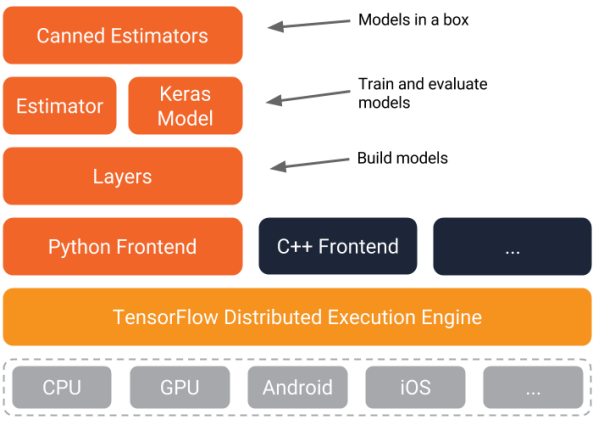
Figura 8: Immagini del dataset GTAV

3.1 I dataset utilizzati

Per questo lavoro sono stati utilizzati principalmente due dataset. Il primo dataset è stato realizzato utilizzando immagini di KITTI, in particolare quelle a colori della telecamera di sinistra e le mappe di profondità del dataset di virtual kitti (21260 immagini con relative mappe). Il secondo dataset è stato realizzato, sempre prendendo le immagini di KITTI e utilizzando le mappe generate dall'algoritmo di stereo visione SGM (48500 immagini a colori con relative mappe). La realizzazione di due dataset è stata imposta dalla scelta di utilizzare in un primo momento un modello, che chiameremo **DualGAN-Godard**, il quale in ingresso accetta coppie di immagini non necessariamente strettamente correlate e in un secondo momento il modello realizzato a partire da *pix2pix* in cui è necessario fornire coppie mappa-immagine strettamente correlate.

Per il secondo dataset l'utilizzo delle mappe realizzate tramite sgm è stato dettato dal fatto che quelle presenti in KITTI sono state realizzate tramite un sistema LIDAR. Questo tipo di sistema, come detto in precedenza, non ha una copertura totale dei pixel dell'immagine e il loro utilizzo può penalizzare la rete nel momento in cui in ingresso riceve mappe dense, ad esempio nel caso in cui si voglia usare la rete per generare delle immagini realistica a partire da mappe sintetiche. Inoltre, l'utilizzo di mappe dense permette di fornire alla rete mappe sintetiche, ad esempio realizzate tramite software 3D, ed ottenere le relative immagini a colori realistiche, con la possibilità di utilizzare le coppie ottenute come dataset per addestrare altre reti con mappe dettagliate di immagini reali, riuscendo così a migliorarle riducendo l'uso di dataset completamente sintetici.

3.2 Il framework TensorFlow



TensorFlow è una libreria open source per il calcolo numerico mediante l'utilizzo di grafi di flusso dei dati. Il framework è cross-platform ed è in grado di essere eseguito in praticamente tutti i tipi di hardware: gpu, cpu e anche tpu[7]. Il framework è in grado di astrarre i numerosi dispositivi supportati e fornisce un core ad alte prestazioni implementato in C++. Oltre a ciò si trovano i frontend Python e C++. L'API Layers fornisce un'interfaccia più semplice per i layer di uso comune nei modelli di deep learning. Inoltre, si aggiungono API di livello superiore, tra cui Keras e l'Estimator API, che facilita la formazione e la valutazione dei modelli distribuiti.

Graphs

Il grafo è una struttura dati che descrive completamente il calcolo che si desidera eseguire. Il suo utilizzo ha molti vantaggi:

- È portatile, poiché il grafo può essere eseguito immediatamente o salvato per essere utilizzato in un secondo momento e può essere eseguito su più piattaforme. Inoltre, può essere distribuito alla produzione senza dover dipendere da alcun codice che lo ha generato.
- È trasformabile e ottimizzabile, può essere trasformato per produrre una versione più ottimale per una data piattaforma. Inoltre, è possibile eseguire ottimizzazioni

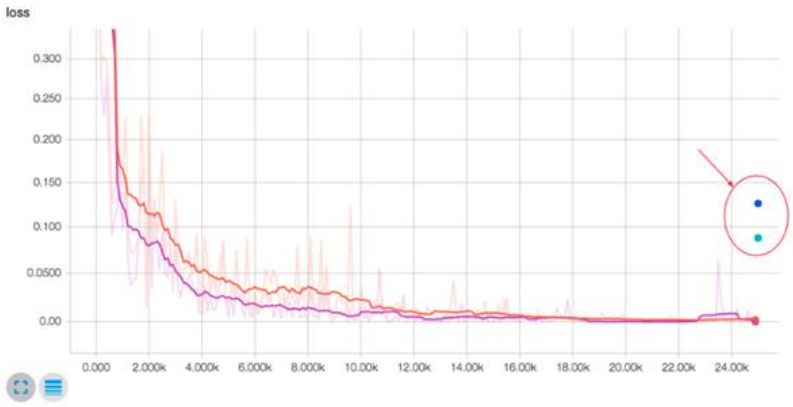
[7] TPU o Tensor Processing Unit è un acceleratore hardware per reti neurali e machine learning creato da Google

di memoria o calcolo e compromessi tra loro. Ciò è utile, ad esempio, nel supportare inferenze mobili più veloci dopo l'allenamento su macchine più grandi.

L'ecosistema TensorFlow include molti strumenti, tra questi troviamo **TensorBoard.**

TensorBoard è una suite di applicazioni Web per l'ispezione, la visualizzazione e la comprensione delle esecuzioni e dei grafi TensorFlow. È possibile utilizzare TensorBoard per visualizzare i grafici del modello e ingrandire i dettagli delle sottosezioni del grafico.

Attraverso TensorBoard è possibile tracciare metriche come la perdita e la precisione durante una sessione di allenamento; mostrare le visualizzazioni dell'istogramma di come un tensore cambia nel tempo. Inoltre, è possibile mostrare dati aggiuntivi come immagini, utilizzo totale della memoria e altro.



Session

TensorFlow usa la classe *tf.Session* per rappresentare la connessione tra il programma client e il runtime C++. L'oggetto *tf.Session* provvede a fornire l'accesso ai dispositivi della macchina locale, inoltre memorizza le informazioni del tf.Graph in modo da poter eseguire più volte lo stesso calcolo in modo efficiente.

Dato che un oggetto *tf.Session* possiede risorse fisiche (come GPU e connessioni di rete), viene in genere utilizzato come gestore di contesto che chiude automaticamente la sessione quando si esce dal blocco. È anche possibile creare una sessione senza utilizzare

un blocco, ma è necessario chiamare esplicitamente *tf.Session.Close* al termine dell'operazione per liberare le risorse.

```
# Create a default in-process session.
with tf.Session() as sess:
  # ...
```

Il metodo *tf.Session.run* è il meccanismo principale per eseguire un *tf.Operation* o valutare un *tf.Tensor*. é possiblie passare più oggetti a *tf.Session.run* e TensorFlow eseguirà le operazioni necessarie per calcolare il risultato.

```
x = tf.constant([[37.0, -23.0], [1.0, 4.0]])
w = tf.Variable(tf.random_uniform([2, 2]))
y = tf.matmul(x, w)
output = tf.nn.softmax(y)
init_op = w.initializer

with tf.Session() as sess:
  # Run the initializer on `w`.
  sess.run(init_op)

  # Evaluate `y` and `output`.
  y_val, output_val = sess.run([y, output])
```

4 I modelli utilizzati

4.1 Il modello Godard

Nel nostro task è fondamentale ottenere in uscita un'immagine a partire dalla mappa di disparità, per questo la scelta è subito ricaduta sulle FCN. Tra le reti fully convolutional vi sono molte implementazioni, ognuna delle quali differisce per complessità e metodo di training. Come ad esempio U-net, nella quale i livelli di convoluzione passano delle informazioni direttamente ai livelli di deconvoluzione, in questo modo i livelli successivi hanno una maggiore conoscenza delle forme e della posizione degli oggetti.

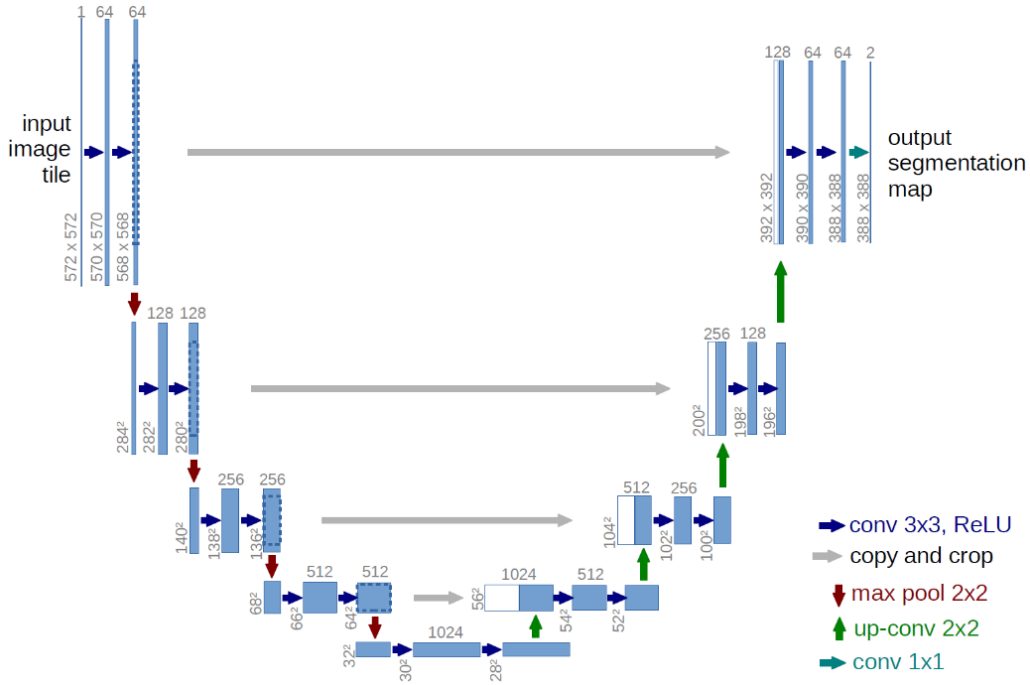


Figura 9: Architettura di U-net

Per riuscire a creare un modello che potesse andare bene per questo lavoro, ci si è basati su quelli attualmente utilizzati nell'ambito della generazione delle mappe a partire da immagini mono, essendo il nostro il problema inverso. In particolare, è stata studiata la rete **Godard.** [1]

La rete è strutturata secondo il modello VGG, presenta in ingresso 7 blocchi di convoluzione.
Ogni blocco di convoluzione è strutturato in due sotto-blocchi distinti di convoluzione, il primo sotto-blocco utilizza uno spostamento della finestra di convoluzione di un passo alla volta, mentre il secondo si sposta di due alla volta.

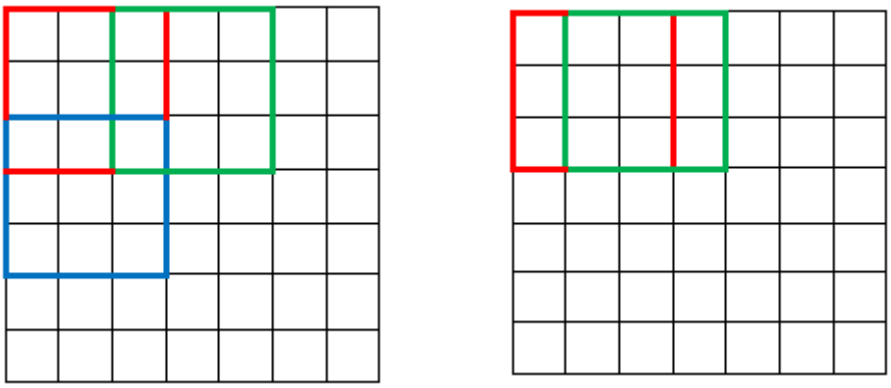


Figura 10: A sinistra finestra di convoluzione con stride a 2, a destra con stride a 1

Con la seguente formula è possibile calcolare le dimensioni dell'immagine in uscita dai livelli di convoluzione:

$$O = \frac{(W - K + 2P)}{S} + 1$$

Dove W è la dimensione in ingresso (altezza/larghezza), k la dimensione del filtro, P il padding e S lo stride. Nel caso di Godard in uscita dal primo blocca di convoluzione le immagini hanno la metà delle dimensioni originali e con 32 livelli, dal secondo blocco si ha immagini che sono un quarto di quella originale e 64 livelli. Al settimo blocco avremo un'immagine di dimensione un centoventottesimo di quella originale e con 512 livelli. Dopo i blocchi di convoluzione sono presenti i livelli di deconvoluzione, questi livelli permettono di ottenere immagini grandi come l'originale a partire da quella del livello precedente. Nel caso specifico questi livelli utilizzano la funzione trasposta alla convoluzione. Dopo ogni livello di deconvoluzione è presente un livello di convoluzione che prende in ingresso il livello precedente con il livello di convoluzione corrispondente, andando a creare le connessioni di skip. Queste connessioni permettono di passare delle informazioni tra i livelli di convoluzione e quelli di deconvoluzione, in modo che i livelli possano utilizzare questi dati per migliorare la loro uscita. Ad esempio, alcune delle

informazioni che potrebbero essere utilizzate sono il contorno e/o le forme dei vari oggetti della scena.

Il training in Godard

Solitamente le reti tipo questa vengono addestrate sfruttando delle immagini target, quindi con un metodo di addestramento *supervised*. L'utilizzo di questo tipo di addestramento non va sempre bene, soprattutto quando le immagini target ideali sono difficili da generare. Per questo in Godard è stato pensato un metodo *unsupervised*, in modo da slegare le mappe generate dalle mappe target, le quali potrebbero essere non dense, nel caso di uso di LIDAR oppure non definite, come nel caso in cui si utilizzino algoritmi per la loro creazione.

Figura 11: Approccio classico supervisionato

Per riuscire a rimuovere le disparità target è necessario portare i loss a confrontare le sole immagini a colori, l'idea alla base quindi è di sfruttare l'immagine destra per generare l'immagine a colori a partire dalla disparità di sinistra generata dalla rete e confrontarla con la sinistra in input alla rete.

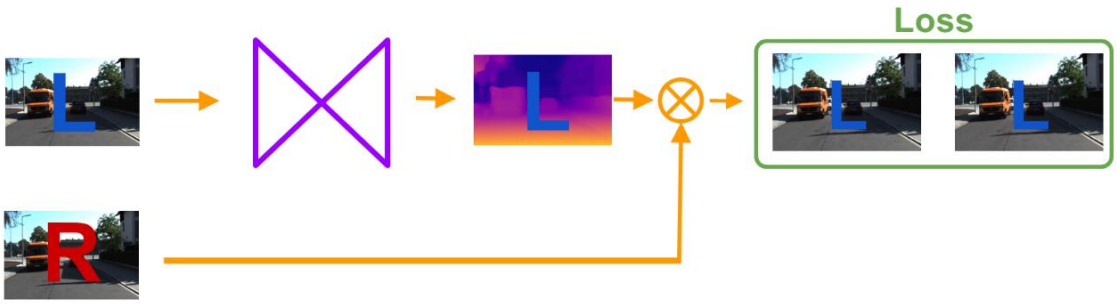


Figura 12: Addestramento non supervisionato in Godard

Per migliorare il modello è utile generare, oltre alla disparità sinistra, quella di destra, così da riproiettare le due immagini. Avendo le due immagini di disparità destra e sinistra è possibile calcolare un loss di consistenza tra le due e, utilizzando la disparità sinistra con

l'immagine di destra generare l'immagine a colori sinistra e confrontarla con quella in input. Lo stesso per l'immagine di sinistra e la disparità di destra.

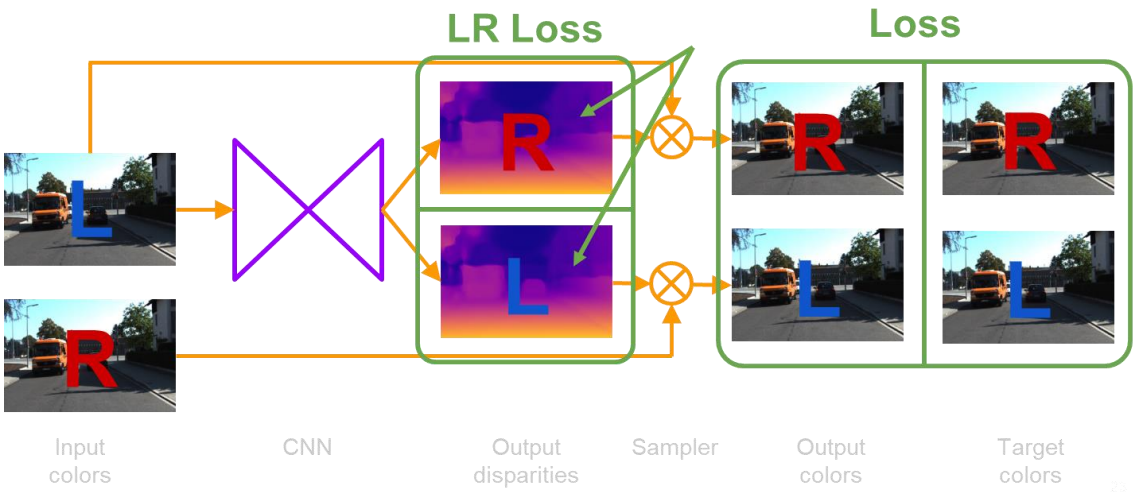


Figura 13: Utilizzo delle immagini desta e sinistra per il calcolo dei loss, senza l'utilizzo di immagini target

Il sistema deve essere quindi strutturato in modo che la rete riceva in ingresso la sola immagine sinistra e fornisca però in uscita le due disparità, l'immagine destra deve essere poi passata a un sampler, il quale tramite la disparità sinistra più l'immagine destra genera la corrispettiva immagine a colori di sinistra, con le immagini così ottenute è possibile poi calcolare il loss con le relative in input. Quindi, il loss totale sfrutterà diversi loss "secondari" per ottenere una stima della qualità delle immagini riproiettate, con il risultato di andare a valutare le disparità in modo "indiretto", tra i loss secondari vi sono L1 e SSIM.

Il loss L1 è la somma delle differenze, in valore assoluto, tra il valore target e quello stimato.

$$S = \sum_{i=1}^{n} |y_i - f_i|$$

Il solo uso di questo loss non è in grado di restituire una stima precisa della somiglianza delle due immagini, con l'introduzione anche di SSIM si può avere una stima migliore.

SSIM o *Structural Similarity* è utilizzato per misurare la somiglianza tra due immagini, la stima della qualità dell'immagine è basata su un'immagine iniziale non distorta.

$$S(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Dove μ_x è il valore medio dei x_i, σ_x^2 la varianza di y, σ_{xy} la covarianza di x *e* y.

Il loss totale, quindi può essere calcolato sfruttando la somma pesata dei due loss precedenti e aggiungendo due ulteriori fattori è possibile migliorare ulteriormente il loss finale. Con il disparity smoothness si "incoraggiano" le disparità ad essere localmente "smooth" con una penalità L1 sui gradienti di disparità ∂d.

$$C_{ds}^l = \frac{1}{N}\sum_{i,j} \left|\partial_x \partial_{ij}^l\right| e^{-\left\|\partial_x I_{ij}^l\right\|} + \left|\partial_y \partial_{ij}^l\right| e^{-\left\|\partial_y I_{ij}^l\right\|}$$

L'altro fattore è il left-right Disparity Consistency, per avere coerenza tra le disparità, avendo in input solo un'immagine, si introduce una penalità di tipo L1 left-right disparity consistency. Questo costo cerca di fare in modo che la mappa di sinistra sia uguale alla proiezione della mappa di destra.

$$C_{lr}^l = \frac{1}{N}\sum_{i,j} \left|d_{ij}^l - d_{ij+d_{ij}^l}^r\right|$$

Come per gli altri termini, questo costo è specchiato per la disparità di destra ed e valutato su tutte le scale in uscita dalla rete.

4.2 Il modello DualGAN

Il modello **DualGAN** [2], utilizzato principalmente per passare da un dominio A ad un dominio B, come ad esempio trasformare la foto di un volto nel suo disegno o per trasformare una foto scattata di giorno nella corrispondente scattata di notte.

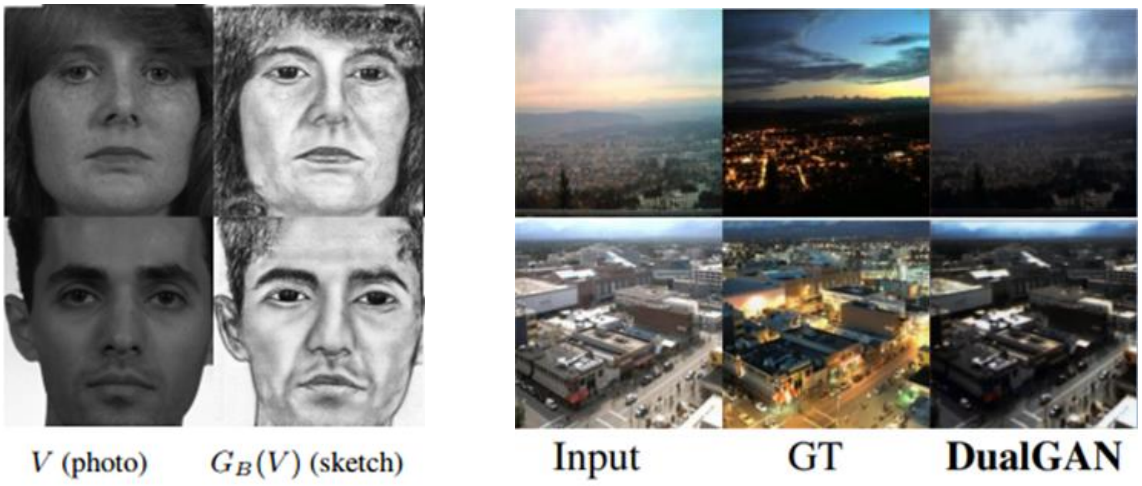


Figura 14: A sinistra il volto viene trasformato nel suo disegno, mentre a destra da foto scattata di giorno a foto notturna (la ground truth, al centro tra input e output di DualGAN)

In questo caso DualGAN non utilizza una sola rete, ma due diverse. È quindi possibile dividere le reti in due tipologie:

- Generatrice
- Discriminatrice

La generatrice, come dice il nome, è quel tipo di rete che ha il compito, a partire da un'immagine del dominio A, di generare l'immagine corrispondente del dominio B, cercando di "ingannare" la rete discriminatrice. Questa rete è sempre del tipo FCN, ma strutturata in modo diverso rispetto VGG, infatti è possibile trovare 8 livelli di convoluzione seguiti da livelli di **batch normalization**.

Il livello di *batch normalization* riduce la quantità di ciò che i valori delle unità nascoste spostano (spostamento di covarianza) e permette di avere un learning rate più alto. Per spiegare il cambiamento di covarianza, basta pensare a una rete profonda che riconosca la presenza o meno di gatti in una foto. Assumiamo che la rete sia stata addestrata solo con foto di gatti neri, se a questa rete venissero passate delle foto di gatti colorati, la

previsione non sarebbe corretta, nonostante il set di addestramento e il set di predizione contengano entrambi dei gatti, ma leggermente differenti. In altre parole, se l'algoritmo apprende alcune mappature da X a Y e se la distribuzione di X cambia, allora potrebbe essere necessario riqualificare l'algoritmo di apprendimento cercando di allineare la distribuzione di X con la distribuzione di Y.

Dopo gli otto livelli si trovano gli 8 livelli di upsampling, realizzati tramite la convoluzione trasposta. Anche in questo caso sono stati realizzati i livelli di skip tra quelli di convoluzione e i relativi di deconvoluzione.

La funzione della rete discriminatrice, è quella di cercare di distinguere le immagini false, create dalla generatrice, da quelle reali presenti nel training set e di fornire un'immagine "punteggio" per indicare quali parti dell'immagine secondo lei sono reali e quali no.

Il training in DualGAN

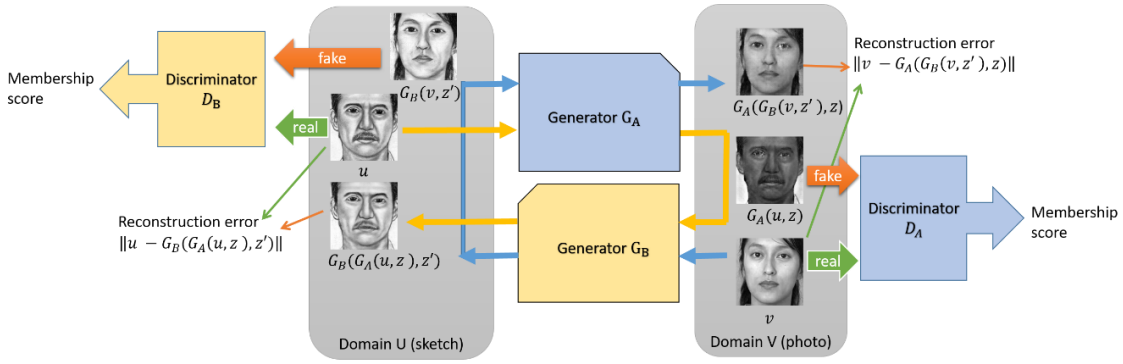


Figura 15: Struttura completa di DualGAN

La struttura completa, riportata in *Figura 15*, permette di avere una visione d'insieme del modello. Partendo dal dominio *u* la rete generatrice G_A riceve in ingresso il disegno del volto e genera la foto corrispondente, la rete discriminatrice D_A prende in ingresso l'immagine generata e un'immagine dal dominio *v* (foto di volti) e tenta di distinguere la reale da quella generata, fornendo uno score. L'immagine generata a questo punto passa nella generatrice G_B, che ricrea il disegno iniziale, con questo appena generato e quello di partenza viene calcolato l'errore di ricostruzione tramite il loss L1. L'ottimizzatore deve sistemare i pesi della rete affinché questo loss diventi zero. Partendo dal dominio *v* (a destra) l'immagine del volto passa per la G_B che genera il disegno e viene passato alla

discriminatrice D_B la quale questa volta deve tentare di riconoscere il disegno originale da quello generato. Il disegno generato da G_B viene anche passato a G_A che restituisce l'immagine del volto, con l'immagine di G_A e con quella iniziale viene calcolato il secondo errore di ricostruzione, anche in questo caso l'ottimizzatore deve tornare a sistemare i pesi della rete G per portare l'errore di ricostruzione a zero. Le due reti discriminatrici migliorano sempre, di conseguenza le generatrici devono migliorare a loro volta per continuare a ingannare le discriminatrici.

L'uso di questi due "loop" permette di eliminare la necessità di avere coppie di immagini strettamente correlate tra loro, infatti all'ingresso delle due discriminatrici non vengono fornite coppie correlate di immagini target, ma immagini random appartenenti ai due domini. Questo aspetto è molto importante, infatti non in tutte le situazioni è possibile avere o generare coppie di immagini strettamente correlate tra loro.

4.3 Il modello PIX2PIX

Il terzo e ultimo modello utilizzato in questo lavoro è pix2pix (Phillip Isola s.d.). Esso è utilizzato principalmente in quei task in cui vi è necessità di eseguire un passaggio tra un dominio A e uno B, ad esempio disegni e foto, oppure mappa e mappa aerea.

Figura 16: Esempio di trasformazioni utilizzando pix2pix

Questo modello è stato implementato sfruttando le *cGAN* (conditional GAN). Le *GAN* sono dei modelli generativi che imparano come mappare un vettore di rumore *z* con l'immagine in uscita y, $G: z \rightarrow y$. Le cGAN invece imparano a mappare dalle immagini osservate e dai vettori di rumore a *y*, $G: \{x, z\} \rightarrow y$. Il generatore *G* viene addestrato a produrre immagini che non possono essere distinte da quelle "reali" da un discriminatore *D*, il cui scopo è quello di riconoscere le immagini generate da *G*. L'obiettivo quindi di una cGAN può essere espresso come

$$L_{cGAN} = E_{x,y}[\log(D(x,y))] + E_{x,z}\left[\log\left(1 - D(G(x,z))\right)\right]$$

dove *G* tenta di minimizzare in contrasto con *D* che tenta di massimizzare, ad esempio: $\arg min_G \, max_D \, L_{cGAN}(G, D)$.

Senza l'utilizzo di *z* la rete è comunque in grado di imparare il mapping tra *x* e y, soltanto che l'uscita prodotta sarà deterministica, con il conseguente fallimento nell'uguagliare una qualsiasi distribuzione diversa dalla funzione di delta.

In questo caso il vettore rumore *z* è stato realizzato attraverso il dropout applicato in molti dei livelli della rete, sia in fase di training che in fase di test.

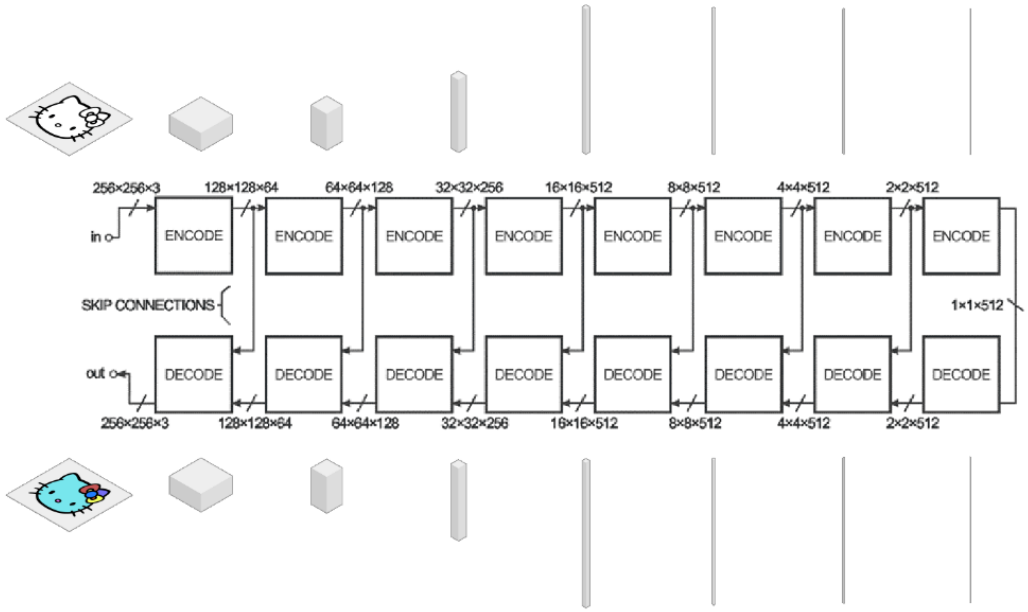


Figura 17: Struttura della rete generatrice

La rete generatrice prende in ingresso l'immagine di input e tenta di ridurla, con una serie di "codificatori", in una sua rappresentazione più piccola. L'idea è che comprimendola si ottiene una rappresentazione più utile dei dati in ingresso.

I livelli di decodifica fanno il lavoro opposto. Partendo dalla rappresentazione compressa dell'ingresso, la espandono fino ad ottenere un'immagine in uscita grande quanto quella in ingresso.

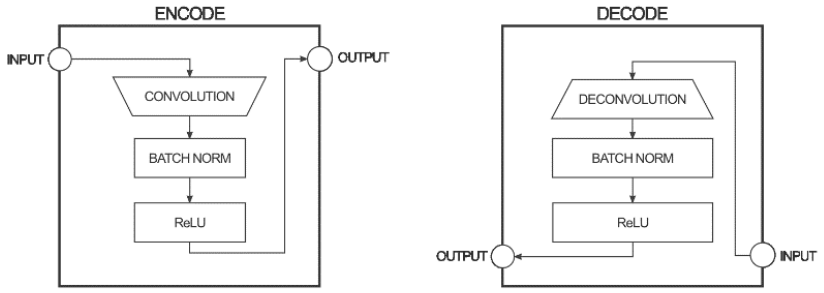


Figura 18: Blocchi di codifica e decodifica

Ogni livello di codifica è formato da diversi livelli: un livello di convoluzione, seguito da batch normalization e dal livello di attivazione ReLu. Il blocco di decodifica, molto simile a quello di codifica, presenta un livello di deconvoluzione, seguito da uno di batch normalization e dal livello di attivazione ReLu.

Per migliorare le performance sono stati aggiunti i collegamenti di skip tra i vari encoder e decoder. Le connessioni di skip permettono alla rete di bypassare la parte di codifica/decodifica.

Il discriminatore ha il compito di prendere due immagini, una di input e una seconda detta unknown, quest'ultima potrà essere sia quella creata dal generatore, sia quella target e decidere se la seconda immagine è stata generata dalla rete generatrice oppure no.

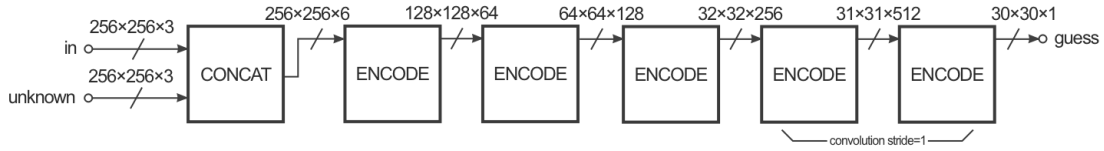


Figura 19: Struttura della rete discriminatrice

La struttura assomiglia alla sezione di codifica del generatore, in questo caso però l'uscita è un'immagine 30x30 in cui ogni pixel rappresenta quanto sia reale la sezione corrispondente dell'immagine sconosciuta. Ogni pixel di questa immagine corrisponde a una patch di 70x70 dell'immagine di input. Questa architettura è chiamata *PatchGAN*. Nei problemi di generazione di immagini i loss di tipo L2 e L1 producono risultati sfuocati. Sebbene questi loss incoraggiano la nitidezza ad alta frequenza, ciò nonostante in molti casi non riescono a catturare accuratamente le basse frequenze. Non è necessario creare un framework completamente nuovo per rafforzare la correttezza alle basse frequenze, è sufficiente avere il loss L1. Per modellare le alte frequenze è sufficiente restringere l'attenzione sulla struttura locale dei patch dell'immagine.

Il training in pix2pix

In questo modello di rete il training è costituito da due step:

- Addestramento del discriminatore
- Addestramento del generatore

Per addestrare il discriminatore, per prima cosa il generatore deve generare un'immagine in uscita. Il discriminatore osserva alle coppie input/target e input/output e produce un'ipotesi su quanto sono realistiche. I pesi del discriminatore vengono quindi aggiustati in base alla classificazione dell'errore tra le coppie input/output e input/target.

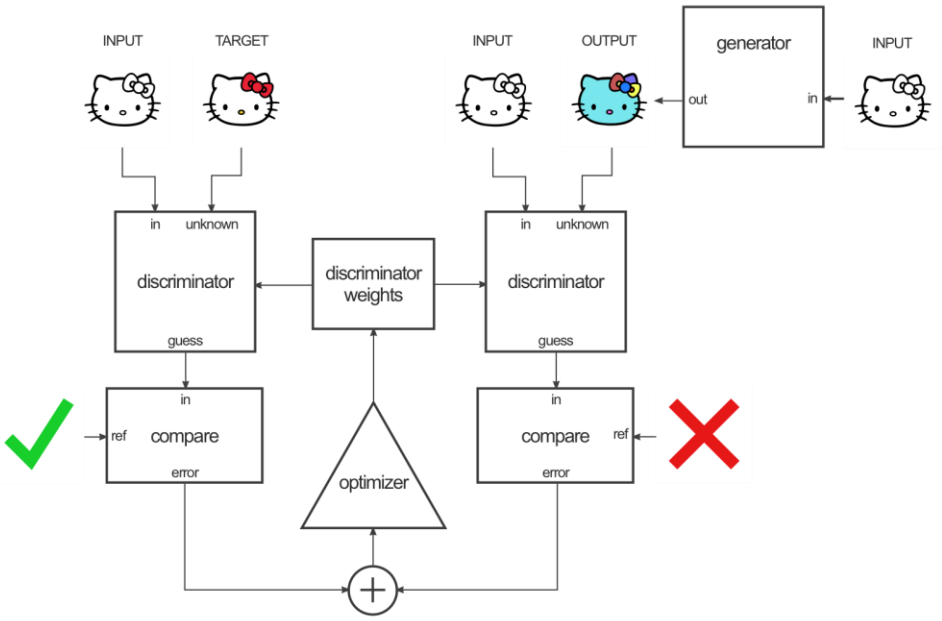


Figura 20: Addestramento della rete discriminatrice

I pesi del generatore vengono poi aggiustati in base all'uscita del discriminatore, anche in questo caso sulla base della differenza tra l'immagine in uscita e quella target. Lo stratagemma in questo caso è che quando si addestra il generatore sull'immagine in uscita del discriminatore, si sta effettivamente calcolando il gradiente attraverso il discriminatore, il che significa si sta addestrando il generatore a battere il discriminatore, il quale continua a migliorarsi. L'idea è che quando il discriminatore migliora anche il generatore lo fa, se il discriminatore è bravo nel suo compito e il generatore è in grado di

apprendere la corretta funzione di mappatura attraverso la discesa del gradiente, si dovrebbero ottenere degli output che potrebbero ingannare anche un essere umano.

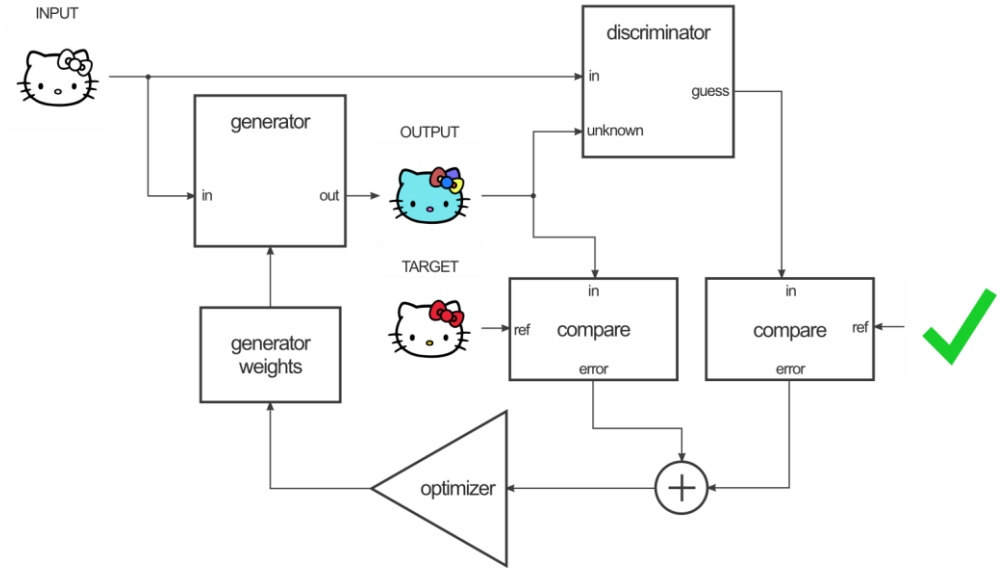


Figura 21: Addestramento della rete generatrice

5 La prima versione

5.1 DualGAN-Godard

È possibile suddividere la realizzazione di questo modello in varie fasi:

- Test e verifica dei due modelli utilizzati
- Individuazione di eventuali problemi
- Risoluzione dei problemi riscontrati
- Risultati finali del modello

Durante la prima fase del lavoro si è testato il funzionamento del modello Godard nel suo task originale, cioè da immagine monoculare a mappa di disparità. Per cercare di riprodurre i risultati del paper si è fatto partire un training da zero ed eseguito per qualche epoca, in modo da essere sicuri che il modello non presentasse problemi. Successivamente si è passati al modello DualGAN, in questo caso la struttura della rete non era delle migliori per il task, per cui si è pensato di unire le caratteristiche dei due modelli al fine di realizzarne una terza che possa ottenere buoni risultati. Questa nuova architettura deve essere in grado di gestire il compito, mantenendo però un livello di complessità allineato con i due modelli precedenti. La complessità del modello è un aspetto da tenere in considerazione perché più il modello è complesso e più facilmente può presentare problemi di apprendimento, inoltre il tempo di training risulterà essere più lungo, un'altra conseguenza riguarda la fase di inferenza in cui il sistema risulterà essere lento.

Dalla prima architettura si è preso il modello della rete e i metodi di calcolo dei loss, i quali per il problema inverso al nostro davano ottimi risultati. Dalla seconda, invece, è stato utilizzato il metodo di training e la struttura di interazione tra le reti, in questo caso non si necessitano di coppie (*imgA*-*imgB*) strettamente correlate tra loro. I loss del primo modello sono strutturati nel seguente modo:

Il loss SSIM:

$$S(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Con il contributo disparity smoothness si "incoraggiano" le disparità ad essere localmente "smooth" con una penalità L1 sui gradienti di disparità ∂d.

$$C_{ds}^l = \frac{1}{N}\sum_{i,j} |\partial_x \partial_{ij}^l| e^{-\|\partial_x I_{ij}^l\|} + |\partial_y \partial_{ij}^l| e^{-\|\partial_y I_{ij}^l\|}$$

Questo ultimo costo cerca di fare in modo che la mappa di sinistra sia uguale alla proiezione della mappa di destra, cosicché da ottenere un uscita coerente.

$$C_{lr}^l = \frac{1}{N}\sum_{i,j} \left| d_{ij}^l - d_{ij+d_{ij}^l}^r \right|$$

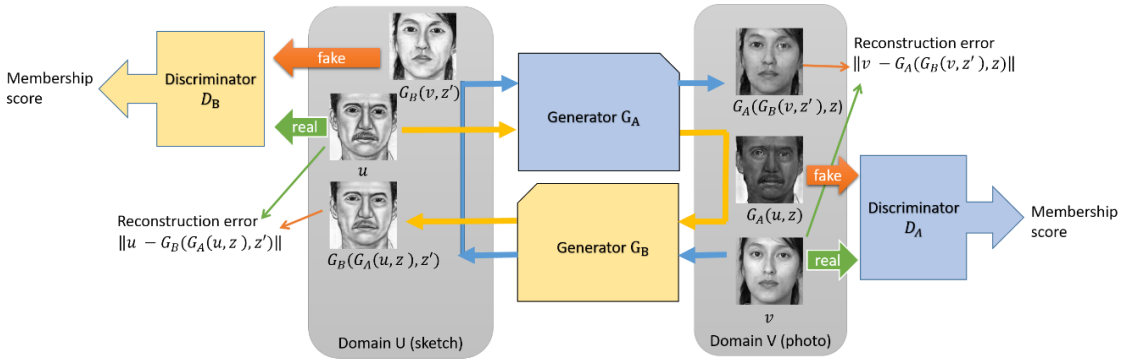


Figura 22: DualGAN

Dalla figura è possibile osservare che la parte più consistente appartiene al modello dualGAN, di conseguenza è più semplice modificare quest'ultimo inserendo i loss e la rete dell'altro.

La prima parte che è stata sostituita è la struttura della rete generatrice, la quale è stata rimpiazzata dal modello presente in Godard (secondo modello). In questa fase la sostituzione non ha presentato grossi problemi, questo perché il codice è stato strutturato in modo da poter apportare modifiche alle reti. Per verificare che la nuova rete sia stata inserita correttamente si è deciso di avviare il training e constatare che le immagini in uscita migliorassero con l'avanzare delle iterazioni.

Nella fase successiva si è reso necessario sostituire il "reconstruction loss". Quello originale è di tipo L1, $\|u - G_B(G_A(u,z),z')\|$ cioè la differenza tra l'immagine reale del dominio u e l'immagine generata da G_B a partire dall'uscita della prima generatrice G_A, questa formulazione però non è in grado di fornire una stima precisa di quanto siano simili le due immagini, in particolare l'uso di questa formulazione porta alla generazione di immagini "sfuocate", con l'utilizzo combinato di L1 e SSIM è possibile ottenere dei valori migliori.

A differenza del reconstruction loss del primo modello che utilizza solo l'immagine in uscita dalla generatrice, il loss del secondo sfrutta l'immagine su 4 differenti scale, quindi si rende necessario che la rete fornisca l'immagine in uscita anche nei livelli intermedi di upsampling. Per poter confrontare le immagini in uscita dalla rete con quelle di riferimento, bisogna anche introdurre la scalatura a piramide delle relative immagini di training. Con il reconstruction loss aggiornato in questo modo è possibile rendere il training completamente non supervisionato. Con l'introduzione di questa tipologia di addestramento si è reso però necessario modificare ulteriormente il modello eliminando il caricamento delle mappe dal training set e fornendo al loro posto le mappe generate da G_A, facendole così passare per immagini del secondo dominio.

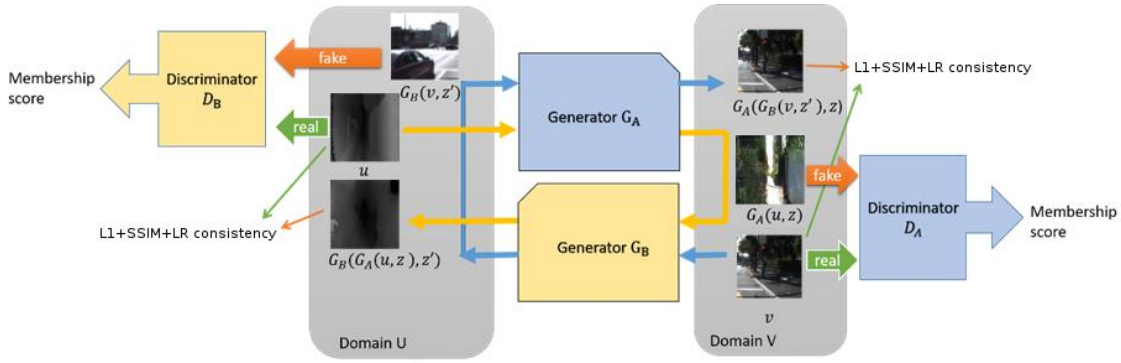


Figura 23: Versione finale della rete

L'utilizzo di questo accorgimento non crea problemi alla struttura, inizialmente le mappe della rete G_A non saranno ottime, di conseguenza la rete G_B non avrà dei buoni input, poi però col passare delle iterazioni G_A migliora con il risultato che anche la seconda generatrice si perfeziona.

Durante questa fase di training sono emersi alcuni problemi del modello, uno di questi è la difficoltà nel generare le mappe di disparità. Per poter capire dove fosse il problema è

stato rivisto il modello e le sue parti critiche, in particolare calcolo dei loss e passaggio delle mappe dalla prima alla seconda generatrice.

Prima però di effettuare cambiamenti alla struttura e/o al dataset si è controllato che non ci fossero errori nel codice.

Avendo constato che non ci sono problemi di implementazione si è deciso di provare ad utilizzare un dataset sintetico (virtual kitti) per verificare che il problema non fosse relativo all'utilizzo di uno complesso come kitti. Con il nuovo dataset la rete era in grado di generare correttamente le immagini a colori, come nel caso di kitti, per le mappe invece ha introdotto un'altra problematica, senza risolvere la precedente. All'inizio del training la rete riesce a generare delle mappe, anche se non propriamente corrette, poi però verso la metà dell'addestramento le mappe iniziano a diventare completamente bianche e al contempo i loss continuano a calare fino ad arrivare quasi a zero, la causa di questo è probabilmente da ricercare nel training non supervisionato.

Figura 24: Le immagini relative al ciclo disparità-colori-disparita (A sinistra) e al ciclo colori-disparità-colori (A destra)

Il metodo implementato utilizza le disparità generate dalla rete per calcolare la relativa immagine tramite riproiezione, utilizzando la disparità sinistra con l'immagine input a colori destra è possibile ottenere l'immagine a colori di sinistra.

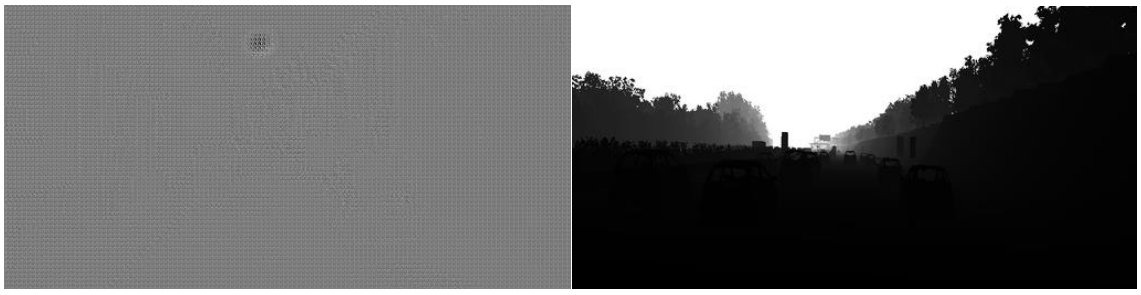


Figura 25: A sinistra l'uscita di GA con evidenti problemi nel ricostruire la disparità, a destra l'immagine groundtruth

Se al calcolo di riproiezione, al posto delle disparità forniamo delle immagini completamente bianche, esso in uscita ci genera l'immagine a colori in ingresso.

In questo caso l'ottimizzatore per raggiungere il suo obiettivo, cioè quello di minimizzare i loss, modifica i pesi della rete in modo da generare delle immagini completamente bianche che poi attraverso la riproiezione portano ad ottenere le immagini a colori di input e quindi i loss a valori vicini allo zero.

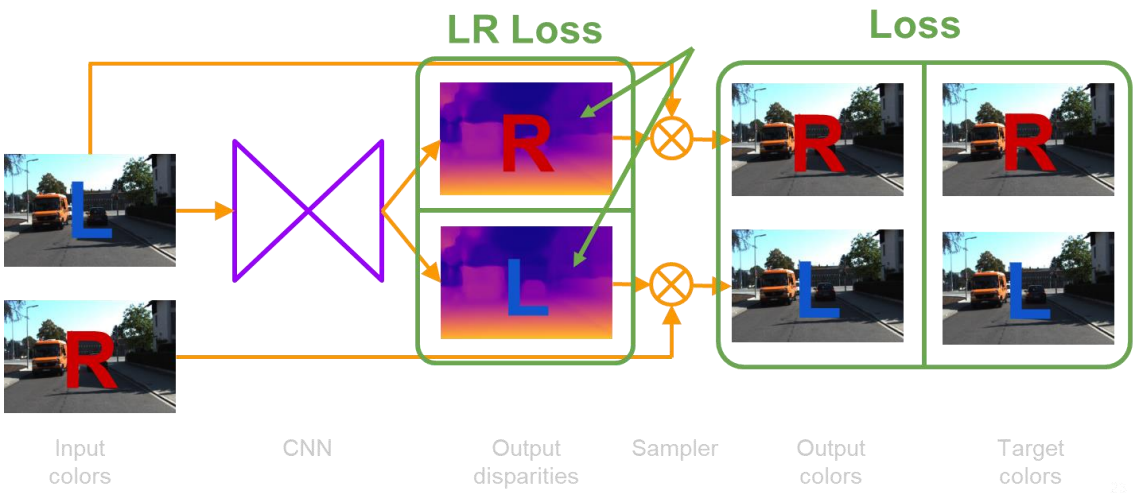


Figura 26: Riproiezione delle disparità

Si è deciso quindi di fare un passo indietro e di provare a fare una versione supervisionata della stessa rete così da evitare, almeno inizialmente, questo tipo di problematica.

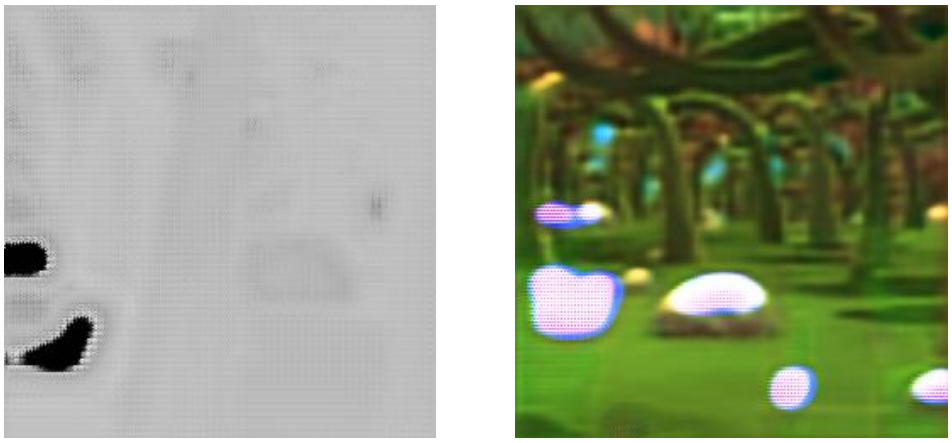


Figura 27: Risultati iniziali della versione supervisionata

Per realizzare la versione supervisionata si è dovuto cambiare leggermente il loss, in questo caso si è mantenuto il calcolo *L1* e *SSIM*, eliminando però il calcolo della riproiezione e introducendo le immagini target per le disparità. Nel dataset *kitti* non sono presenti le mappe di disparità dense, ma quelle realizzate tramite *LIDAR*, per evitare di doverle generare si è deciso di utilizzare il dataset sintetico *virtual kitti*, il quale per ogni scena possiede le mappe dense e molto più dettagliate di quelle ottenibili tramite algoritmi tipo SGM. L'obiettivo di questa nuova versione è quello di verificare se eliminando il problema delle disparità bianche, la rete è in grado di generarle in modo corretto.

Come è possibile osservare dalla *Figura 27*, la rete è in grado di generare correttamente le immagini a colori, come nel caso precedente, ma le mappe risultano essere ancora errate. Esse infatti sono più simili alle immagini in input in toni di grigio, che alle disparità target. Una possibile spiegazione è che il compito affidato alla rete sia troppo complesso per la sua architettura, per verificare se il problema sta nella complessità del task si è passati a semplificare il lavoro della rete. Come detto inizialmente il compito di generare le mappe a partire dall'immagine monoculare è molto complesso, per semplificare è sufficiente fornire più informazioni alla rete G_A attraverso l'utilizzo di immagini stereoscopiche, in questo modo dovrebbe essere in grado di ricostruire le profondità della scena con meno problemi. Purtroppo, il dataset di virtual kitti non fornisce le immagini stereo, per cui si è dovuto cambiare ulteriormente dataset e passare a *monkaa*. Questo dataset è sempre di tipo sintetico, ma, oltre alle mappe, fornisce anche le immagini stereo relative.

Figura 28: Immagine sinistra e relativa mappa del dataset monkaa

Per escludere problemi relativi alla profondità della rete si è deciso anche di incrementarne la complessità aggiungendo qualche ulteriore livello. L'aggiunta di un paio

di livelli alle generatrici dovrebbe permettere alle reti di risolvere problemi più complessi, entro certi limiti. Nonostante l'utilizzo di immagini stereo e l'incremento di profondità la rete continua a presentare problemi alle mappe di disparità, mentre le immagini a colori continuano ad essere corrette come nelle prove precedenti.

Da questi test è stato possibile quindi escludere problemi relativi alla profondità della rete e in parte anche quelli che riguardavano il dataset, di fatto il modello presenta problemi in tutte e due le versioni, sia in quella mono che in quella stereo; i risultati ottenuti nelle due versioni non sono molto diversi. Se il problema non riguarda principalmente la profondità della rete e il dataset, è possibile quindi ritornare alla versione stereo iniziale, senza la presenza dei nuovi livelli.

A questo punto il problema potrebbe risiedere nei vari pezzi assemblati, per riuscire a capire cos'è che non va si è deciso di testare i pezzi del modello in modo separato. Per prima cosa si è testato il modello Godard originale nel suo task inverso, da disparità a immagine monoculare. Testandolo in questa "modalità" è possibile capire se la rete è in grado di apprendere il compito inverso, escludendo così che i problemi siano relativi alla rete. Dalla prova si è quindi potuto constatare che la rete è in grado di assolvere al proprio compito senza apparenti difficoltà, cioè esclude che il problema riguardi la struttura della rete e permette di spostare la ricerca delle criticità nella parte di DualGAN.

Per verificare quale sia il problema si è deciso di fare alcune prove mantenendo la versione supervisionata, cercando però di caricare i pesi finali di Godard nella rete G_A, così facendo l'architettura non deve più imparare a generare le mappe, ma solamente a generare le immagini a colori. Inoltre, con la G_A già in grado di fornire mappe corrette la generatrice G_B dovrebbe impiegare meno tempo ad apprendere il suo task. Inizialmente per realizzare i loop tra le generatrici sono state creare quattro reti, le due principali G_A e G_B, che relativamente prendono in ingresso le immagini monoculari a colori e le mappe, e poi altre due reti che formano i loop, quest'ultime prendono in ingresso le uscite delle precedenti condividendone i pesi. Nel momento in cui si è andati a modificare il codice per aggiungere il caricamento della rete G_A sono sorti i primi problemi, la struttura del codice non è stata pensata per caricare solamente delle parti dell'architettura, ma per caricare l'intero modello, per poter caricare la rete G_A si è dovuto modificare anche il

metodo con cui venivano create G_A e G_B. La creazione delle due reti veniva fatto in un passaggio unico, questo però non permette di effettuare il caricamento della singola rete, ma solo di entrambe. Per riuscire a caricare solo la G_A si è dovuto separare in tre fasi la creazione delle generatrici, nella prima fase viene creata la rete G_A, come in precedenza; nella seconda avviene il caricamento dei pesi di G_A tramite l'utilizzo del checkpoint finale; nell'ultima fase vi è la creazione della G_B. Risolto il primo ostacolo si è potuto iniziare a testare nuovamente il sistema.

Anche in questo caso il training è avvenuto con il precedente dataset; il caricamento dei pesi della rete ha però introdotto nuovi problemi, uno tra questi riguarda sempre la mappa di disparità generata dalla prima rete. Se nel caso precedente le immagini a metà training diventavano completamente bianche, in questo caso le mappe non vengono neanche abbozzate, risultano essere completamente nere. Uno dei fattori che può provocare un "malfunzionamento" della rete caricata è il metodo con cui le vengono passate le immagini.

Nel modello DualGAN le immagini vengono ritagliate alla dimensione corretta e passate alla rete, mentre nel modello da cui sono stati presi i pesi le immagini venivano ridimensionate, senza alcun ritaglio e passate.

Entrambe le modalità di pre-processing sono valide, l'unica differenza è che nella prima (*Figura 29*) ritagliando l'immagine vi è una perdita di informazioni, mentre nell'altro metodo l'intera scena viene mantenuta inalterata e di conseguenza le informazioni contenute in essa. Il fatto di fornire ritagli, quindi solo una parte delle informazioni, a una

Figura 29: Tagliando l'immagine (area del riquadro blu) si escludono alcune informazioni della scena

rete che è stata addestrata su immagini complete può portare a una errata previsione delle mappe.

Le due modalità di pre-processing, quindi, non sono completamente compatibili, per cui è necessario rimuovere la prima modalità e al suo posto inserire il ridimensionamento senza ritaglio delle immagini, come in Godard. Per quanto riguarda invece i valori dei pixel le due strutture utilizzano lo stesso formato, i valori sono compresi tra 0 e 1.

Figura 30: Ridimensionando si hanno tutte le informazioni presenti nella scena, ma si introduce una deformazione

Nonostante l'adattamento del pre-processing delle immagini, le mappe risultano essere ancora completamente nere, quindi non è l'input, ma qualcosa che lega i due modelli.

Per verificare ulteriormente che il problema non sia relativo al caricamento dei pesi, si è deciso di provare una rete differente da quella di Godard in grado di generare mappe precise. La nuova rete a differenza delle precedenti prende in ingresso valori compresi tra 0 e 255, per cui si rende fondamentale rimuovere la normalizzazione delle immagini in ingresso. Con la nuova rete le mappe iniziano a comparire, mettendole però a confronto con le uscite, che la rete dovrebbe fornire, risultano essere errate.

Un ultimo test che si è voluto effettuare è quello di utilizzare la rete originale di DualGAN, mantenendo però sempre il calcolo dei loss come in Godard. Con questo ulteriore test si vuole capire se la rete di DualGAN è in grado di gestire il task, e nel caso lo fosse di come modificarla per rendere questo modello la nuova base di partenza per una possibile versione finale.

Per il test si è deciso di non toccare il dataset e di “aiutare” il modello cercando di caricare dai due domini le coppie di immagini. Durante il training si vede subito che la rete non è in grado di gestire la complessità, infatti in questo caso fa fatica anche a generare le immagini a colori, questo però non implica automaticamente che il modello non sia in grado di generare le immagini per il nostro scopo. Per capire se è possibile risolvere il problema si è cercato di aumentare la profondità della rete, andando così a incrementarne la complessità. Nonostante questa nuova modifica la rete fatica a generare le immagini, la conclusione è che il modello non va bene per questo tipo di problema.

5.2 Conclusioni

Dalle varie prove effettuate si vede che il sistema creato è in grado di generare le immagini monoculari e quelle stereo a partire dalle disparità target, ma non è in grado di generare le mappe a partire dalle immagini a colori. Una possibile motivazione è che i due sistemi, quando legati assieme non sono più in grado di “elaborare” i dati in modo corretto.

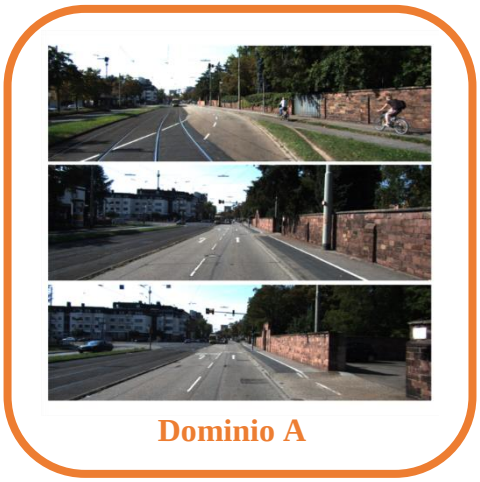
Il modello di rete del primo sistema non è quindi completamente compatibile con il metodo di training del secondo, questa incompatibilità è osservabile solamente nel momento in cui il training arriva ad avere un numero di iterazioni abbastanza elevate.

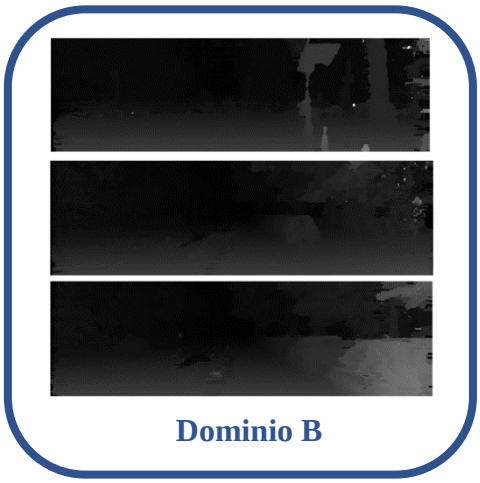
Per evitare di complicare il modello si è deciso di lasciare la strada del training non supervisionato e dell’architettura DualGAN e cercare di creare un modello di uguale complessità o meglio ancora un modello con una difficoltà minore. Per riuscire a creare un sistema con queste caratteristiche è fondamentale quindi utilizzare sistemi GAN in cui però il training sia gestito in maniera differente dai due modelli utilizzati in questo approccio.

6 La versione finale

6.1 Il modello

Dato che la versione precedente non ha portato i risultati sperati, si è deciso di cambiare e di rivedere completamente l'architettura. Per poter fare ciò si rende necessario però cambiare i modelli di riferimento e introdurne di nuovi.





Se i problemi del modello precedente sono dovuti alla tipologia di training e a come interagiscono le varie parti tra di loro, si rende necessario cercare una struttura che sia quanto più "semplice" possibile; il modello che presenta queste caratteristiche e si appresta di più all'obiettivo di questo lavoro è quello di *pix2pix*. Come detto precedentemente, questo modello è stato pensato per trasformare un'immagine da un dominio A ad un altro dominio di tipo B nel migliore dei modi. Anche il task che si vuole risolvere è un problema di cambio di dominio, come effettivamente veniva anche fatto nella versione precedente. La scelta di sfruttare pix2pix è che in questo particolare modello si fa uso sempre delle GAN, ma sono leggermente diverse dalle precedenti, infatti in questo caso sono delle *cGAN*. L'idea quindi non è molto diversa da quella che ha portato alla realizzazione della prima versione, in questo caso è necessario anche avere un dataset reale con le mappe di disparità. Visto che l'interesse è quello di realizzare delle immagini reali a partire dalle disparità si è subito deciso di riprendere il dataset kitti e di generare le mappe dalle immagini stereo tramite l'algoritmo SGM. In questo modo il "nuovo" dataset risulta avere le coppie di immagini a colori e mappe di disparità dense.

Le mappe generate da SGM presentano degli artefatti, più evidenti nei contorni dell'immagine e in alcuni tipi di oggetti.

È possibile dividere la progettazione della nuova struttura in più fasi:

1. Modifica della struttura affinché possa caricare le immagini specificate in file di testo
2. Verifica se il modello è in grado di gestire i due task singolarmente
3. Creazione di un sistema che inglobi entrambi i task
4. Training parziale per verificare se il modello funziona
5. Miglioramento della prima rete (immagine-disparità)
6. Verifica ed eventuale realizzazione di una versione migliore
7. Training definitivo
8. Test con immagini non presenti nel training set
9. Valutazione e confronto con i risultati ottenuti dal modello Godard

La prima cosa che si è reso necessario fare è quella di modificare il sistema e fare in modo che non prenda le immagini in modo statico da una cartella predefinita, ma che possa caricare le immagini tramite dei file di testo strutturati. L'utilizzo della lettura da una cartella predefinita semplifica il codice, non serve fare il parsing di file, ma implica che il dataset sia strutturato in maniera fissa e "standard". Rimuovendo la lettura dalla cartella predefinita è possibile rendere il codice più elastico senza dover modificare la struttura del dataset stesso o di dover legare il sistema ad un dataset in particolare. Si è deciso quindi strutturare i file di testo in un formato compatibile con quelli utilizzati in altri paper; la struttura del file risulta essere: per ogni riga viene indicato il percorso per l'immagine e la relativa mappa di disparità, separati da uno spazio.

Per rendere il codice ancora più elastico sono stati aggiunti anche i parametri di esecuzione, con i quali è possibile indicare il file contenente le immagini di train, quello con le immagini di validazione e il file di test. In questo modo il sistema può essere utilizzato con qualsiasi dataset, aventi struttura completamente diversa tra loro e senza dover modificare codice. Oltre ai parametri di esecuzione precedenti vi sono anche altri parametri fondamentali delle reti, tra questi si trovano: il parametro di learning rate, larghezza e altezza dell'immagine in input alla rete, numero massimo di epoche.

Prima di poter passare alla fase successiva, si sono dovute sistemare delle parti di rete, essa infatti prende in ingresso un'immagine a colori e fornisce in uscita un'altra immagine a colori, i due domini sono "omogenei", entrambi possiedono immagini a 3 canali.

Nel nostro caso abbiamo un dominio che è formato da immagini a colori e l'altro che è formato da immagini a toni di grigio, ciò comporta che la rete deve prendere in ingresso un'immagine con 3 canali (RGB) e fornire in uscita la mappa a 1 canale.

$$create\_model(rgbTrain, 3, depthTrain, "rgb\_depth")$$

Dopo queste prime modifiche è possibile verificare se il modello è in grado di "risolvere" il primo task e generare le disparità a partire dalla singola immagine. Se la rete non è in grado di eseguire il primo task, sicuramente avrà numerose difficoltà nel secondo, che risulta essere più complesso del precedente. Per verificare che funzioni tutto, si è deciso di prendere una piccola porzione del dataset, circa 400 immagini, e di eseguire il training su di esse. L'uso di poche immagini permette di velocizzare il training, in questa fase, infatti, non interessa che la rete sia in grado di generalizzare molto bene, è sufficiente verificare che sia in grado di "imparare" a risolvere il problema.

Dopo qualche epoca di addestramento è possibile affermare che la rete è in grado di eseguire il compito (immagine-disparità) senza problemi e in modo molto veloce. Dato che la prima parte funziona si è passati a verificare che sia in grado anche di fare il task più complesso, quello inverso.

Per questa verifica, è necessario modificare la rete e impostarla affinché in ingresso riceva un'immagine con un solo canale e fornisca in uscita l'immagine a colori. È possibile utilizzare il mini-dataset precedente, perché anche in questo caso l'interesse ricade sulla verifica che la rete sia in grado di eseguire il task e non sul grado di generalizzazione.

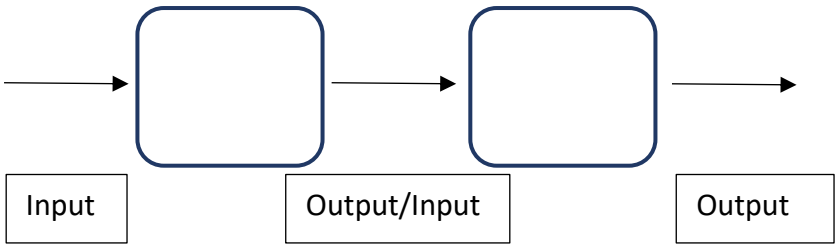
$$create\_model(depthTrain, 2, rgbTrain, "depth\_rgb")$$

Per eseguire il nuovo task bisogna scambiare le immagini target con quelle di input, per fare ciò è sufficiente cambiare i valori con cui viene creato il modello. Per verificarne il corretto funzionamento è stato sufficiente un training di qualche epoca.

Nonostante il mini-dataset la rete è in grado di generare delle buone immagini a partire dalle disparità, questo vuol dire che il modello, con il dataset completo e con qualche accorgimento è in grado di gestire i due problemi senza particolari difficoltà.

L'obiettivo successivo è quello di modificare l'architettura affinché sia in grado di eseguire i due compiti.

Questa nuova architettura non è possibile renderla uguale a DualGAN perché presenta un metodo di training che andrebbe modificato radicalmente e ciò porterebbe ad avere le stesse eventuali criticità riscontrate precedentemente. Partendo dalle verifiche effettuate in precedenza, un'ipotesi è quella di impostare il progetto con blocchi in cascata, cioè fare in modo che la seconda rete sia dipendente dalla prima.



Con questa tipologia di struttura è quindi possibile avere le due reti legate tra loro, ciò comporta che in fase di training il miglioramento della prima porta ad un miglioramento della seconda. Per rendere però la struttura in cascata si rende necessario aggiungere alcuni accorgimenti al metodo di generazione delle reti.

Prima di tutto è necessario riuscire a creare due reti distinte all'interno della stessa sessione di TensorFlow, ognuna delle due avrà a disposizione un suo ottimizzatore con il compito di minimizzare i loss, in modo separato. In pratica all'interno della singola sessione esisteranno due reti con due strutture di training separate, ma al contempo in modo indiretto i due sistemi interagiranno tra di loro. Con la realizzazione di questo sistema si è voluto anche introdurre un nuovo componente, non presente nella versione precedente, che ritornerà utile nelle fasi successive. Esso riguarda la parte di valutazione delle reti ogni *x* iterazioni, questo permette di controllare il grado di generalizzazione ed

evitare l'overfitting[8] durante la fase di training finale. Per rendere ulteriormente variabile il dataset si introduce anche il ritaglio dell'immagine, rispetto alla versione precedente si è deciso di eseguire il ritaglio in modo random, così da fare in modo che il training set sia sempre variabile e la rete possa sfruttare tutte le informazioni contenute nelle immagini. Inoltre, con il ritaglio casuale si è in grado di ridurre il numero di immagini contenenti artefatti.

Per far sì che il ritaglio sia efficace è necessario che le immagini e le relative mappe target siano tagliate in modo coerente. Come detto in precedenza in questo modello vi è la necessità di fornire alla rete delle coppie di immagini strettamente correlate tra loro, questo perché non vi sono loop che permettano alle reti di riuscire a gestire le immagini target in modo non vincolante con quelle in ingresso. Con ritaglio coerente si intende che l'immagine a colori e la disparità devono essere ritagliate nello stesso punto e modo (*Figura 29*).

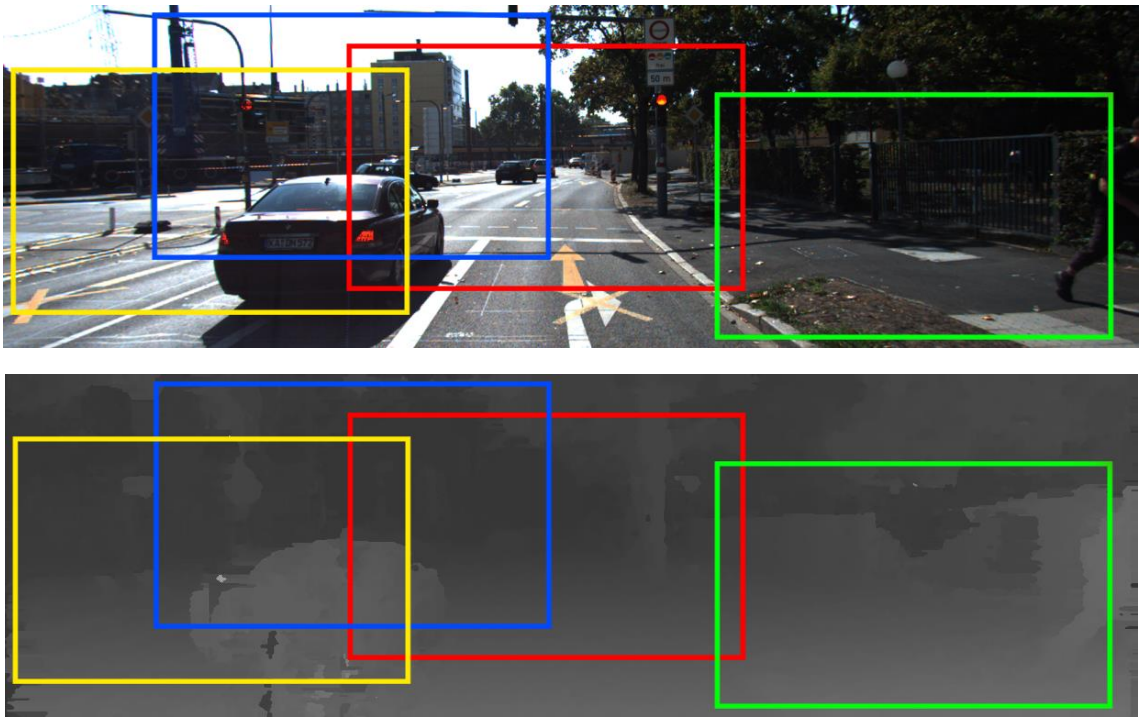


Figura 29: Ritaglio della coppia di immagini in modo random e congruente

[8] Soprattutto nei casi in cui l'apprendimento è stato effettuato troppo a lungo o dove c'è uno scarso numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche specifiche del training set e che non hanno riscontro nel resto dei casi.

Per eseguire i ritagli in modo coerente è sufficiente utilizzare alcune funzioni messe a disposizione dal framework: *tf.random_crop*, *tf.concat* e *tf.split*. La prima funzione esegue un ritaglio random dell'immagine lungo ogni suo canale, la seconda permette di concatenare più immagini lungo un asse a scelta e la terza permette di dividere l'immagine in più parti. Per far sì che il ritaglio sia uguale in quella a colori e nella disparità è sufficiente concatenare le immagini lungo l'asse dei colori, dopo di che basta tornarle a dividere e ottenere così dei ritagli uguali. La funzione di concatenazione permette di unire immagini con un numero differente di livelli di colore.

$$image_{concat} = tf.concat([raw_{rgb}, raw_{depth_l}, raw_{depth_r}], -1)$$
$$image_{crop} = tf.random(image_{concat}, [height, width, rgb_{ch} + depth_{ch}])$$

$$raw_{rgb}, raw_{depth} = tf.split(image_{crop}, [rgb_{ch}, depth_{ch}], -1)$$

Ogni qual volta il training richiede delle nuove immagini, la funzione di crop fornisce un ritaglio diverso dai precedenti, rendendo così il training set "virtualmente" più grande. Con questa modifica quindi è possibile ridurre la probabilità che la rete vada in overfitting durante il training. Con la struttura impostata in modo che le due reti siano una in cascata all'altra, la prima riceve in ingresso le immagini a colori e genera le disparità; la seconda riceve in ingresso le disparità prodotte dalla prima e genera le immagini a colori di partenza è quindi possibile fare partire il training.

Avendo provato con un "mini-training" si ha quasi la certezza che il sistema possa funzionare, i parametri da controllare in TensorBoard sono i loss delle discriminatrici e delle generatrici, che devono essere minimizzati e il loss totale della GAN che deve crescere. Altri fattori da controllare sono le immagini che le due reti ricevono in ingresso e quelle che producono in uscita. Un particolare del sistema realizzato riguarda la struttura degli ingressi, le reti possano riceve in ingresso dei batch con un numero di immagini fissato prima dell'esecuzione dello script tramite i parametri di esecuzione. L'utilizzo di batch permette di velocizzare il training e di ridurre il rumore nei gradienti, senza il loro utilizzo le reti posso ricevere in ingresso una sola immagine alla volta e basandosi sull'immagine corrente aggiustare i vari pesi, con conseguente bassa velocità di training e gradienti rumorosi.

Dopo qualche epoca di training è possibile osservare come le reti siano già in grado di gestire i due task e fornire in uscita delle immagini molto verosimili alle originali.

Osservando bene le immagini però è possibile osservare come compaia un effetto scacchiera.

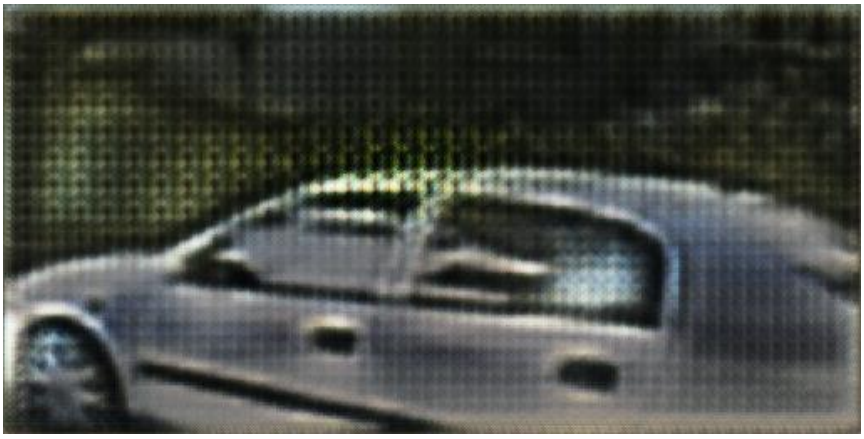


Figura 30: Immagine con effetto a scacchiera ottenuta dopo qualche epoca

L'effetto scacchiera nelle GAN

Quando le reti generano le immagini, le costruiscono da "descrizioni" a bassa risoluzione e di alto livello. Ciò consente alla rete di descrivere l'immagine approssimativa e quindi di inserirvi i dettagli, per fare ciò, si utilizzano i livelli di deconvoluzione. Sfortunatamente però la deconvoluzione può facilmente avere delle sovrapposizioni disomogenee, mettendo più informazioni ("colore") in alcuni punti rispetto ad altri. Questa sovrapposizione disomogenea può avvenire quando la dimensione del kernel non è divisibile per lo spostamento dello stesso (stride).

Il modello di sovrapposizione si può formare anche nelle due dimensioni. Le sovrapposizioni irregolari sui due assi si moltiplicano, creando questo caratteristico schema a scacchiera.

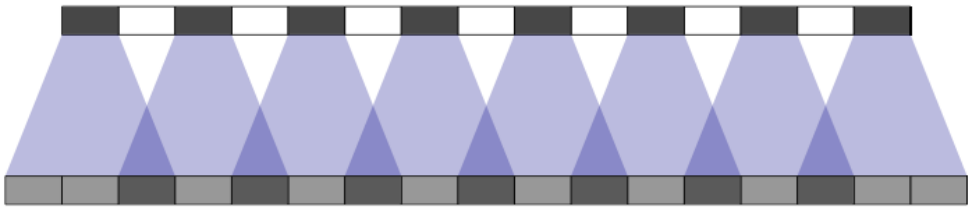


Figura 31: Sovrapposizione disomogenea in una dimensione

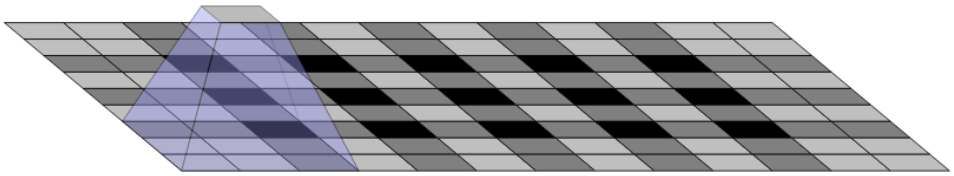


Figura 32: Sovrapposizione disomogenea in due dimensioni

In teoria, i modelli potrebbero imparare a gestire questo tipo di problema, anche se questo è difficile da raggiungere. Una possibilità è quella di modificare i vari kernel in modo che siano divisibili per il passo, questo però porta a limitare in modo significativo i possibili filtri e quindi sacrificare le capacità del modello.

Essendo questo task complesso e articolato è meglio mantenere il modello così com'è senza modificare il kernel o il passo, col rischio di limitarlo e di non riuscire più a ottenere dei buoni risultati. Quindi per questa prima fase si è deciso di non modificare nulla delle reti e verificare se con il passare delle epoche il problema viene risolto in maniera autonoma dal modello.

Durante questa fase di training si sono tenuti sotto controllo anche i parametri relativi ai loss e alle mappe di disparità, che nelle versioni precedenti presentavano molti problemi. Com'è possibile osservare dall'immagine, le disparità vengono generate in modo coerente con quelle target, senza che vi siano problemi durante la loro generazione.

Figura 33: Immagine target

Figura 34: Immagine generata

Come si può osservare dalla *Figura 34*, la mappa generata è simile all'originale, ci sono solo alcuni problemi con la definizione dei contorni dei vari oggetti. Inoltre,

dall'immagine generata è possibile osservare la presenza di alcuni artefatti soprattutto nel bordo dell'immagine, in particolare nel contorno dell'automobile, essi sono dovuti all'utilizzo di disparità target realizzate tramite SGM. Per riuscire a generare delle uscite corrette la rete si basa su immagini target SGM, per cui acquisisce anche la "capacità" di riprodurre gli artefatti presenti in esse.

Il modello per ora realizzato è in grado di gestire la complessità dei task, per migliorare ulteriormente la prima fase, cioè quella di generazione delle mappe è possibile introdurre un ulteriore passaggio che aiuti la rete a generare immagini migliori e di conseguenza porti ad un miglioramento della seconda. Come idea base è possibile adottare un sistema simile alla prima versione dove erano presenti i loop.

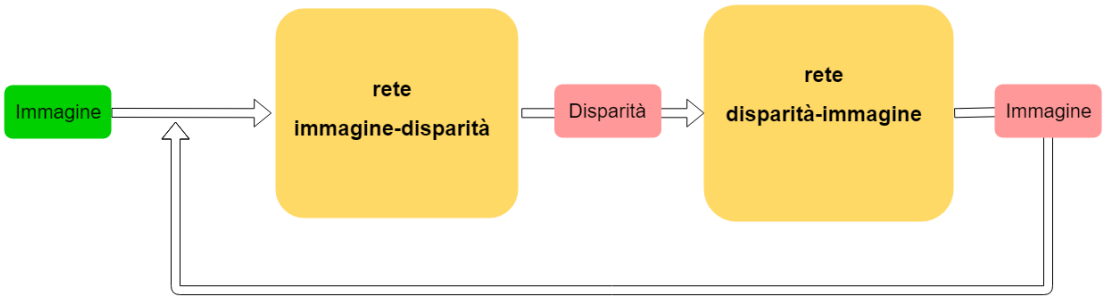


Figura 35: loop teorico tra la prima e la seconda rete.

Con l'introduzione del loop all'interno del sistema si introduce un ulteriore collegamento tra le due reti, questa volta però non è più indiretto come l'altro, ma permette di sfruttare direttamente le informazioni generate dalla seconda rete per migliorare le disparità della prima. Lo schema di principio è quindi quello di sfruttare la seconda rete per "aiutare" la prima a gestire meglio la propria uscita. Attraverso questa nuova configurazione si è in grado anche di rendere il modello più simile alla versione DualGAN, e quindi di riuscire ad avvicinarsi a una versione semi-supervisionata. L'introduzione del loop porta anche a delle modifiche nella struttura del grafo, infatti per riuscire ad aggiungere il "collegamento" tra l'uscita della seconda rete e l'ingresso della prima si rende necessario aggiungere una terza rete che funga da "collegamento" tra le due.

In pratica, questa rete deve comprendere sia la parte di training che quella del modello vero e proprio, infatti in questa tipologia di architettura non è possibile separare completamente le due parti. Quindi in questo caso, anche se parliamo di rete è più corretto dire che vi è l'aggiunta di un terzo modello alla struttura generale.

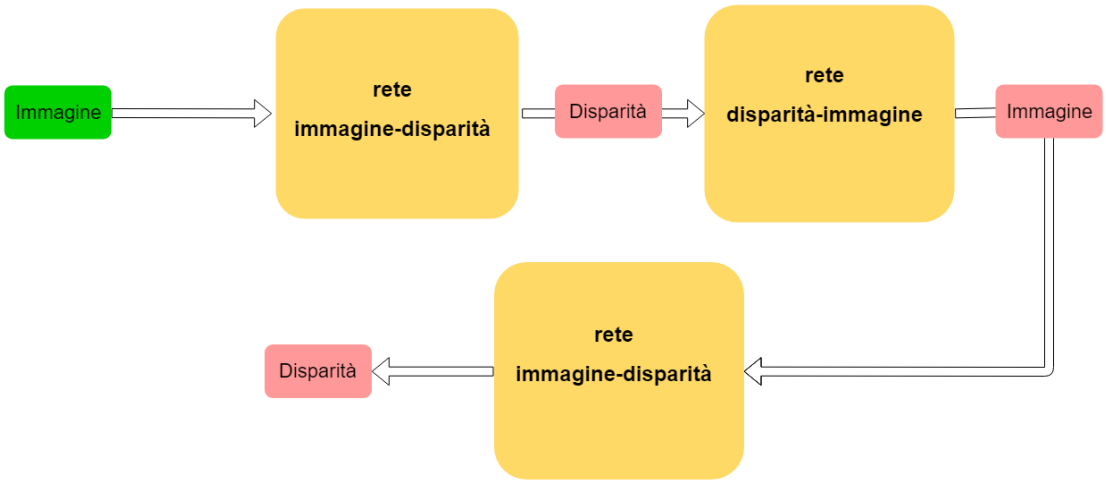


Figura 36: Realizzazione pratica del loop

Osservando lo schema di *Figura 36* è possibile osservare la presenza di tra macro-blocchi, i primi due in alto sono quelli presenti fin dall'inizio, mentre il terzo è quello inserito per riuscire a creare il loop. La particolarità di questi blocchi è che al loro interno, come detto prima, non vi è solo la parte di rete, ma anche quella di addestramento, è possibile quindi affermare che in realtà il sistema è formato da tre modelli distinti, che però interagiscono tra loro per arrivare a portare a termine un obiettivo comune. Questa terza rete (contenuta all'interno del nuovo blocco) condivide i pesi della prima, ma non possiede lo stesso ottimizzatore, l'utilizzo di un ottimizzatore separato è d'obbligo perché in questo caso non è possibile condividerlo tra i macro-blocchi. L'utilizzo pratico di questa terza "rete" si rende necessario perché in questo modello non vi è l'uso dei *placeholder*[9], i quali invece avrebbero permesso di evitare l'aggiunta della "rete" e l'inevitabile incremento di overhead del sistema durante la fase di training. Per questa seconda fase di training si è deciso di non inserire i placeholder e di verificare se il modello è ancora in grado di

[9] Permette di passare dati alla rete in modo dinamico senza che vi sia un legame statico tra la variabile in ingresso e l'input stesso della rete

generare delle immagini, nonostante il training possa impiegare più tempo per terminare la singola epoca.

Nella fase di training è possibile osservare un overhead, si è passati dall'elaborazione di quasi 5 immagini al secondo a 3.5, questo comporta una fase di training più lunga, ma non va a influire sull'apprendimento delle reti. Per riuscire a comprendere cosa generi la

Figura 37: Disparità generata dalla nuova rete durante la prima epoca di training

nuova rete, si sono aggiunte tutte le informazioni possibili a TensorBoard, così che da avere tutti parametri sotto controllo. Con l'uso del collegamento tra la prima e la seconda rete è possibile osservare un miglioramento delle disparità prodotte dalla prima rete e come la nuova rete sia in grado di generare delle mappe a partire anche da immagini poco definite e non ancora realistiche.

L'aggiunta di questa nuova parte al modello, ha portato benefici, ma non ha risolto il problema dell'effetto a scacchiera, presente anche nella disparità. Come per le immagini anche in questo caso, prima di modificare la rete è meglio verificare che il problema non si risolvi da solo nella fase avanzata del training.

Dato che il sistema sembra funzionare, almeno per le prime epoche, si è deciso di provare ad inserire i placeholder, cosicché da poter eventualmente rimuovere la terza rete e ridurre l'overhead risultante da essa. Per l'inserimento dei placeholder si rende necessario modificare l'input delle reti e anche alcune parti del training.

La struttura del placeholder prende in ingresso le dimensioni delle immagini ed eventualmente il loro numero, questo elemento può essere inserito come valore "non conosciuto" attraverso il valore *None.* In fase di training, permette di passare batch con un numero di immagini superiori a uno, mentre, durante la fase di inferenza è possibile passare un numero di immagini diverso da quello utilizzato durante il training. In questo modo durante l'inferenza è possibile passare anche una singola immagine senza dover riaddestrare tutto.

$$tf.placeholder(\boldsymbol{dtype} = tf.float32, \boldsymbol{shape} = [None, height, width], 'input\_rgb')$$

Per come è strutturata la creazione della rete non è possibile inserire il *None*, per cui per la prima parte di prove è sufficiente impostare il valore a 8. Nel caso in cui i placeholder dovessero funzione si possono sistemare e introdurre la possibilità di rendere variabile il numero di immagini in ingresso.

L'introduzione dei placeholder presenta una quantità di modifiche non indifferenti nella parte di scrittura dei *summary* in TensorBoard, per cui per un primo momento si è deciso di rimuovere una parte delle immagini di summary e lasciare solo quelle utili a capire se il sistema presenta dei problemi.

Con l'introduzione dei placeholder bisogna anche modificare una parte del training, in particolare bisogna cambiare la parte in cui la rete viene eseguita all'interno della sessione di TensorFlow. In questa parte del codice è fondamentale passare ai palceholder le immagini che vogliamo fornire alle reti; nella versione precedente, questa tipologia di operazione non bisognava eseguirla, questo perché le reti avevano già gli ingressi fissati in modo statico sulle variabili che contenevano le immagini.

$$sess(train_{operations}, options = options, feed\_dict = \{rgbP: rgb, depthP: depth\})$$

Attraverso *feed_dict* è possibile fornire ai *placeholder* (rgbP e depthP), le variabili contenti le immagini da passare alla rete.

Dopo l'introduzione dei placeholder, nelle prime iterazioni del training il modello non sembra presentare problemi, anche se le sue uscite sono differenti da quelle precedenti.

Figura 38: In alto la disparità in ingresso alla rete e sotto la relativa uscita

Soltanto verso la fine dell'epoca però si nota subito che il sistema non si comporta più come prima, ma presenta dei problemi durante l'apprendimento (*Figura 38*).

Dall'immagine in alto è evidente come la rete, dopo l'introduzione dei placeholder, non riesca a generare delle buone immagini nonostante in ingresso riceva l'immagine come nelle precedenti prove, senza la presenza di apparenti modifiche dovute ai placeholder. Solitamente l'uso dei placeholder non porta a malfunzionamenti delle reti, in questo caso però, per come sono creati i primi livelli della rete o per come vengono caricate le immagini il loro uso porta la rete a non fornire più uscite corrette.

Per il momento l'uso dei placeholder è da evitare, nonostante essi sarebbero in grado di garantire un uso più efficiente del sistema.

Tornando alla versione prima si è proceduto ad effettuare il training completo di 50 epoche per vedere come funziona effettivamente il modello e per verificare che non compaiano problemi con l'avanzamento dell'addestramento. La scelta di limitare le

epoche a 50 è stata fatta in base al numero di immagini all'interno dei batch, pari a 8, e alla dimensione totale del dataset, 22600. Con un batch di 8 e il dataset da 22600 in 50 epoche si effettuano 141250 iterazioni, sufficienti ad ottenere un modello in grado di generalizzare e di evitare l'overfitting causato da un eccessivo training. Lo stesso numero di epoche è presente anche nel modello originale di Godard.

Terminato il training si sono confrontati alcune immagini di validazione del dataset con le relative uscite, si può notare come la rete sia in grado di generare delle buone disparità, mentre le immagini a colori presentano qualche artefatto. Essendo questo il primo modello in grado di eseguire i task, la presenza di artefatti era una possibilità che era stata tenuta in considerazione già dall'inizio.

Figura 39: Immagine generata a partire dalla disparità in alto

Come è evidente dalle immagini precedenti il nuovo modello riesce ad eseguire sia il task principale, da disparità a immagine che quello inverso. Le immagini generate, nonostante in alcuni casi siano buone (*Figura 39*), hanno ancora parecchio margine di miglioramento. Com'è possibile osservare dalle immagini di *Figura 40* e *Figura 41* in alcuni punti la rete non è in grado di "capire" quale oggetto vada inserito, in altri invece vi è la presenza di pezzi creati in modo random.

Figura 40: Esempio di immagine con la presenza di pochi artefatti

Figura 41: Esempio in cui la rete presenta delle difficoltà nel generare la macchina

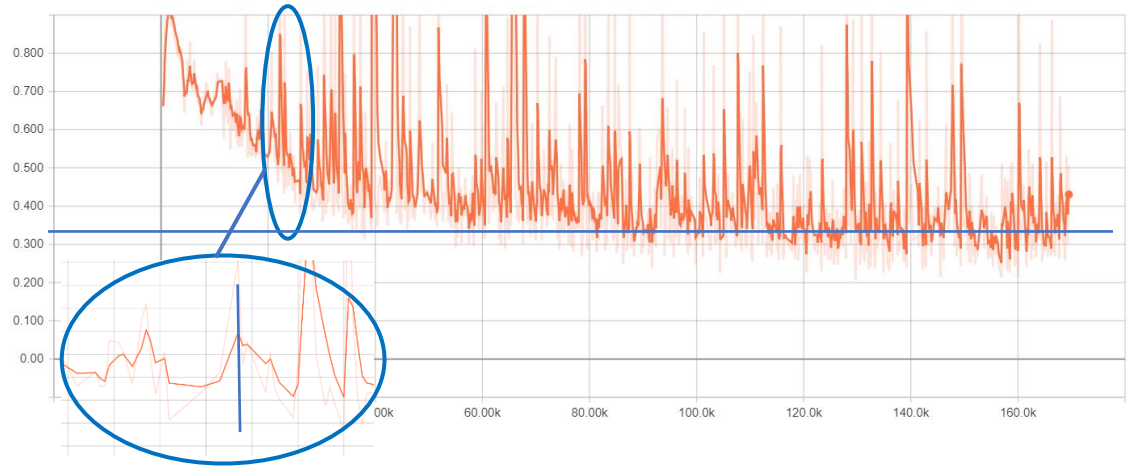


Figura 42: Grafico loss discriminatrice da immagine a disparità

Osservando il grafico di *Figura 42*, si nota che il loss della rete discriminatrice, nonostante sia decrescente presenta molti picchi. La rete discriminatrice inizialmente riesce a distinguere le immagini reali da quelle generate dalla generatrice, ogni volta però che la generatrice migliora la discriminatrice ha difficoltà nel distinguere le immagini (decrescita del valore), a sua volta però la discriminatrice si perfeziona nel distinguere le due immagini e quindi si ha un picco negativo nel loss della GAN e della generatrice.

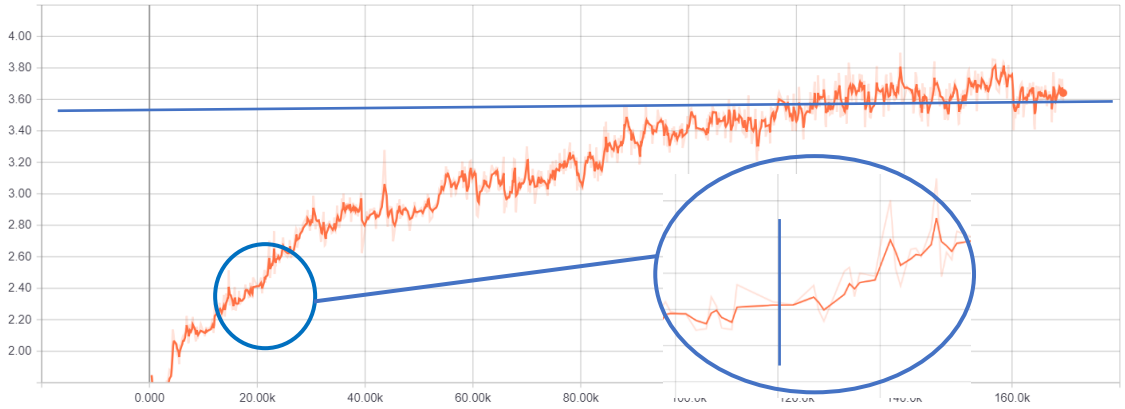


Figura 43: Grafico loss GAN da immagine a disparità

Dal confronto dei grafici è possibile osservare come i tre loss siano "collegati" tra loro, a livello macroscopico è possibile vedere come, mentre il loss della discriminatrice e della generatrice calano nel tempo quello della GAN cresce fino a raggiungere il suo massimo. Osservandoli a livello microscopio è possibile constatare come un picco di crescita nel grafico di *Figura 42* porta a un picco negativo nel loss della generatrice (*Figura 44*) ripercuotendosi di conseguenza anche nel loss della GAN.

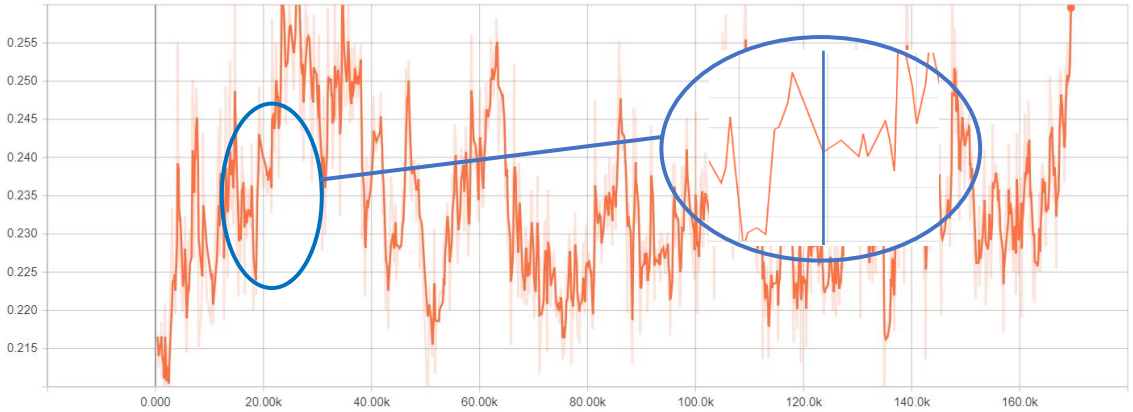


Figura 44: Grafico del loss L1 della generatrice

Dai due grafici si deduce che la rete dopo le 50 epoche non ha un margine molto ampio di miglioramento, infatti osservando i due loss si nota che all'iterazione 140000 iniziano a presentare un asintoto ai valori di 0.350 per la discriminatrice e 3.60 per la generatrice.

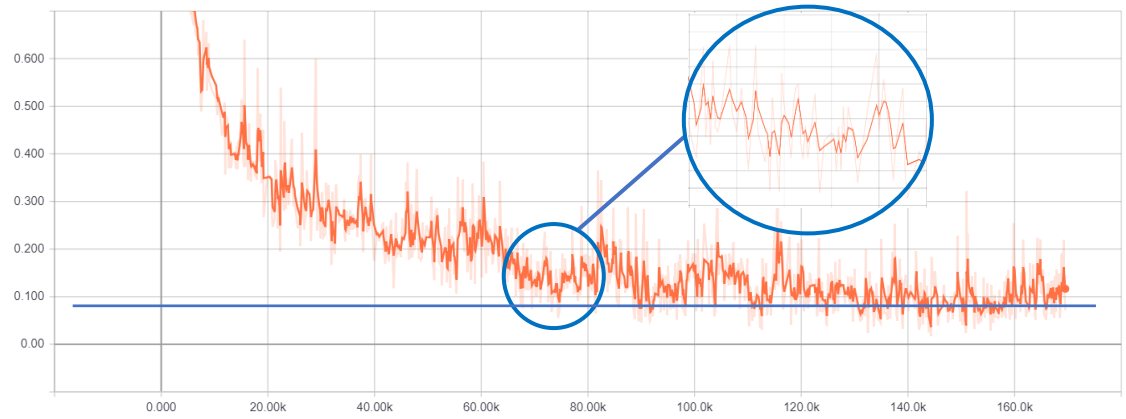


Figura 45: Loss della discriminatrice da disparità a immagine

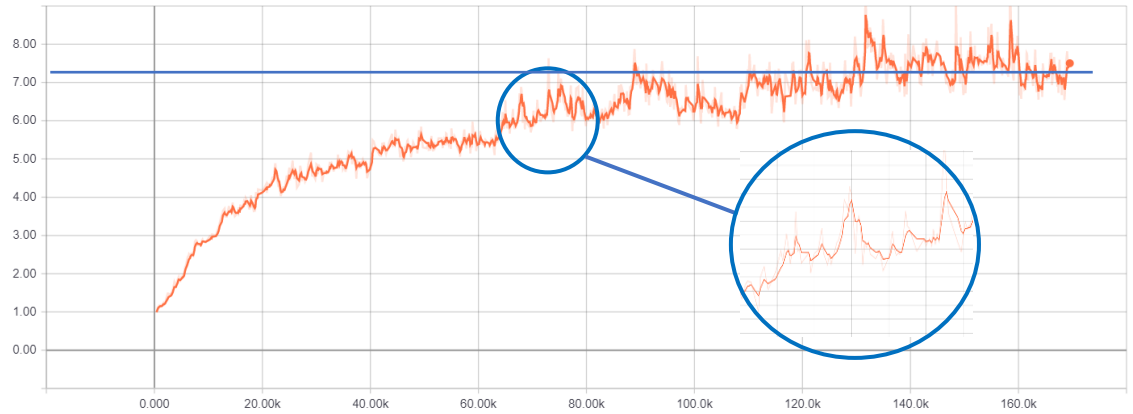


Figura 46: Loss GAN da disparità a immagine

Anche nel caso dei grafici del secondo blocco, che gestisce la “trasformazione” da disparità a immagine, è possibile individuare le stesse caratteristiche dei precedenti. In questo caso però la discriminatrice inizia l’asintoto dopo circa 120000 iterazioni e si stabilizza sul valore di 0.100, mentre per il loss GAN si ha che il valore si stabilizza a 7.40 dopo circa 130000 iterazioni.

Dai grafici si può dedurre come non sia possibile migliorare i loss semplicemente aumentando il numero di epoche, dato che i loss sono già in una fase asintotica è probabile che con il prolungamento della fase di training si introduca dell’overfitting. Per migliorare il tutto si è pensato, come nella versione DualGAN-Godard, di utilizzare non più una singola disparità, ma entrambe le coppie, cosicché la rete abbia il maggior numero di informazioni possibili per riprodurre in modo ancora più preciso l’immagine a colori. Il nuovo modello quindi, è costituito sempre da tre macro-blocchi come nel precedente, ma in questo caso il blocco che è dedito alla creazione delle immagini a colori deve ricevere in ingresso due mappe e di conseguenza il primo e il terzo blocco a partire dalla singola immagine devono riuscire a generare le due disparità, quella sinistra e quella destra. Con questo sistema è possibile fornire più dati alla seconda rete, di conseguenza semplificarle leggermente il lavoro di “capire” come deve generare i vari oggetti ed ottenere immagini più realistiche e migliori.

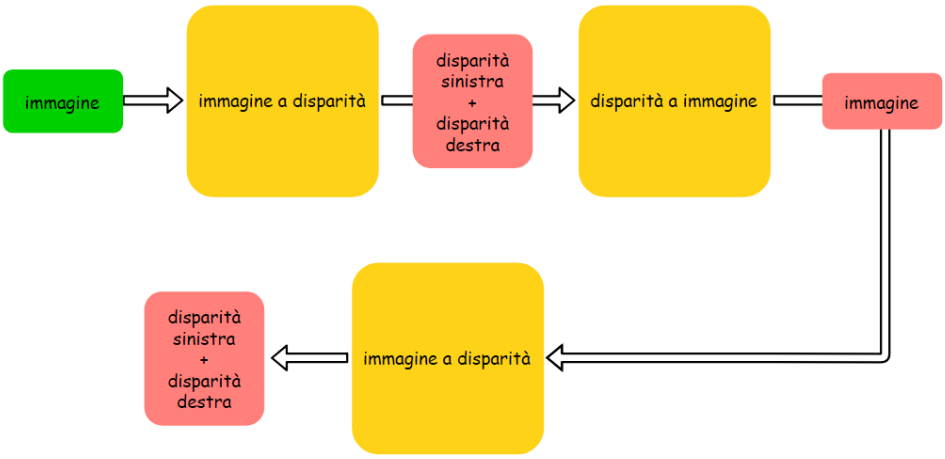


Figura 47: Schema della versione con le due disparità

L’unica modifica da fare al sistema è quella di caricare non più la singola mappa, ma entrambe le disparità che SGM ha generato.

Figura 48: Disparità sinistra (In alto) e disparità destra (In basso)

Dalla *Figura 48* è possibile vedere come le due mappe siano molto simili tra loro, ma presentano alcune differenze. Tra le principali vi sono quelle dovute alla creazione delle mappe usando come telecamera di riferimento quella sinistra e quella destra. Oltre al fatto che tra le due immagini gli oggetti risultano essere traslati è possibile osservare come, nella mappa destra, alcuni oggetti presentano più dettagli rispetto a quelli presenti nell’altra mappa, un esempio è il cartello attaccato al palo, in cui nell’immagine sinistra è quasi completamente assente. Inoltre, l’introduzione della seconda mappe fornisce alla rete un secondo punto di vista della scena.

Con questa nuova configurazione la rete è in grado di generalizzare meglio e di ottenere risultati migliori con immagini che non ha mai visto. Il miglioramento della generalizzazione della rete è una cosa fondamentale, anche nella prima versione era in grado di ricostruire le immagini da scene che non erano presenti nel dataset, ma con questa modifica dovrebbe essere in grado di riprodurre le scene in maniera ancora più fedele alle originali. Inoltre, con l’ulteriore miglioramento della generalizzazione, la rete è in grado di ricostruire l’immagine anche in scene differenti da quelle con cui è stata

addestrata, ovviamente da queste tipologie di scene non si avranno gli stessi risultati ottenuti con le altre.

6.2 Training finale

Il modello precedente, nonostante sia quello che fino a questo momento ha fornito i risultati migliori, presenta molti artefatti nella generazione delle disparità. Per cercare di ridurli è stato modificato ulteriormente aggiungendo le immagini stereo in ingresso, così facendo dovrebbero ridursi gli artefatti presenti nelle mappe senza che le immagini generate abbiano un calo di qualità.

Con l'introduzione delle immagini stereo si è provveduto anche a rimuovere la terza rete, la quale andava a migliorare le disparità tramite l'utilizzo delle immagini generate dalla seconda rete. La terza rete in questo caso non è più necessaria, adesso la rete ha molte più informazioni con cui lavorare e non necessita più di ricevere un feedback dalla rete dopo.

Per questa fase di training si è partiti con i pesi della rete caricati in modo random ed è stato suddiviso il dataset kitti in 3 set distinti: una parte viene utilizzata come train-set, le immagini vengono usate per addestrare la rete; una seconda porzione come validation-set, più piccolo del precedente e in questo caso le immagini contenute in esso vengono sempre usate a tempo di training, ma per verificare il grado di generalizzazione; l'ultima parte invece è il test-set, quest'ultimo viene utilizzato alla fine del training a tempo di test.

Grazie all'introduzione del caricamento delle immagini da file di testo, la creazione di questi "dataset" non necessita di apportare modifiche alla struttura di kitti, è sufficiente creare i file: trainset.txt, validationset.txt e testset.txt. Per mantenere un certo collegamento tra questo lavoro e gli altri, anche per poi riuscire a fare eventuali confronti si è deciso di utilizzate i file eigen_test, eigen_train ed eigen_val, che sono diventati un riferimento in questa tipologia di lavoro.

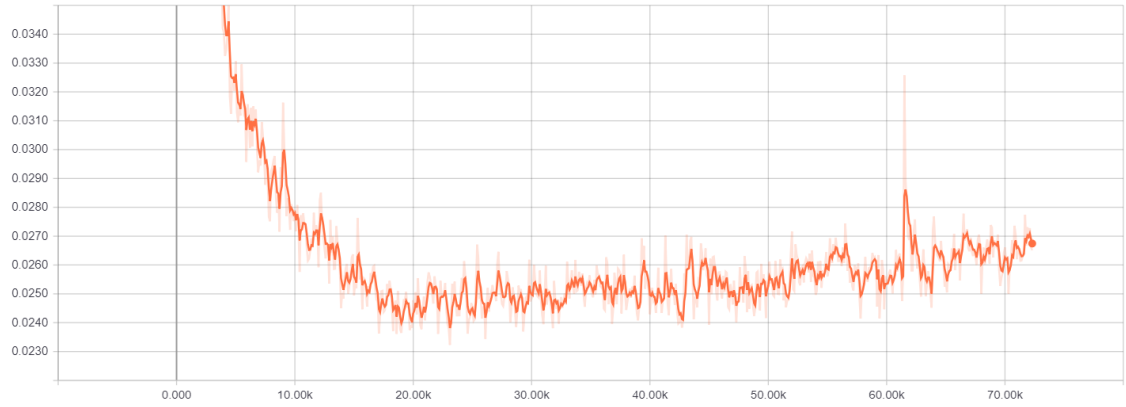


Figura 49: Loss generatrice immagine disparità

Come nelle prove precedenti anche in questo caso il loss della generatrice e della discriminatrice si comportano in modo simile.

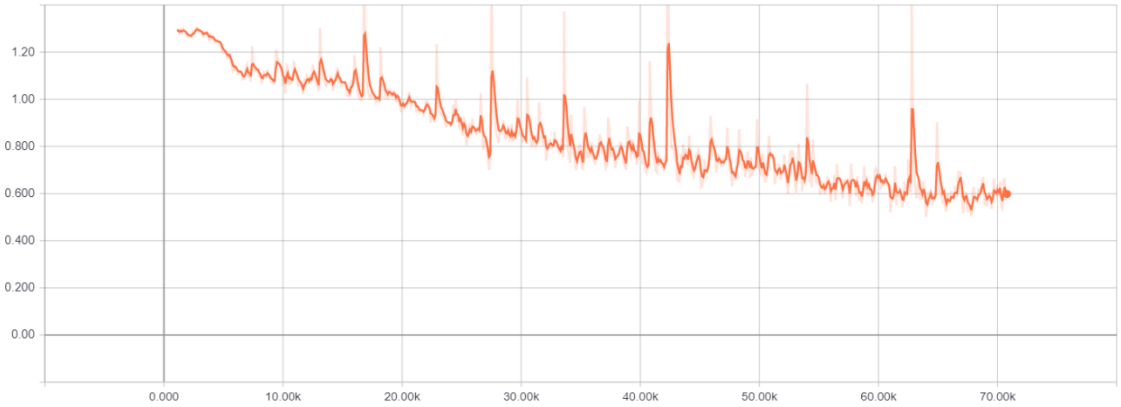


Figura 50: Discriminatrice immagine disparità

Il loss della discriminatrice decresce correttamente, ciò indica che la generatrice, nonostante il grafico sia a 0.0250 e inizi leggermente a crescere, sta imparando a generare disparità sempre migliori e di conseguenza la discriminatrice inizia a fare sempre più fatica a distinguere quelle reali dalle altre.

La stessa cosa è osservabile per la seconda parte del modello.

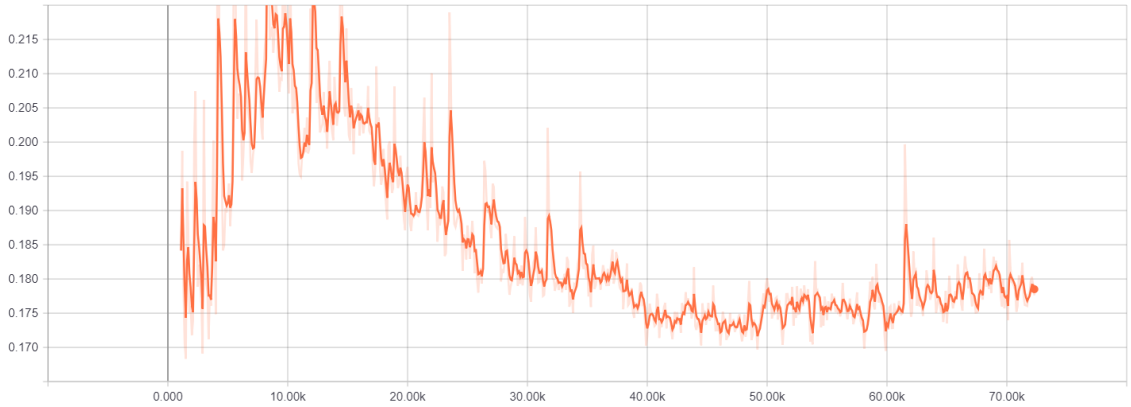


Figura 51: Loss generatrice disparità immagine

Una cosa che differenzia i due grafici dai precedenti è il comportamento della generatrice, nel caso prima l'apprendimento era più lineare, c'era la decrescita iniziale e poi l'assestamento con una leggera crescita. Mentre in questa sezione si vede come c'è stata una crescita iniziale, quindi un peggioramento della rete, ma poi c'è stata una decrescita meno repentina e più frastagliata dell'altra.

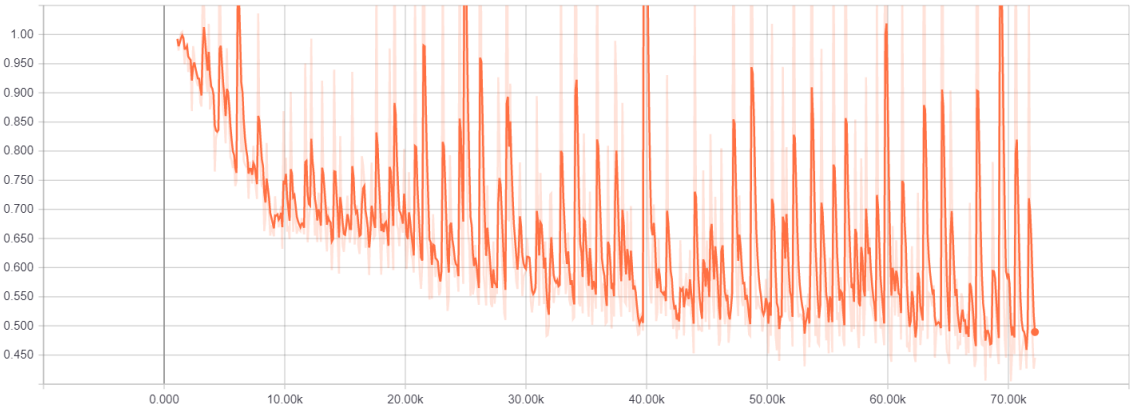


Figura 52: Loss discriminatrice disparità immagine

6.3 Confronto tra versione singola e doppia disparità (monoculare)

Tra le due versioni le differenze non sono molte, quelle principali sono:

- Utilizzo di più disparità
- Struttura della rete

L'utilizzo di più disparità è la principale differenza, le altre sono conseguenza di essa. Com'è possibile osservare dalle immagini di seguito la prima rete presenta difficoltà nel ricostruire delle piccole parti dell'immagine, andando così a creare degli artefatti che ne riducono la verosimiglianza con le immagini reali. Questo non crea grossi problemi nel caso in cui l'utilizzo della rete si fermi alla sola creazione, ma se dovessimo usare questa tipologia di immagine per creare un nuovo dataset generato a partire dalle disparità, per addestrare una rete, si avrebbero difficoltà durante l'inferenza della stessa. La scelta delle immagini per creare un dataset è di estrema importanza, durante il training l'utilizzo di immagini non veritiere, come succede ad esempio con quelle sintetiche, portano la rete ad avere problemi nel caso in cui poi in ingresso si forniscano immagini che non appartengono allo stesso dominio. Esistono diversi modi per cercare di arginare il problema, ma in questo caso è meglio fornire già da subito delle immagini il più corrette possibili.

Figura 53: Immagine generata dalla prima versione

Le due versioni del modello presentano risultati differenti, nella prima l'utilizzo di una sola disparità non permette alla rete di riuscire a ricostruire tutta l'immagine in maniera corretta, questo lo si nota soprattutto quando cerca di generare i veicoli.

Nel caso dei veicoli si osserva come la rete riesca ad abbozzarne il contorno, ma poi quando deve inserire i dettagli inizia ad avere difficoltà e genera dei pattern casuali.

Figura 54: Immagine generata dalla seconda versione

Nella seconda versione invece la rete è in grado di gestire molto meglio le parti in cui vi sono veicoli, ed è in grado di riprodurli senza molti artefatti. Dal confronto tra le due si evince anche come la rete sia riuscita a migliorare nel resto dell'immagine, diventando in grado di riprodurre le ombre e di inserirle in modo congruo nella scena. Infatti, le ombre generate dalle macchine sull'asfalto sono tutte rivolte verso la stessa direzione, ciò rispecchia abbastanza fedelmente quella che si potrebbe avere nella scena reale.

6.4 Aggiunta dei placeholder al modello

Visto che il modello è in grado di generare delle buone immagini, anche in scene mai viste a tempo di training si è voluto sistemare la parte di input e di aggiungere i placeholder. Già nella versione precedente si era tentato il loro utilizzo, ma senza risultati.

$$rgbP = tf.placeholder(tf.float32, shape = [8, height, width, 3])$$

$$depthP = tf.placeholder(tf.float32, shape = [8, height, width, 2])$$

In questo caso è stato necessario definire il numero di immagini in ingresso al placeholder perché nella fase successiva la creazione della rete necessita di quel valore.

Figura 55: La versione precedente dopo l'introduzione dei placeholer

Anche in questa versione l'introduzione dei placeholder porta allo stesso problema, nonostante le immagini in ingresso non presentino anomalie. L'introduzione dei placeholder, *rgbP* e *depthP* riguarda solamente l'ingresso della prima rete e gli ingressi per le immagini target.

$$model\_rToD = create\_model(\boldsymbol{rgbP}, 3, \boldsymbol{depthP}, "rgb\_depth")$$

$$model\_dToR = create\_model(model\_rToD.outputs, 2, \boldsymbol{rgbP}, "depth\_rgb")$$

Il problema che crea le anomalie in *Figura 53* deve essere tra la parte di caricamento dell'immagine e il passaggio al placholder, probabilmente c'è un problema di inconsistenza tra i formati che il modulo di caricamento fornisce e l'input che si aspetta il placeholder.

Nella parte di caricamento si effettuano principalmente 5 passaggi:

- Lettura del file
- Parsing delle righe lette
- Lettura dell'immagine
- Decodifica dell'immagine
- Creazione dei batch

Se il problema riguardasse i primi 3 punti, si avrebbe lo stesso risultato anche nel sistema senza placeholder, gli unici due punti che posso influire sono gli ultimi. La creazione dei batch sfrutta la funzione $tf.train.batch$, il sistema restituisce un tensore tramite il quale poi è possibile ricevere i vari batch.

I placeholder possono riceve in ingresso tensori, per risolvere il problema è sufficiente richiamare la funzione *eval()* del tensore ed ottenere le immagini da passare al placeholder, è necessario fornire anche la sessione attualmente in uso nel framework. Visto che queste operazioni non modificano in alcun modo le immagini il problema non può riguardare i batch, ma il punto precedente: la decodifica.

Le immagini, una volta caricate da disco, vengono decodificate in base al formato in cui sono, in questo caso viene effettuata la decodifica png. La decodifica permette di ottenere i byte decompressi dell'immagine ed è possibile ottenere i valori in differenti *dtype*, nel caso in cui non venga definito in modo esplicito la funzione restituisce delle stringhe. I placeholder però sono stati impostati in modo che in ingresso ricevano dei tipi float32, è necessario quindi trovare un formato comune ai due. Il primo può fornire soltanto valori di tipo *string*, *uint8* e *uint16*, il secondo può riceve in ingresso float16, float32, uint8, string. Il formato *string* nei placeholder non permette di impostare un valore di *shape*, per cui in questo caso, senza una rivisitazione del resto, non è utilizzabile. Il formato uint8 sarebbe quello comune, ma il resto della rete opera su valori flaot32, non essendo possibile mantenere entrambi i tipi, non è utilizzabile.

Per poter utilizzare i placeholder, senza modificare troppo il codice, si è pensato di effettuare il cast delle string in float32, tramite l'apposita funzione di TensorFlow. Con questo stratagemma quindi è possibile rendere i due pezzi omogenei.

$$rgb = tf.cast(train\_images.rgbTrain, tf.float32).eval(session = sess)$$

$$depth = tf.cast(train\_images.depthTrain, tf.float32).eval(session = sess)$$

L'avvio della fase di training ha permesso di constatare che il problema è stato risolto e la rete è tornata ad imparare come in precedenza. Se ne è introdotto però uno nuovo, le immagini fornite dal sistema che genera i batch non restituisce più le immagini a coppie corrette, ma in modo casuale. Per cercare di risolvere il problema sono state eseguite varie prove, come ad esempio eliminare pezzi di codice superflui, modificare il metodo con cui le immagini vengono inserite nei batch. Nonostante i vari tentativi fatti il problema resta e dato che questo sistema deve avere per forza delle coppie strettamente legate si rende necessario abbandonare i placeholder e passare alle fasi successive.

6.5 Aggiunta della parte di testing al modello

Per poter eseguire alcuni test di "qualità" del modello si rende necessario riuscire ad utilizzarlo escludendo la parte di training, lasciando quindi solo la parte di inferenza. La parte di test inoltre permetterà di utilizzare i risultati della rete per fare un confronto con quelli di Godard.

Al fine di poter sfruttare i risultati del test è necessario aggiungere il caricamento delle immagini da file, come per quelle di training la loro struttura è molto semplice, in ogni riga vi sono: l'immagine a colori e le due relative disparità. Una parte fondamentale riguarda la parte operativa del modello, se durante la fase di training non serviva caricare i pesi della rete, adesso si rende necessario anche aggiungere il caricamento della rete salvata a fine training. Il framework TensorFlow mette a disposizione una funzione che permette di caricare i parametri a partire da un checkpoint del training, in questo caso da quello finale. È possibile dividere il caricamento del checkpoint in tre fasi: la prima fase riguarda la creazione della sessione di TensorFlow, una volta creata è necessario implementare le reti.

Nella seconda fase, è sufficiente creare solo l'inferenza del sistema, cioè tutta quella parte che permette di utilizzare la rete ed escludere le sezioni di training. In questo caso è sufficiente creare le due generatrici, la prima che fa da immagine a disparità e la seconda che fa l'inverso. La parte delle discriminatrici e dei loss fanno parte del training, che in questa fase non è necessario.

Nella terza e ultima fase vi è il caricamento vero e proprio dei pesi ottenuti durante l'addestramento, in questo passaggio il framework mette a disposizione una funzione che permette di individuare l'ultimo checkpoint e di applicare i pesi presenti in esso alle reti presenti nel grafo della sessione.

$$saver = tf.train.Saver(\max\_to\_keep = 5)$$
$$checkpoint = tf.train.latest\_checkpoint(path\_to\_checkpoint\_dir)$$
$$saver.restore(sess, checkpoint)$$

Volendo incrementare l'efficienza del sistema si potrebbe evitare la creazione manuale delle reti affidando il compito al framework, che attraverso i dati salvati durante la fase di apprendimento è in grado di ricreare il grafo.

Una volta eseguite le fasi di caricamento è possibile passare alla parte di inferenza vera e propria. In questa parte la rete deve ricevere le immagini del test-set, come per il training anche in questa fase entra in gioco la funzione run della sessione.

$$results = sess.run(operations)$$

A differenza del training, è necessario salvare i risultati su disco rigido ed è possibile utilizzare funzioni di TensorFlow e del sistema operativo. Il framework mette a disposizione alcune funzioni che permettono di codificare le immagini da formato raw a quello desiderato, png in questo caso. Codificando l'immagine è poi sufficiente utilizzare il metodo classico di scrittura dei file binari.

$$tf.image.encode\_png$$

Si è deciso anche di creare un file npy, in cui salvare tutte le disparità generate durante la fase di test, così da poterlo poi utilizzare nella fase di valutazione e confronto con Godard. Il file npy permette di salvare dei NumPy array su disco, in questo tipo di formato sono presenti tutte le informazioni necessarie a ricostruire correttamente l'array anche su architetture differenti da quella di origine.

6.6 Metriche di valutazione del modello

Oltre alla parte di test è necessario aggiungere quella di valutazione, che permette di capire quanto il sistema sia preciso nel costruire le immagini di disparità rispetto ai valori reali.

Le principali metriche che sono state utilizzate per la valutazione sono:

- Abs Rel
- Sq Rel
- RMSE
- RMSE Log

RMSE e RMSE Log

$$\sqrt{\frac{\sum_{i=0}^{n}(y_i - y_i')^2}{n}}$$

La deviazione quadratica media o l'errore quadratico medio (RMSE) è una misura utilizzata per la valutazione delle previsioni di un modello e i valori campione.

$$\sqrt{\frac{\sum_{i=0}^{n}(\log y_i - \log y_i')^2}{n}}$$

Abs Rel (Relative Absolute error)

$$\frac{1}{N}\sum_{i=0}^{n}\frac{|y_i - y_i'|}{y_i'}$$

Sq Rel (Root relative squared error)

$$\frac{1}{N}\sum_{i=0}^{n}\frac{(y_i - y_i')^2}{y_i'}$$

6.7 Confronto tra DualGAN-Godard e il modello finale

Il primo modello, formato da una parte di DualGAN e una di Godard presenta caratteristiche che potevano essere utili al fine di realizzare un sistema completamente non supervisionato. In questo modello si era voluto principalmente puntare al fatto di non dover utilizzare delle disparità target e quindi di riuscire ad eliminare il problema della loro creazione.

La sezione principale del modello è la struttura della rete, infatti essa è del tipo VGG, molto utilizzata anche in altre applicazioni. Se utilizzata nel modello originale questa struttura di rete funziona in modo corretto, è in grado di gestire il passaggio da immagine a disparità e il passaggio inverso. Invece presenta problemi nel caso in cui si vada ad inserirla in un modello differente, come DualGAN.

Il modello DualGAN, da parte sua presentava caratteristiche molto utili per questo lavoro, permettendo di rimuovere completamente le immagini target dal dataset di training e di poter sfruttare quelle generate dalla prima GAN. Inoltre, il modello era stato studiato appositamente per eseguire la trasformazione tra un dominio A e un dominio B, nel nostro caso tra il dominio delle immagini e quello delle disparità senza dover utilizzare coppie di immagini strettamente correlate. Il modello risultante univa tutte queste caratteristiche per andare a formare una struttura che potesse risolvere il problema in modo corretto.

Nel secondo modello la struttura è completamente differente, nonostante vi sia sempre la presenza di GAN.

Figura 56: Esempio di immagine della patchGAN, ogni pixel corrisponde a un quadrato 70x70 dell'immagine in input

Come per l'altro, anche questo è stato pensato per effettuare dei cambi di dominio, utilizzando però un differente approccio.

In questa versione il cambio di dominio avviene in modo più simile ad altri lavori, con la differenza che, mentre negli altri casi vi è l'utilizzo delle GAN classiche, in questo vi è l'utilizzo delle *cGAN* (conditional GAN) e delle *PatchGAN*.

Una delle principali differenze tra il primo modello e l'ultimo riguardano proprio l'uso delle PatchGAN, queste consentono al modello di gestire meglio la complessità delle informazioni. Ulteriori differenze stanno nella struttura interna della rete generatrice; nel modello finale la rete generatrice è strutturata in encoder e decoder, i quali internamente sono strutturati in livelli di convoluzione, deconvoluzione e batch normalization. La generatrice quindi è formata da 7 encoder e 7 decoder, per formare una struttura complessiva di 14 "macro-livelli", nella versione precedente invece la rete era formata sempre da macro-livelli, che però internamente erano formati da più livelli di convoluzione e deconvoluzione, che variavano con l'aumento di profondità e con l'assenza dei livelli di batch normalization.

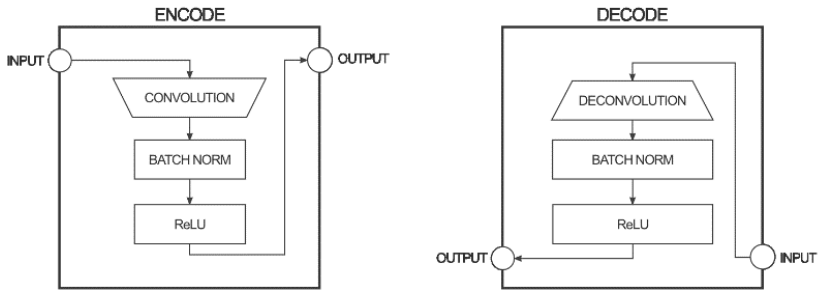


Figura 57: Macro-livelli del modello pix2pix

L'utilizzo del batch normalization, come detto in precedenza, limita lo spostamento covariante normalizzando le attivazioni di ogni livello. Questo, presumibilmente, consente a ciascun livello di apprendere una distribuzione più stabile degli input e accelerare così l'addestramento della rete. Le due reti, nonostante a livello macroscopico siano molto simili, a livello "microscopico" presentano differenze sostanziali.

Altra differenza tra i due modelli è la struttura di training, il primo modello presenta una struttura a doppia U o a due loop. Con questa struttura quindi la prima U gestisce la parte che riguarda la generazione delle mappe a partire dalle immagini, mentre la seconda U ha

il compito inverso. Le due U “interagiscono” tra loro tramite i reconstruction loss, strutturando il modello in modo da formare le due U è possibile rimuove alcuni legami tra i due domini, non vi è più necessità che nel secondo dominio vi siano immagini strettamente legate al primo.

Nel secondo modello invece non vi è la presenza di loop come quelli del primo, ma vi è un sistema in cascata di due modelli, i quali in modo “indiretto” interagisco per arrivare all’obiettivo finale.

Tra i due modelli quindi vi è una netta differenza nell’approccio di training, da una parte la scelta di slegare i due domini per semplificare la realizzazione dei dataset, dall’altra l’uso di un sistema di training più comune, che però necessita di dataset in cui vi siano delle coppie di immagini strettamente correlate. Queste due differenti strutture, nel caso del nostro task, portano ad ottenere risultati differenti, la seconda strategia porta il sistema ad una soluzione più o meno valida (sistema a singola e doppia disparità), mentre il primo non è in grado di fornire dei risultati validi, in nessuna delle sue varianti.

7 Risultati sperimentali

7.1 Modello monoculare

Da immagine a disparità

Con la sezione di test e quella di valutazione è possibile quindi iniziare la parte sperimentale e successivamente quella di confronto tra Godard e la versione finale. Il modello utilizzato per le prove è quello costituito dalle due disparità. Dalle prove precedenti si è visto come questa versione del modello sia in grado di gestire meglio la parte di generazione delle immagini.

Per riuscire ad avere un riferimento sul grado di generalizzazione della rete si è provveduto ad eseguire la rete su immagini che non ha mai visionato durante la fase di test. Utilizzando immagini nuove è possibile quindi capire se la rete riesce a gestire scene mai viste e con che grado di precisione è in grado di ricostruirle, rispetto a quelle utilizzate per la fase di training.

Per la fase di test le immagini non sono state ritagliate, ma ridimensionate a 256x512 e fornite in ingresso alla rete.

Di seguito alcuni esempi di come la rete sia in grado di ricostruire le disparità a partire dall'immagine monoculare a colori (A sinistra). Le disparità rispecchiano abbastanza fedelmente la scena in ingresso, in Figura 54 è possibile osservare come in alcuni casi la rete non è in grado di generare una disparità uniforme in alcune sezioni dell'immagine. La parte dove è più evidente è quella del muro, in cui la rete crea una profondità uniforme tra l'ombra e la sezione di albero sopra.

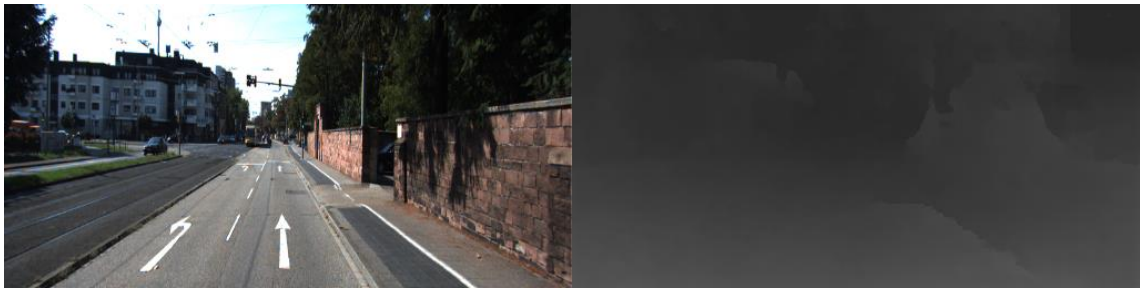


Figura 58: Ricostruzione della disparità

Qualche altro problema è riscontrabile nella disparità di *Figura 55*, dove la rete non riesce a separare la macchina dal muro, questo però è anche presente nell'algoritmo SGM, per cui il problema potrebbe essere dovuto a quello.

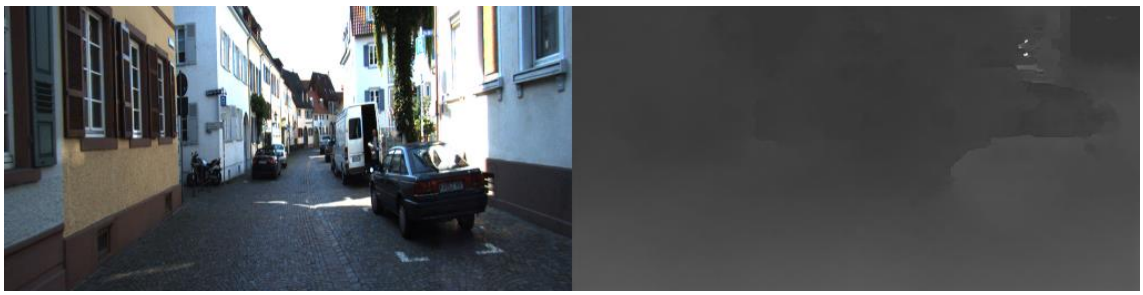


Figura 59: Ricostruzione della disparità scena città

In questa ultima immagine è possibile osservare come la rete sia in grado di gestire correttamente la presenza di ombre in determinate circostanze. In questo caso infatti il modello è riuscito a ricreare il furgoncino in modo corretto, senza venire condizionata dall'ombra come nel caso precedente.

Figura 60: Ricostruzione della disparità scena strada

Da disparità a immagine

Per quanto riguarda la generazione dell'immagine a partire dalla disparità la rete si comporta discretamente nonostante in alcuni casi vi sia ancora la presenza di artefatti.

In *Figura 59* si nota subito come la rete sia in grado di ricreare una scena basandosi esclusivamente sulle disparità, in questo caso l'immagine corrisponde a quella in *Figura 54.* Dal confronto tra le due si nota come la rete sia in grado di generare delle scene semanticamente corrette, ma che non sono la copia esatta delle originali, questo fatto permette di creare scene leggermente differenti le une dalle altre, sempre però partendo da una base comune.

Figura 61: Ricostruzione scena strada a partire dalla disparità

Il modello finale che utilizza le due disparità riesce a creare le scene in modo più corretto rispetto a quello con singola, in alcuni casi però il risultato non cambia molto. Nella figura seguente è possibile vedere come la costruzione delle vetture non sia riuscita particolarmente bene. Uno dei fattori che influenzano la qualità delle uscite è da ricercare nelle disparità, com'è possibile osservare la disparità presenta alcuni artefatti sui veicoli e altri sul contorno dell'immagine. Se si va a confrontare l'immagine generata e la relativa disparità ci si accorge che gli errori presenti nella ricostruzione sono causati anche dalla presenza di artefatti nella disparità di riferimento, questo però non è sempre vero, molte volte gli artefatti delle disparità non causano problemi durante la ricostruzione.

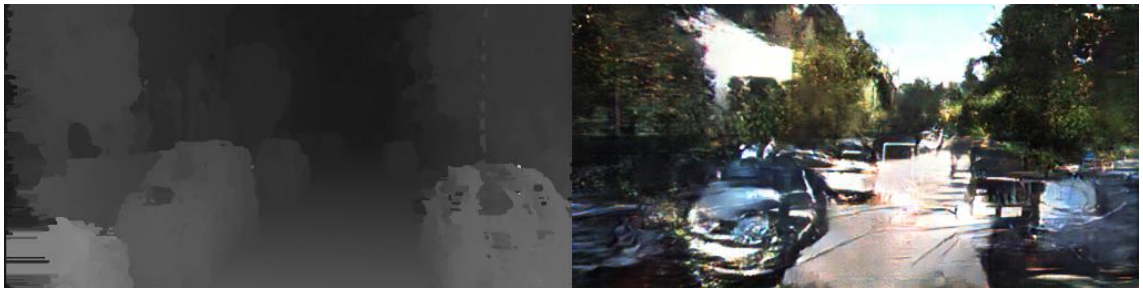


Figura 62: Ricostruzione scena periferia da disparità

In questo caso (*Figura 62*) il problema al contorno sinistro della mappa non causa problemi alla rete, che riesce a creare il bordo in modo corretto e senza errori. Inoltre, da un confronto tra le due immagini sotto, si nota subito come i problemi principali della ricostruzione si trovano sul lato dove c'è il veicolo.

Figura 63: Ricostruzione scena parco da disparità

7.2 Modello Stereo

Da immagine a disparità

Con l'introduzione delle immagini stereo il modello è in grado di creare più facilmente le mappe di disparità e con meno artefatti. Rispetto al precedente, il task inverso non ha subito molti cambiamenti, in alcune situazione la rete dovrebbe essere in grado di gestire meglio alcuni particolari della scena, che nella versione mono potevano creare problemi.

Dalle immagini in Figura si vede come le mappe risultano essere più definite e con meno artefatti. Molte delle zone in cui la rete aveva problemi risultano essere corrette o presentano meno artefatti, questo miglioramento è evidente soprattutto in quelle aree dove sono presenti veicoli e oggetti sottili. Nella versione mono, gli oggetti piccoli e i veicoli erano quelli che presentavano più problemi di tutti, dalla loro mancata compara all'errore di valutazione della disparità.

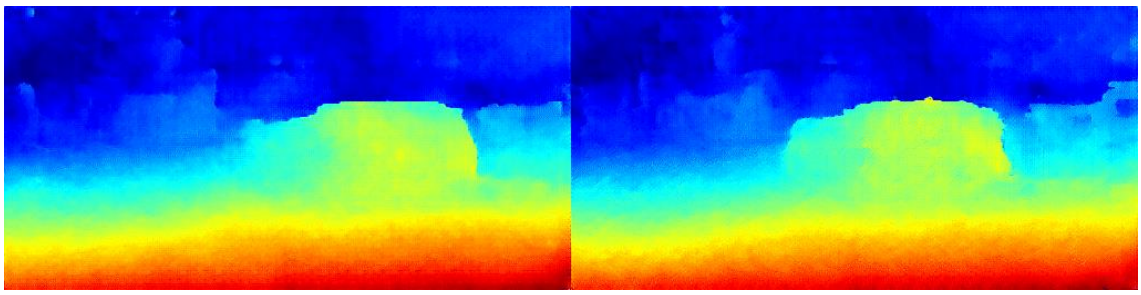


Figura 64: Mappa generata dalla rete

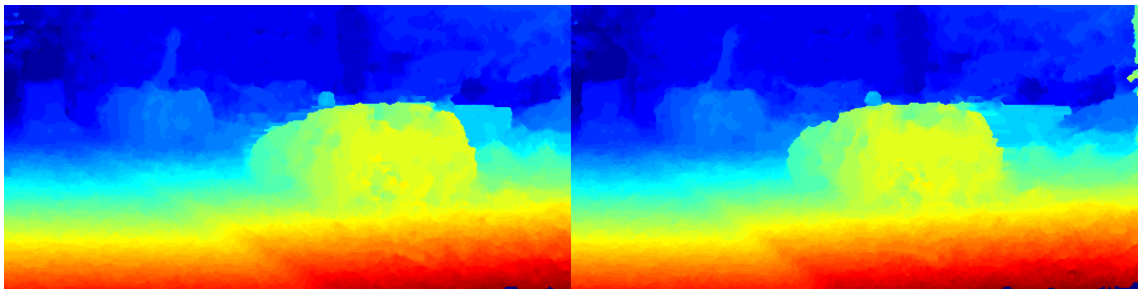


Figura 65: Mappa target

Dalle immagini sopra (*Figura 64 e 65*) si vede subito come la rete sia in grado di riprodurre i contorni in modo più fedele delle precedenti, vi è ancora la presenza di problemi, come ad esempio la parte frontale del veicolo, ma sono molto meno presenti rispetto alla versione mono.

Un altro esempio di trasformazione che non presenta particolari problemi è quella in *Figura 66.* Il palo che sorregge il segnale risulta essere meno "presente" nella disparità generata dalla rete rispetto a quella target e il muro presenta qualche problema di forma, ma sostanzialmente le distanze sono molto più corrette di prima.

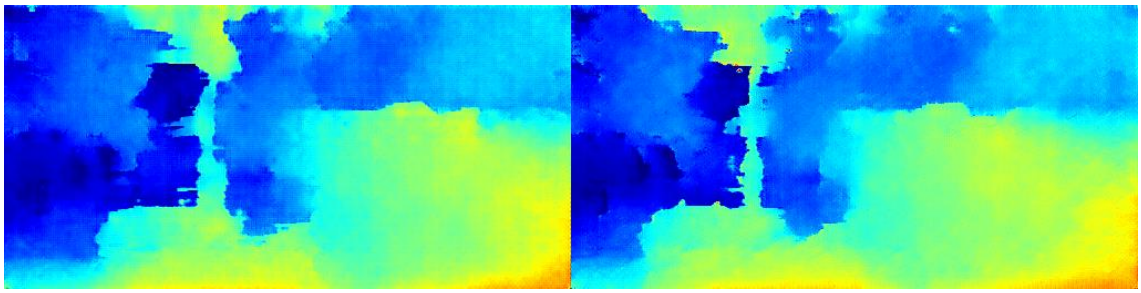


Figura 66: Mappa generata

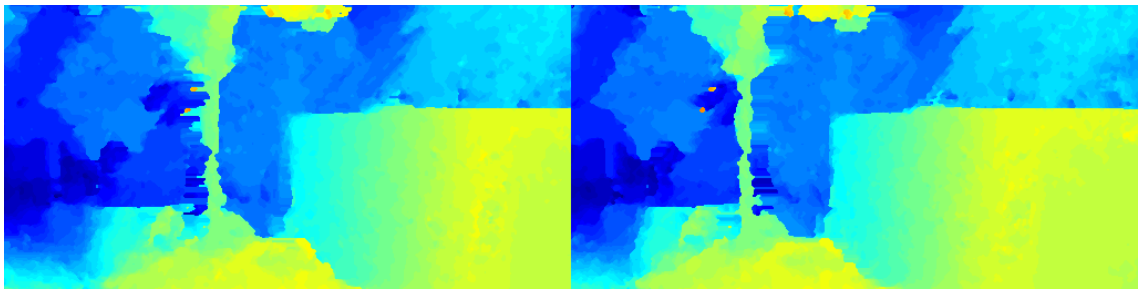


Figura 67: Mappa target

Da disparità a immagine

Per quanto riguarda il task mappa-immagine i cambiamenti rispetto alla versione mono non sono molto evidenti. Si ha un miglioramento in alcune parti delle immagini, soprattutto in quelli dove vi è la presenza di veicoli, in altri punti vi è stato qualche peggioramento, ma nel complesso il cambiamento non è stato molto grande.

Figura 68: Ricostruzione da disparità

Com'è evidente dall'immagine sopra in alcune situazioni gli artefatti non sono molto evidenti, una leggera sfuocatura a destra con alcuni punti random, ma nulla che anche nel modello mono non venisse generato in situazioni particolari.

Figura 69: Ricostruzione scena città

Nell'immagine sopra è evidente come l'uso di immagini stereo abbia portato benefici nella ricostruzione dei veicoli rispetto alla versione mono. L'introduzione delle stereo però ha introdotto alcuni problemi in altre zone, come a sinistra.

Figura 70: Ricostruzione scena auto

Un altro esempio di come la rete sia migliorata nel generare i veicoli senza che il resto peggiorasse rispetto alla versione mono. In alcuni casi la rete è migliorata a tal punto da riuscire quasi a ricreare anche le biciclette, che nelle versioni precedenti non creava e riempiva il vuoto con colori random.

Figura 71: Ricostruzione scena bicicletta

Figura 72: Ricostruzione scena edificio

CONCLUSIONI

Dalle varie prove effettuate su immagini di validazione durante il training e successivamente con quelle di test si è potuto constatare che il modello funziona egregiamente. Essendo questa una prima versione sicuramente c'è ancora margine di miglioramento, sia dal punto di vista delle immagini generate, sia dal punto di vista del training. Per quanto riguarda il training una possibile modifica è quella di renderlo non supervisionato, cosicché non vi sia la necessità delle immagini target generate tramite SGM e quindi evitando che la rete acquisisca la capacità di riprodurre anche gli artefatti presenti. Inoltre, l'introduzione della modalità non supervisionata permetterebbe di ottenere delle mappe più definite e simili a quelle di Godard. Oltre a questo tipo di modifica si può pensare di effettuare un addestramento iniziale del modello con il dataset *cityscapes* e successivamente "ritoccare" i pesi tramite il fine tuning utilizzando il dataset *kitti*. Tramite questo processo sarebbe quindi possibile "estendere" il campo di funzionamento del modello in altri scenari. Alcuni possibili test, che non si sono potuti realizzare, riguardano il comportamento della rete con immagini completamente diverse da quelle utilizzate in questo lavoro. Alcuni test potrebbero riguardare anche l'uso di mappe sintetiche, con essi si riuscirebbe a verificare se la rete è in grado di ricostruire le immagini partendo da mappe sintetiche, molto più definite delle altre e nel caso modificare il modello affinché possa utilizzarle correttamente.

BIBLIOGRAFIA

[1] Clement Godard, Oisin Mac Aodha, Gabriel J. Brostow. «Unsupervised Monocular Depth Estimation with Left-Right Consistency.»

[2] David Eigen, Christian Puhrsch, Rob Fergus. «Depth Map Prediction from a Single Image.»

[3] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer. «A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation.»

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. «Image-to-Image Translation with Conditional Adversarial Networks.»

[5] Qifeng Chen, Vladlen Koltun. «Photographic Image Synthesis with Cascaded Refinement Networks.»

[6] Zili Yi, Hao Zhang, Ping Tan, Minglun Gong. «DualGAN: Unsupervised Dual Learning for Image-to-Image Translation.»