Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria del Cinema e dei Mezzi di Comunicazione

A.a. 2021/2022

Sessione di Laurea Dicembre 2022

Creazione di dataset di immagini sintetiche per l'addestramento di reti neurali dedicate alla generazione automatica di oggetti 3D

Relatori:
Andrea Sanna
Federico Manuri

Candidato:
Ismaele Piparo

Ai miei genitori.
Che mi hanno sempre supportato
e sopportato in questi anni.

Sommario

Negli ultimi anni si è visto un generale aumento di interesse nel campo dell'intelligenza artificiale e delle reti neurali soprattutto nell'ambito della Computer Vision. Se in principio la maggior parte delle reti neurali erano progettate per svolgere compiti come il riconoscimento o la classificazione di oggetti a partire da immagini 2D, oggi l'intelligenza artificiale viene spesso usata anche per la ricostruzione di oggetti 3D a partire da una o più immagini bidimensionali dello stesso oggetto.
In particolare, con lo sviluppo di smartphone sempre più performanti che permettono l'utilizzo di applicazioni in grado di gestire la realtà aumentata e la nascita di nuovi "*ambienti social virtuali*" si stanno sempre di più sviluppando applicazioni che, mediante l'utilizzo di specifiche reti neurali, sono in grado di "ricostruire" un oggetto reale all'interno di uno spazio virtuale 3D.

Di base una rete neurale prima di essere in grado di svolgere il compito per cui è stata progettata necessita di una fase di "training" nella quale la rete viene allenata mediante l'analisi di dati, come per esempio immagini, a svolgere un determinato compito. Una corretta fase di training risulta quindi essenziale per il buon funzionamento di una rete neurale.
Per quanto riguarda le reti neurali utilizzate per la ricostruzione di oggetti 3D, queste sono allenate mediante l'utilizzo di dataset di immagini rappresentanti varie categorie di oggetti. In molti casi questi dataset non contengono immagini "reali" di oggetti ma delle immagini sintetiche ottenute tramite render di modelli 3D. Diversi studi e progetti hanno mostrato, infatti, che utilizzando un dataset di immagini bidimensionali, anche sintetiche, di un oggetto ripreso da diverse angolazioni e pose, sia possibile allenare una rete neurale in modo tale che questa sia in grado di ricostruire fedelmente un oggetto 3D a partire da una singola (o più) immagine 2D. Al momento però non esiste una regola generale per la creazione di questi dataset di immagini sintetiche, infatti, diverse reti che si occupano di ricostruzione di modelli 3D utilizzano dataset di immagini con caratteristiche diverse e questo rende difficile capire quali attributi di un'immagine sintetica risultino più o meno "utili" alla rete neurale in fase di training per avere poi una migliore ricostruzione di un oggetto 3D.

Scopo di questa tesi è dunque quello di indagare quali parametri di un'immagine sintetica, usata per il training di una rete neurale, vadano ad impattare di più sulla qualità dell'oggetto 3D ricostruito dalla rete neurale. Più in generale, dato che sia il processo di training della rete neurale che il processo di generazione di immagini sintetiche possono

essere molto onerosi in termini di risorse hardware e di tempo di calcolo, si è voluto inoltre indagare quali fossero i parametri che potessero ottimizzare al meglio le due fasi ottenendo comunque buoni risultati in termine di ricostruzione di un oggetto 3D.

In letteratura non sono state definite delle regole standard da seguire per la creazione di un dataset di immagini sintetiche da utilizzare per il training di una rete neurale. Diversi studi che fanno ampiamente uso di dataset di immagini sintetiche, infatti, tendono a creare dei propri dataset con determinate caratteristiche utili per quel determinato studio. Di conseguenza i dataset che si trovano in rete sono spesso molto eterogenei tra di loro in quanto, per esempio, alcuni utilizzano immagini a colori piuttosto che in scala di grigi oppure la risoluzione delle immagini varia da dataset a dataset rendendo difficile valutare quali parametri delle immagini vadano ad influire maggiormente sulle prestazioni di una rete neurale.
Come primo step, quindi, si è iniziato definendo una pipeline da seguire per la generazione di dataset di immagini sintetiche da utilizzare per il training. La pipeline proposta consiste in determinati passaggi e indicazioni pratiche da seguire per la creazione di un proprio dataset partendo dalla scelta dei modelli fino ad arrivare al render delle varie immagini. La pipeline è stata pensata per essere utilizzata con il software *Blender* ed in particolare, in questa tesi è stata sviluppata utilizzando le varie funzioni offerte dalla pipeline procedurale di *BlenderProc*.

Avendo definito la pipeline da seguire si è proceduto all'identificazione dei potenziali attributi di un'immagine che potessero influire sul processo di training della rete neurale. Tra i vari attributi evidenziati ci sono: la risoluzione dell'immagine, la tipologia di materiali e texture utilizzati, l'illuminazione del modello 3D e l'utilizzo di immagini a colori piuttosto che immagini in scala di grigi.

Una volta definita la pipeline e definiti quali fossero gli attributi che maggiormente potessero impattare sul training della rete neurale, si è proceduto alla scelta delle categorie di oggetti da utilizzare per la generazione dei vari dataset. Le categorie scelte sono state sedie, tavoli e armadi/librerie e per ognuna delle quali sono stati selezionati 40 modelli 3D. Avendo tutti i modelli si è dunque proceduto alla creazione dei vari dataset di immagini sintetiche su cui effettuare le valutazioni.

Lo step successivo è stato quello dell'implementazione della rete neurale. Come modello da utilizzare è stato scelto quello proposto nel lavoro *BSP-Net: Generating Compact Meshes via Binary Space Partitioning*. BSP-net è una rete neurale sviluppata in Python utilizzando il framework PyTorch che permette la ricostruzione di mesh 3D a partire da una singola immagine in input (*Single View Reconstruction*) basata sul modello di *ResNet-18*. In particolare, BSP-net lavora utilizzando come input immagini con risoluzione 224×224px in scala di grigi quindi per poter effettuare le varie prove con i vari dataset è stato necessario apportare alcune modifiche al codice fornito dagli autori.

Messa a punto la rete neurale si è passati alla fase di training ed alla successiva fase di ricostruzione e valutazione degli oggetti 3D ricostruiti. Per il training della rete neurale sono state utilizzate le immagini sintetiche dei dataset creati a partire dai modelli scelti

mentre per la ricostruzione si sono utilizzate delle immagini "reali" prese dal dataset *Pix3D*. Questo particolare dataset contiene sia dei modelli 3D di oggetti appartenenti a diverse categorie sia le corrispondenti immagini reali di tali oggetti. Sfruttando infatti il dataset Pix3D è stato possibile utilizzare le immagini reali per la ricostruzione degli oggetti e i modelli 3D come "*ground truth*" per effettuare le valutazioni sulla qualità della ricostruzione. Per valutare la fedeltà dell'oggetto ricostruito al modello originale ground truth è stata utilizzata la *Chamfer Distance* ossia una metrica di valutazione ampiamente utilizzata in letteratura per determinare le prestazioni delle reti neurali che si occupano di ricostruzione di oggetti 3D. La Chamfer Distance, infatti, è una metrica che tiene conto della distanza di ogni singolo punto appartenente ad una nuvola di punti A rispetto al corrispondente punto "più vicino" appartenente ad un'altra nuvola di punti B. Quindi in generale più piccolo è il valore della Chamfer Distance più fedele sarà l'oggetto 3D ricostruito al ground truth originale. Calcolando il valore della Chamfer Distance tra i modelli ricostruiti e i modelli ground truth è stato quindi possibile avere una valutazione oggettiva della qualità della ricostruzione.

La fase di training della rete neurale e la successiva fase di ricostruzione dei modelli 3D è stata ripetuta per tutti i dataset generati in base ai vari attributi di cui si volevano verificare gli effetti.

Una volta avuti tutti i dati si è passati alla fase di analisi dei risultati.
In particolare, dalle valutazioni fatte è emerso che immagini a risoluzione maggiore, entro un certo limite, permettono alla rete neurale di ricostruire un oggetto 3D in maniera più accurata. Di contro però si è osservato che sia il tempo impiegato dalla rete per effettuare il training che il tempo necessario alla generazione del dataset cresce al crescere della risoluzione delle immagini.
Per quanto riguarda i materiali invece si è osservato che materiali con texture "semplici" tendono a dare risultati migliori rispetto a materiali aventi texture che presentano dei particolari pattern.
Mentre, infine, per quanto riguarda le luci si è osservato che una illuminazione omogenea del modello 3D porta ad avere migliori risultati in fase di ricostruzione.

Le analisi fatte in questa tesi hanno mostrato che una accurata scelta degli attributi di un'immagine sintetica utilizzata per il training di una rete neurale potrebbe influire in modo positivo sul buon funzionamento della rete.
Inoltre, conoscendo questi dati, chiunque abbia la necessità di allenare la propria rete neurale utilizzando immagini sintetiche sarebbe molto avvantaggiato nella creazione del proprio dataset di immagini da usare per il training in quanto saprebbe su cosa è possibile intervenire per ottimizzare al meglio le varie fasi di lavoro evitando così spreco di tempo e risorse utili.

Infine, utilizzando come base di partenza la pipeline proposta in questa tesi si potrebbe cominciare a pensare a delle regole generali, che tenendo conto anche dei risultati ottenuti, siano utili per la creazione di dataset di immagini da usare per il training di una rete neurale evitando così di avere così decine di dataset ognuno dei quali utilizza differenti tipologie di immagini sintetiche.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto

Al giorno d'oggi applicazioni che sfruttano l'intelligenza artificiale sono largamente utilizzate in svariati ambiti, dal semplice riconoscimento vocale sino alla gestione di compiti più complessi come la guida autonoma. Tutto questo è stato reso possibile grazie al rapido incremento della capacità di calcolo dei moderni computer il quale ha permesso di far svolgere molti più studi e ricerche nel campo del machine learning rendendo l'intelligenza artificiale sempre più accessibile a tutti e non solo appannaggio di grandi aziende che potevano disporre di enormi risorse hardware. Fino a qualche anno fa, infatti, era impensabile di poter implementare da zero un modello che sfruttasse una rete neurale per un qualsiasi scopo; oggi invece grazie alle nuove tecnologie e all'ampia offerta di librerie pubbliche come *PyTorch* [27], *TensorFlow* [36], *Keras* [17] ed altre ancora, chiunque con un minimo di conoscenza nel campo dell'informatica è in grado di sviluppare la propria rete neurale personale.

Esistono oggi reti neurali dedicate al riconoscimento della voce, al riconoscimento del volto, alla classificazione di oggetti, alla guida autonoma, alla creazione di modelli previsionali, alla gestione degli NPC[1] nei videogiochi, alla gestione di attori e set "virtuali" nel mondo del cinema, alla creazione di musica, e molte altre ancora in contesti sempre diversi.

In particolare, negli ultimi anni con l'avvento di smartphone sempre più performanti che permetto di utilizzare applicazioni per la realtà aumentata e lo sviluppo e la nascita dei nuovi "*metaversi*" e di altri spazi virtuali, si stanno sviluppando sempre più applicazioni che permettono di "trasportare" un oggetto reale in un ambiente virtuale. Per fare ciò si ricorre sempre più spesso all'utilizzo di particolari reti neurali in grado di ricostruire un oggetto 3D a partire da una singola immagine 2D. Questa tecnica è detta *Single View Reconstruction* – ricostruzione da una singola immagine.

[1] NPC sta *Non Playeble Character*. Nell'ambito dei videogiochi indica un personaggio che non è sotto il controllo diretto di un giocatore ma che è controllato direttamente dal computer che ne guida l'iterazione con il giocatore mediante l'uso di determinati algoritmi o, come spesso accade per i nuovi videogiochi, anche mediante l'utilizzo dell'intelligenza artificiale.

Di base però una qualsiasi rete neurale prima di essere pronta ad eseguire il compito per cui è stata progettata necessita di una fase di *training*. Nella fase di training, infatti, la rete viene allenata mediante l'uso di dati quali possono essere registrazioni vocali per le reti che si occupano di riconoscimento vocale oppure immagini come nel caso delle reti neurali dedicate alla ricostruzione di oggetti 3D.
La fase di training, dunque, risulta essere molto importante per il corretto funzionamento di una rete neurale come anche la quantità di dati utilizzati per allenare la rete.

Le reti neurali dedicate alla ricostruzione di oggetti 3D, in fase di training, vengono allenate con dataset di immagini che rappresentano vari oggetti ripresi da diverse angolazioni e diverse pose. Così facendo la rete sarà poi in grado di ricostruire, quasi fedelmente, un oggetto reale in un mondo virtuale.
In generale, per avere a disposizione grandi quantità di dati su cui fare il training, i dataset usati per l'allenamento di reti neurali dedicate alla ricostruzione di oggetti 3D non contengono quasi mai immagini "reali" di un oggetto ma delle immagini sintetiche ottenute tramite render[2] di modelli 3D di centinaia di oggetti, generati mediante l'uso di appositi software come *Blender* [6], *Maya* [20], *Rhinoceros* [29] ed altri ancora.

Al momento però non esistono dei criteri standard per la creazione di dataset sintetici da utilizzare per il training di reti neurali. Diverse reti che si occupano di ricostruzione di oggetti 3D, infatti, utilizzano spesso dei propri dataset di immagini sintetiche con caratteristiche differenti da quelli usati da altre reti neurali. Per esempio, alcune reti effettuano il training con immagini a risoluzione 128×128px altre con risoluzioni 224×244px; alcuni dataset contengo immagini in scala di grigi mentre altri immagini a colori. Quindi, dato che i vari dataset sono molto "eterogenei" tra di loro non si riesce a comprendere effettivamente quali parametri delle immagini risultino poi essere più o meno impattanti in fase di training di una rete neurale andando, di conseguenza, ad incidere sul buon funzionamento della rete stessa.

1.2 Scopo della tesi

Lo scopo principale della tesi è quello di indagare quali parametri di un'immagine 2D sintetica, usata per il training di una rete neurale dedicata alla ricostruzione di mesh 3D, vadano ad impattare di più sulla qualità del risultato finale.
Più in generale, dato che sia il processo di generazione di un dataset di immagini sintetiche a partire da un modello 3D tramite render, che il processo di training di una rete neurale possono essere molto onerosi in termini di risorse hardware ma anche di tempo di calcolo, si è cercato di capire quali fossero i parametri che meglio ottimizzino le due fasi facendo così risparmiare tempo e risorse, ottenendo comunque buoni risultati in fase di ricostruzione.

[2]Nel campo della Computer Grafica con il termine *render* si fa riferimento alla generazione, mediante l'uso di specifici algoritmi, di un'immagine sintetica a partire da un modello tridimensionale di un oggetto realizzato utilizzando dei software di modellazione 3D.

Per fare ciò, come primo step, è stata definita una pipeline per la generazione di immagini sintetiche che fosse utile per la creazione dei vari dataset di immagini da utilizzare per il training di una rete neurale.
Definita la pipeline si è proceduto alla raccolta dei modelli 3D riguardanti le varie "classi", ossia le categorie di oggetti scelti da utilizzare per le varie prove, ed in seguito si è provveduto alla creazione di tutti gli script necessari per automatizzare il processo di generazione dei vari dataset con cui effettuare i vari esperimenti.

Come secondo step, si è cercato di capire quali fossero i parametri che maggiormente potessero impattare sul processo di training di una rete neurale. Come punto di partenza si è verificato se in letteratura fossero presenti degli studi o delle ricerche specifiche riguardanti la generazione di dataset di immagini sintetiche da utilizzare per il training di reti neurali dedicate alla ricostruzione di oggetti 3D. Ad oggi però non si trova in letteratura nessuno studio di questo genere, ma solamente alcuni riguardanti più il campo della classificazione o l'identificazione di oggetti [30][34].
Non essendoci dunque dei riferimenti puntuali da cui trarre spunto, una prima fase degli esperimenti ha riguardato l'impatto che le dimensioni delle immagini, usate nella fase di training, hanno sul risultato finale della ricostruzione e come la risoluzione di tali immagini vada ad influenzare i risultati della ricostruzione.
In secondo luogo, si è proceduto ad analizzare più nel dettaglio le singole immagini per capire come i materiali e le texture degli oggetti, le luci, la posizione della camera, l'utilizzo di immagini a colori rispetto ad immagini in scala di grigio, influissero sulla buona riuscita della ricostruzione.

Infine, sono stati monitorati anche i tempi necessari alla rete per completare la fase di training in relazione alle varie prove effettuate ed i tempi di render per la generazione delle immagini dei vari dataset utilizzati per allenare la rete. Così facendo si è cercato di avere un quadro il più generale possibile per comprendere dove si potrebbero effettuare delle ottimizzazioni.

1.3 Possibili applicazioni

Grazie al continuo sviluppo di applicazioni grado di fondere elementi reali con elementi presenti in ambienti virtuali, l'utilizzo di reti neurali dedicate alla ricostruzione di oggetti 3D potrebbe estendersi a qualsiasi campo in cui sia necessario avere delle copie virtuali di oggetti reali. L'utilizzo di queste reti neurali permette infatti di avere in maniera istantanea un modello 3D di un qualsiasi oggetto reale a partire semplicemente da una singola fotografia evitando così di dover ogni volta modellare da zero oggetti anche molto semplici.

Avendo una panoramica dei vari parametri che vanno ad influire maggiormente il processo di training di queste reti neurali e la conseguente buona riuscita della ricostruzione di un oggetto 3D si vuole cercare di ampliare la letteratura sul tema dimostrando l'effettiva importanza di una accurata generazione di dataset di immagini sintetiche da usare per il

training di un modello di rete neurale.

Inoltre, conoscendo questi dati, chiunque avesse la necessita di allenare la propria rete neurale utilizzando immagini sinetiche sarebbe molto avvantaggiato nella creazione del proprio dataset di immagini da usare per il training potendo scegliere su cosa intervenire per ottimizzare al meglio le varie fasi di lavoro evitando così spreco di tempo e risorse utili.

Infine, sfruttando come base di partenza la pipeline proposta in questa tesi si potrebbe cominciare a pensare a delle regole generali, che tenendo conto anche dei risultati ottenuti in questa tesi, si possano poi applicare per la creazione di dataset di immagini sintetiche da usare per il training di una rete neurale evitando in questo modo di avere decine di dataset ognuno dei quali utilizza differenti tipologie di immagini.

Capitolo 2

Stato dell'arte

2.1 Reti neurali per la ricostruzione di oggetti 3D

Tra le varie tipologie di reti neurali, negli ultimi anni, quelle dedicate alla ricostruzione di oggetti 3D sono sempre più largamente utilizzate in vari campi con risultati spesso sorprendenti.

Fino a poco tempo fa per avere una copia virtuale di un determinato oggetto reale bisognava procedere attraverso varie tecniche come, ad esempio, la scansione dell'oggetto tramite appositi sensori come il *LiDAR*[1] [21][19]; oppure attraverso l'acquisizione di un numero considerevole di immagini che riprendessero un determinato oggetto da diverse angolazioni in modo tale da poter poi estrarre da queste immagini, tramite appositi algoritmi di fotogrammetria, la forma dell'oggetto [24].
In entrambi i casi, comunque, il modello generato deve necessariamente subire poi un processo di retopology[2] per semplificare la mesh ottenuta in modo tale da renderla più pulita e più facile da lavorare.
Se il lavoro in fase di scansione o acquisizione delle immagini viene effettuato seguendo determinati accorgimenti il modello virtuale ottenuto risulta essere praticamente identico a quello reale, tuttavia, il costo da pagare in termini di tempo e risorse hardware risulta essere in molti casi davvero troppo alto.
In generale, infatti, i programmi che permetto di ottenere modelli 3D a partire da una "scansione" di un oggetto reale tendono ad eseguire molti calcoli impiegando di conseguenza molto tempo per la creazione del modello virtuale. Anche la fase di retopology può richiedere un tempo abbastanza lungo perché in questa fase, nella maggior parte dei casi, il lavoro è spesso eseguito manualmente andando ad intervenire anche sui singoli vertici di

[1] LiDAR sta per *Light Detection and Ranging* ed è una tecnologia che sfrutta la luce di un laser per determinare la distanza di un oggetto. Mediante l'utilizzo di appositi sensori, grazie a questa tecnologia, è possibile effettuare una "scansione continua" di un oggetto o di un intero ambiente ottenendo tutte le informazioni necessarie per la creazione di una copia virtuale di quell'oggetto o ambiente.

[2] La *retopology* è un processo della modellazione 3D in cui mesh molto complesse vengono semplificate e rese più pulite riducendo il numero di vertici mantenendo comunque la forma originale del modello.

una mesh.

Le reti neurali dedicate alla ricostruzione di oggetti 3D risolvono gran parte di questi problemi in quanto riescono, in maniera del tutto autonoma ed in poco tempo, a ricostruire un determinato oggetto a partire da una singola immagine (o più immagini, ma sempre un numero ragionevolmente piccolo rispetto alla fotogrammetria).
Attualmente esistono diverse tipologie di reti neurali dedicate alla ricostruzione di oggetti 3D. Non esistendo però uno standard ben definito ognuna di queste reti utilizza un determinato approccio per la gestione della ricostruzione.
In linea di massima una delle differenze più evidenti tra le varie tipologie di reti neurali utilizzate per la ricostruzione di oggetti 3D sta nel tipo di output fornito. Non tutte le reti neurali, infatti, ricostruiscono una mesh 3D ma molto spesso solo una nuvola di punti - *point cloud* o semplicemente l'oggetto in formato *voxel*, ossia la controparte tridimensionale del pixel. Un esempio dei diversi formati è mostrato in figura 2.1.

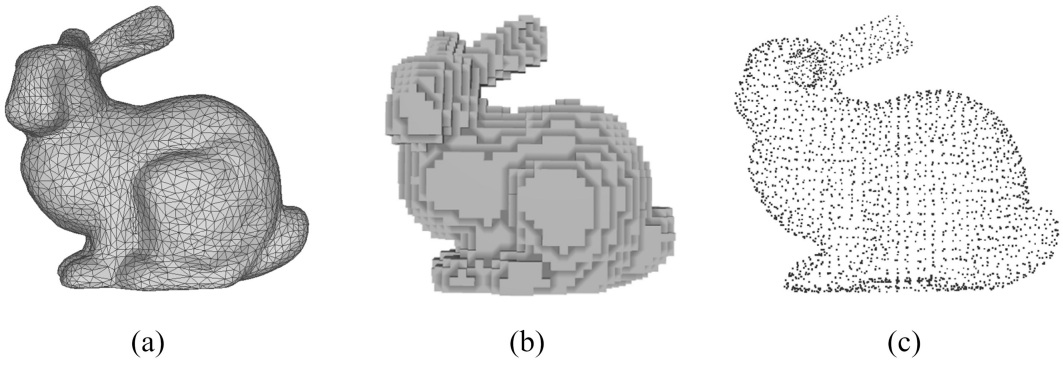

Figura 2.1: Rappresentazione di un modello 3D come: (a) mesh poligonale, (b) voxel e (c) nuvola di punti

Un'altra differenza che si può riscontrare tra le varie tipologie di reti neurali sta nel loro approccio alla ricostruzione.
Ci sono ad esempio reti che ricostruiscono un oggetto 3D a partire da una forma primitiva di base (spesso una sfera di raggio unitario o un ellisse). Tali reti estraggono delle feature da una immagine di input e le utilizzano per "modellare" la forma primitiva. Questa subisce vari processi di "mesh deformation" in modo tale da farla somigliare sempre più fedelmente all'oggetto che si vuole ricostruire (vedi figura 2.2).
Molte di queste reti però hanno delle limitazioni in quanto non sono in grado ricostruire fedelmente mesh di oggetti che presentino concavità (*Pixel2Mesh* [15], *TMNet* [14]).

Altre reti invece, come BSP-net [39], non utilizzano forme primitive ma eseguono una doppia fase di training (figura 2.3): nella prima fase la rete è allenata tramite dei modelli 3D di diverse classi di oggetti in formato voxel. Da questa fase si ottengono dei "predittori" che serviranno alla rete per consentire la ricostruzione 3D dei vari oggetti.

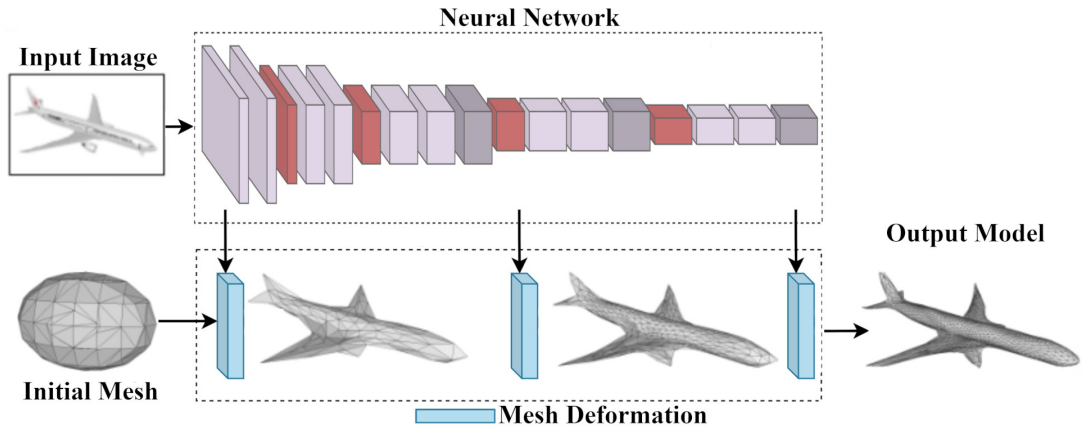

Figura 2.2: Schema di una rete neurale che sfrutta una forma primitiva per la ricostruzione di oggetti 3D

Nella seconda fase, il training viene effettuato sia con delle immagini dei vari oggetti sia con i "predittori" ottenuti dalla fase precedente.
Così facendo la rete imparerà ad associare una determinata immagine ad un particolare modello consentendo così la ricostruzione a partire da una singola immagine.

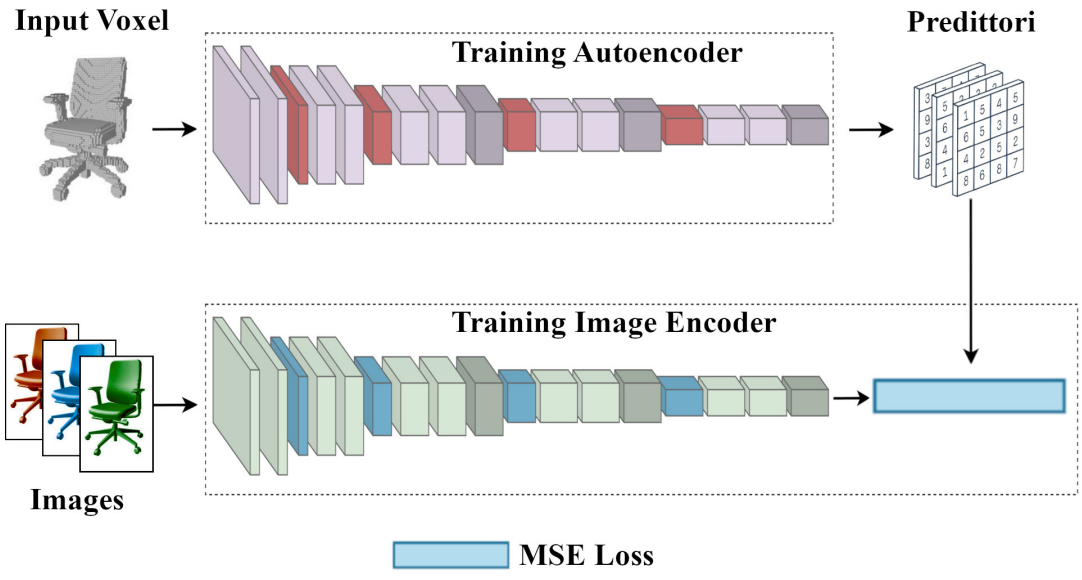

Figura 2.3: Schema di funzionamento del processo di training della rete neurale BSP-net

Altre reti ancora utilizzano in fase di training delle immagini in formato RGB-D[3] dalle quali riescono poi a ricavare la forma dell'oggetto da ricostruire.

2.2 Dataset di immagini

Una cosa in comune a tutte le reti dedicate alla ricostruzione di modelli 3D è che tutte queste, per la fase di training, utilizzano dei dataset di immagini di varie classi di oggetti. Anche in questo caso però reti differenti utilizzano tipologie di immagini differenti. Le immagini a colori in formato RGB sono le più comuni ma non di rado vengono usati dataset di immagini in scala di grigi (un solo canale L[4]) e come detto prima anche formati di immagini RGB-D (D = depth map) sono spesso utilizzate. In figura 2.4 sono mostrate le differenti tipologie di immagini.

Oltre alla tipologia di immagine anche la risoluzione di queste risulta essere molto variabile da rete a rete. Questo, in primo luogo, perché ogni rete neurale è progettata per lavorare al meglio con una determinate risoluzione dell'immagine. Non essendoci uno standard o delle regole uniche, chi progetta la propria rete neurale tende a scegliere determinati parametri che nel proprio caso risultino ottimali. Per esempio, usare immagini a bassa risoluzione consente di risparmiare molto spazio di archiviazione specialmente se si devono gestire grandi quantità di dati; di contro utilizzando immagini a bassa risoluzione si potrebbero perdere dettagli importanti utili durante il processo di training della rete neurale (figura 2.5). In particolare, alcune reti come BSP-net vincolano l'utilizzatore ad usare solamente immagini di risoluzione 128×128px, altre invece consentono (entro certi limiti) di utilizzare immagini con risoluzioni differenti.

In generale però tutti questi dataset non contengo quasi mai immagini "reali" di oggetti ma, quasi sempre, solo immagini sintetiche realizzate a partire da render, tramite l'utilizzo di appositi software, di modelli 3D di vari oggetti.
Diversi studi infatti hanno dimostrato la validità dell'utilizzo di dataset di immagine sintetiche per allenare modelli di reti neurali in grado poi di performare bene anche ricevendo in input immagini reali [13][11].
Grazie a questi risultati non vi è più la necessità ricercare grandi quantità di immagini di determinati oggetti da utilizzare poi per il training della rete, ma basta semplicemente avere un solo modello 3D di un oggetto reale da cui poi verranno create tutte le decine, se non centinaia, di immagini sintetiche necessarie.
I dataset di immagini sintetiche non sono però utilizzati solamente per il training di reti neurali ma trovano applicazione anche in altri ambiti della Computer Vision, come l'automotive, la realtà virtuale, la mixed reality, l'augmented reality ed altri campi ancora.

[3]Un'immagine RGB-D contiene al suo interno, oltre le tre informazioni relative al colore (RGB), anche delle informazioni riguardo la profondità (D sta per *depth*) della scena ripresa.

[4]La lettera L sta ad indicare che l'immagine contiene al suo interno un solo canale con le informazioni relative alla luminaza, ossia all'intensità della luce, dell'immagine.

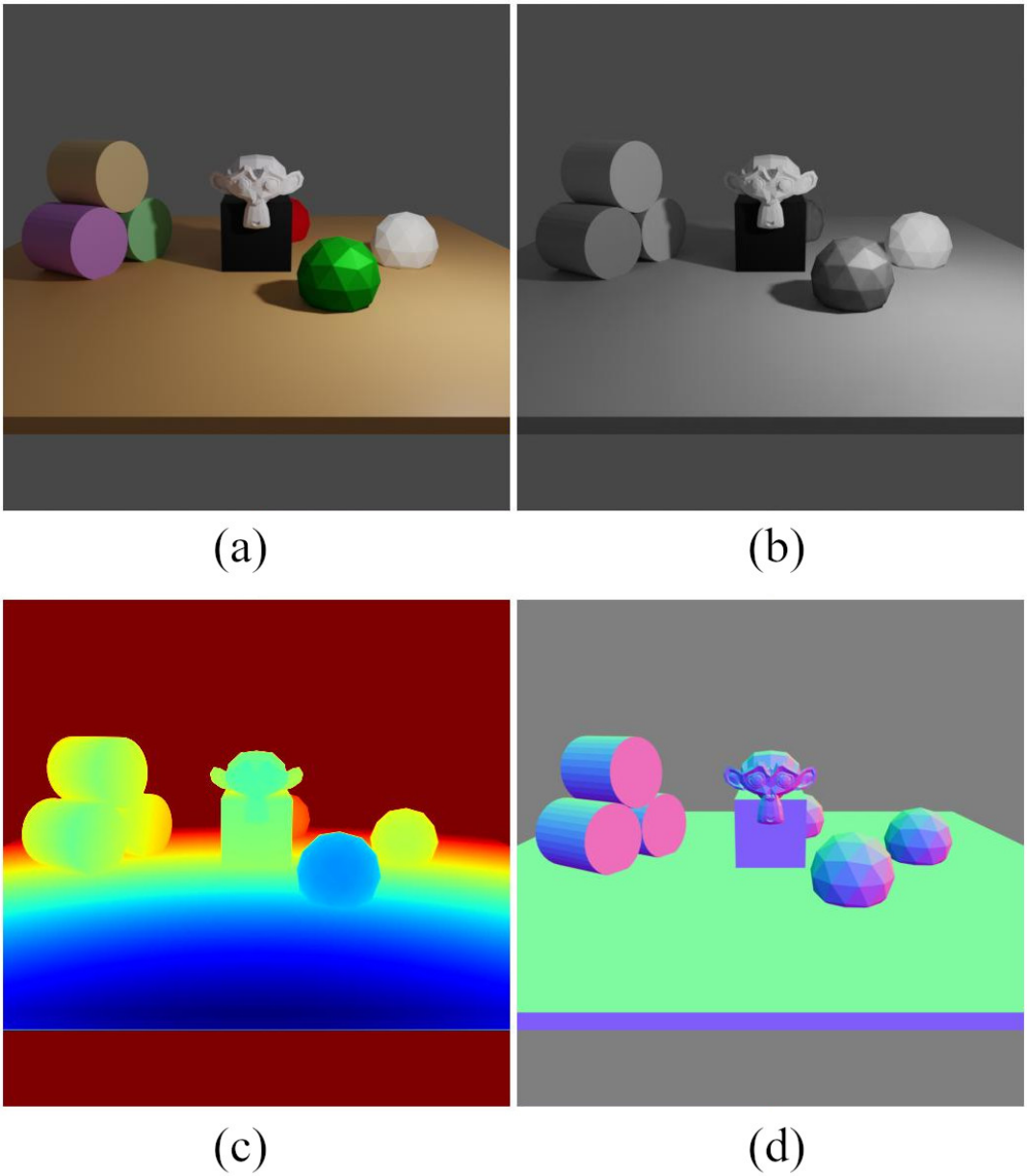

Figura 2.4: (a) Immagine a colori RGB, (b) immagine in scala di grigi L, (c) Depth Map, (d) Normal Map

Non essendoci, come detto in precedenza, delle regole fisse per la creazione dei dataset di immagini sintetiche molti di essi sono "custom" ovvero realizzati su misura dagli autori di una ricerca per dimostrare la validità di un determinato algoritmo di image processing o modello di rete neurale proposto, altri invece sono creati per essere utilizzati liberamente da altri ricercatori per i loro studi.

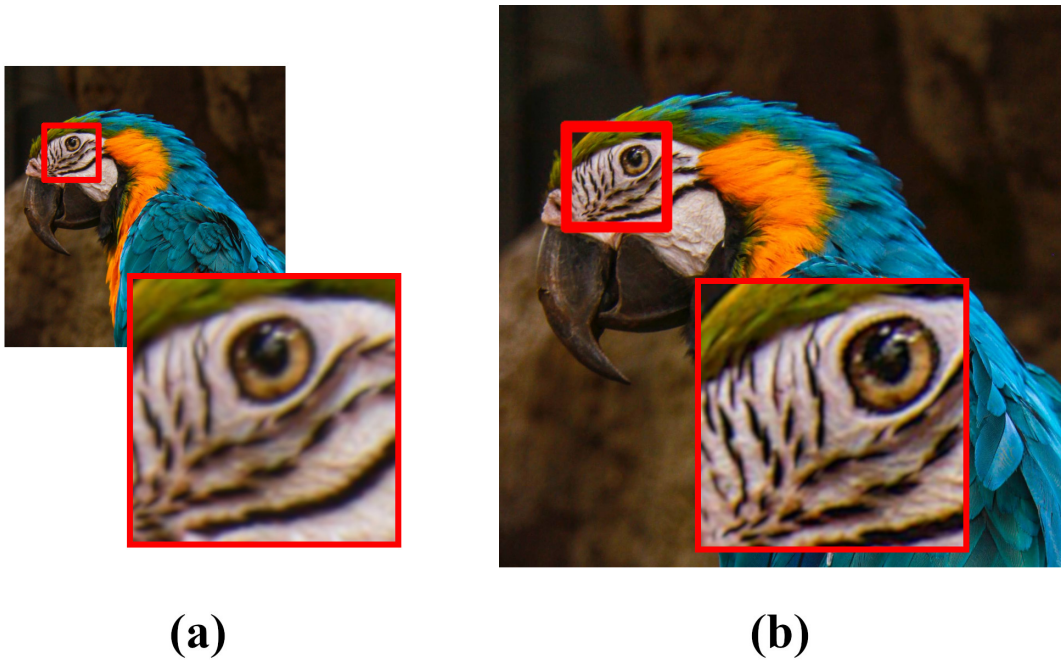

Figura 2.5: (a) Livello di dattaglio di un'immagine con risoluzione 512×512 pixel e (b) una con risoluzione 1024×1024 pixel

Spesso i dataset di immagini sintetiche vengono creati a partire da dataset di modelli 3D pubblici (nella maggior parte dei casi comprendono principalmente modelli realizzati tramite software CAD[5]). I modelli 3D pubblici, il più delle volte, sono completi anche delle varie texture e materiali semplificando di molto il processo di creazione del dataset in quanto basterà solamente eseguire i render dei modelli. Se le texture non sono presenti o se si volesse avere una variazione rispetto alle texture originali, vengono in soccorso altri dataset contenenti centinaia di texture che possono essere applicate ai vari modelli 3D prima di essere renderizzati. In entrambi i casi la generazione di dataset di immagini sintetiche risulta essere abbastanza semplice in quanto i modelli 3D "pubblici" sono spesso "rifiniti" e già normalizzati su dimensioni standard evitando cosi di dover intervenire manualmente su ogni singolo modello ma, tramite dei semplici script, è possibile gestire tutta la fase di render in maniera automatizzata.

Un altro metodo per la generazione di dataset di immagini sintetiche è quello di creare in maniera autonoma i propri modelli con i propri materiali e texture.
In generale questo secondo metodo viene utilizzato se si ha bisogno di modelli di oggetti

[5]CAD sta per *computer-aided design* e fa riferimento a tutti quei programmi di grafica 2D o 3D a supporto della progettazione di oggetti o strutture soprattutto in ambito edilizio e di design. A differenza dei classici software di modellazione 3D i programmi CAD offrono un approccio più ingegneristico alla creazione di oggetti 3D in quanto questi oggetti, nella maggior parte dei casi dovranno poi essere effettivamente realizzati.

molto specifici o con determinate caratteristiche. Il lavoro necessario per la realizzazione di un dataset in questo modo risulta però essere molto oneroso. Questo perchè, in particolare, per effettuare il training di una rete neurale da zero si ha bisogno di molti dati e quindi di molti modelli 3D da realizzare, texturizzare ed infine normalizzare per poter poi eseguire tutti i render necessari.
Un aiuto in questo caso viene dato dall'aumento, dovuto alla crescente richiesta di modelli 3D di qualsiasi tipo che si è vista negli ultimi anni, della presenza di molti siti online che offrono la possibilità di scaricare, anche gratuitamente, qualsiasi modello 3D di qualsiasi oggetto.

Infine, si possono anche creare dei dataset di immagini "unendo" più dataset provenienti da fonti diverse o espandere dataset pubblici con immagini dei propri modelli 3D, con l'unica accortezza che le immagini siano comunque omogenee evitando, almeno che non sia voluto, di avere, per esempio, immagini con risoluzioni differenti.

I principali datatset pubblici sono:

- **Shapenet** [38]: dataset diviso in due subset:
  - ShapeNetCore: sottoinsieme ShapeNet completo anche di singoli modelli 3D. Copre 55 categorie di oggetti comuni con circa 51.300 modelli 3D.
  - ShapeNetSem è un sottoinsieme più piccolo di ShapeNet composto da 12.000 modelli distribuiti su un insieme più ampio di 270 categorie.
- **Pix3D** [10]: dataset che oltre ad offrire circa 400 modelli 3D di vari oggetti divisi in 9 categorie mette a disposizione, per ogni modello, delle immagini reali dell'oggetto che rappresenta.
- **ImageNet** [9]: dataset di sole immagini sintetiche utilizzato soprattuto per il training di reti neurali dedicate alla classificazione di oggetti. Il suo subset principale comprende 1000 classi di oggetti e contiene 1.281.167 immagini di addestramento, 50.000 immagini di convalida e 100.000 immagini di test.
- **AmbientCG** [3]: dataset di textures accessibile utilizzando delle funzioni offerte dalla pipeline di BlenderProc [16]. Disponibile anche come sito web online, contiene un migliaio di textures relative a diversi materiali.
- **Haven** [12]: dataset che contiene modelli 3D, texture e immagini HDRI[6]. Disponibile anche questo utilizzando la pipeline di BlenderProc [16] oppure tramite sito web. Contiene circa 160 modelli 3D di oggetti, 600 immagini HDRI e più di 250 texture.

[6]HDRI sta per *High Dynamic Range Image* ed è una tecnica fotografica utilizzata per ottenere immagini che contengono molti più dettagli rispetto a una fotografia in cui la gamma dinamica, ovvero l'intervallo tra le aree visibili più chiare e quelle più scure, è limitata e priva di dettagli nelle ombre o nelle alte luci. Nel campo della modellazione 3D spesso vengono utilizzate immagini HDRI a 360° di paesaggi o determinati luoghi dalle quali, tramite appositi algoritmi, vengono estratte tutte le informazioni relative le luci, le ombre ed i colori per ottenere poi una illuminazione della scena 3D molto accurata e realistica.

Mentre i principali siti e servizi che permetto di scaricare modelli 3D sono:

- **BlenderKit** [7]: commuty online che offre circa 11,500 modelli 3D, materiali e texture. È disponibile come add-on[7] di Blender [6] e consente di cercare e poi scaricare modelli, materiali e texture direttamente in Blender.
- **Turbosquid** [37]: sito online che offre migliaia di modelli 3D sia free che a pagamento. I modelli sono disponibili in diversi formati come file *.blend*, *.obj*, *.fbx*, ed altri ancora.
- **3D warehouse** [2]: è un sito web dove i modellatori possono caricare, scaricare e condividere modelli 3D. Anche qui è possibile sfogliare un catalogo di migliaia di modelli 3D disponibili sia gratuitamente che a pagamento.
- **Sketchfab** [33]: è un sito web dove è possibile pubblicare, condividere, vendere o acquistare modelli 3D anche animati. Fornisce inoltre un "visualizzatore 3D" che consente agli utenti di poter visualizzare in maniera interattiva qualsiasi modello presente sul sito direttamente dal proprio browser o anche attraverso un visore per realtà virtuale.
- **Poliigon** [25]: sito web dal quale è possibile scaricare, anche gratuitamente, migliaia di modelli 3D appartenenti a svariate categorie, texture e materiali, immagini HDRI e persino "pennelli" da utlizzare per la creazione di modelli 3D attraverso la tecnica dello sculpitng.

2.3 Altri studi

Nell'ambito specifico della generazione di dataset di immagini sintetiche da utilizzare per il training di una rete neurale, o di quali siano i parametri di un'immagine che possano impattare maggiormente sul funzionamento di una rete neurale, non ci sono grossi esempi in letteratura. In generale ogni studio o ricerca che aveva la necessità di utilizzare un dataset di immagini sintetiche o lo creava da zero o sfruttava quelli già esistenti che però nella maggior parte dei casi erano stati creati per altri scopi. Nel caso del training di reti neurali però, così facendo, si rischia di lavorare con dei dataset contenenti immagini che nel migliore dei casi contengono molte più "informazioni" di quelle necessarie alla rete nella fase di training o, nel peggiore dei casi, che non ne contengono abbastanza.

Gli unici studi effettuati in questo senso, riguardanti il training delle reti neurali, fanno riferimento solamente modelli neurali dedicati alla classificazione e identificazione di determinati elementi all'interno di un'immagine (in particolare per reti neurali usate in ambito medico [30][34] per l'individuazione e la diagnosi di alcune patologie). Tali studi si concentrano solamente sulla risoluzione delle immagini arrivando alla conclusione che in

[7] In Blender un *add-on* è un componente aggiuntivo, sotto forma di script, che consente di estendere delle funzionalità del software o aggiungerne di altre.

linea generale, e comunque guardando sempre a come la rete neurale è stata progettata, aumentando la risoluzione delle immagini utilizzate in fase di training i risultati tendono ad essere migliori entro un certo limite, oltre il quale il "guadagno" tende ad essere molto piccolo rispetto alle risorse utilizzate [32][18].

Capitolo 3

Tecnologie utilizzate

3.1 BSP-net

Il modello di rete neurale utilizzato come riferimento in questa ricerca è stato quello proposto nel lavoro *BSP-Net: Generating Compact Meshes via Binary Space Partitioning* [39]. BSP-Net è una rete neurale che, a differenza di altre reti che ricostruiscono un oggetto a partire da una "forma primitiva" di base, permette la ricostruzione di mesh 3D sfruttando il concetto di *Binary Space Partition* (BSP) per facilitare il processo di apprendimento. La Binary Space Partition è un metodo per il partizionamento dello spazio che consiste nel suddividere ricorsivamente uno spazio euclideo in due insiemi convessi utilizzando gli iperpiani come partizioni. Sfruttando questa scomposizione ricorsiva, BSP-Net è quindi in grado di apprendere e poi rappresentare la mesh di un oggetto.
BSP-Net permette anche la *Single View Reconstruction* (SVR) ossia la ricostruzione di una mesh a partire solamente da una singola immagine di un oggetto.

Il modello di base su cui è costruita BSP-net è quello delle reti neurali convoluzionali (CNN) ed in particolare BSP-net è organizzata secondo il modello standard di *ResNet-18* [35].

BSP-net è rilasciata dagli autori dello studio in quattro diverse implementazioni, ognuna delle quali utilizza librerie differenti tra le quali:

- *TensorFlow v1.15* [36]
- *TensorFlow v2.0* [36]
- *PyTorch v1.2* [27]

Quella usata in questa ricerca sfrutta le librerie di PyTorch.

3.1.1 CNN

Le reti neurali convoluzionali (CNN o ConvNet) sono una tipologia di reti neurali artificiali utilizzate soprattutto per compiti quali l'analisi di immagini visive (figura 3.1).

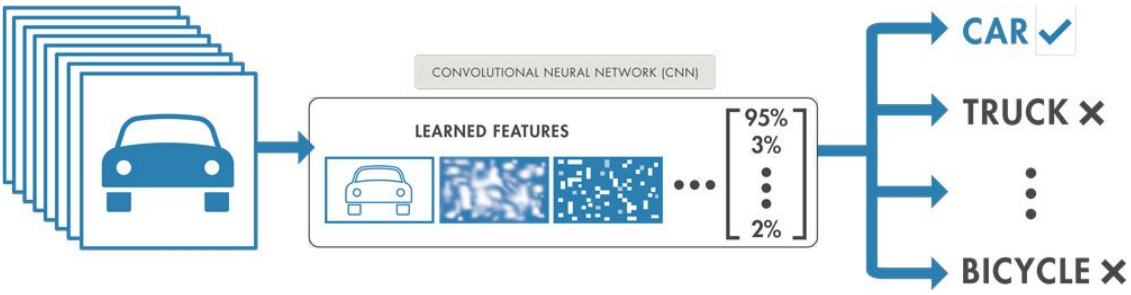

Figura 3.1: L'immagine in input viene analizzata dalla rete la quale è in grado di riconoscere cosa viene rappresentato nell'immagine

Sono dette convoluzionali perché sfruttando una serie livelli (layer) all'interno dei quali traminte delle operazioni di convoluzione si estrae da un'immagine di input una matrice detta "*mappa delle caratteristiche*" (o feature map).
Una rete neurale convoluzionale può avere dalle decine alle centinaia di layer, ciascuno dei quali è in grado di apprendere feature diverse di un'immagine.
Ad ogni livello, a ciascuna immagine usata per l'addestramento della rete vengono applicati dei filtri a diverse risoluzioni e l'output di ciascuna immagine convoluta viene utilizzato come input per il layer successivo. I filtri possono estrarre inizialmente feature molto semplici come, ad esempio, la luminosità o i bordi degli oggetti rappresentati, e diventare sempre più complessi fino a includere feature che definiscono in modo univoco l'oggetto.
Analogamente ad una rete neurale tradizionale, una CNN possiede neuroni con pesi e bias. Il modello imposta e gestisce questi valori durante il training aggiornandoli costantemente ad ogni nuovo passo di addestramento.

In generale, la maggior parte delle reti convoluzionali presentano una struttura composta da tre tipi principali di layer, che sono:

- **Convolutional Layer** (Strato Convoluzionale)
- **Pooling Layer**
- **Fully-Connected Layer** (Strato Completamente Connesso)

In linea generale, una rete neurale convoluzionale è organizzata secondo lo schema riportato in figura 3.2.

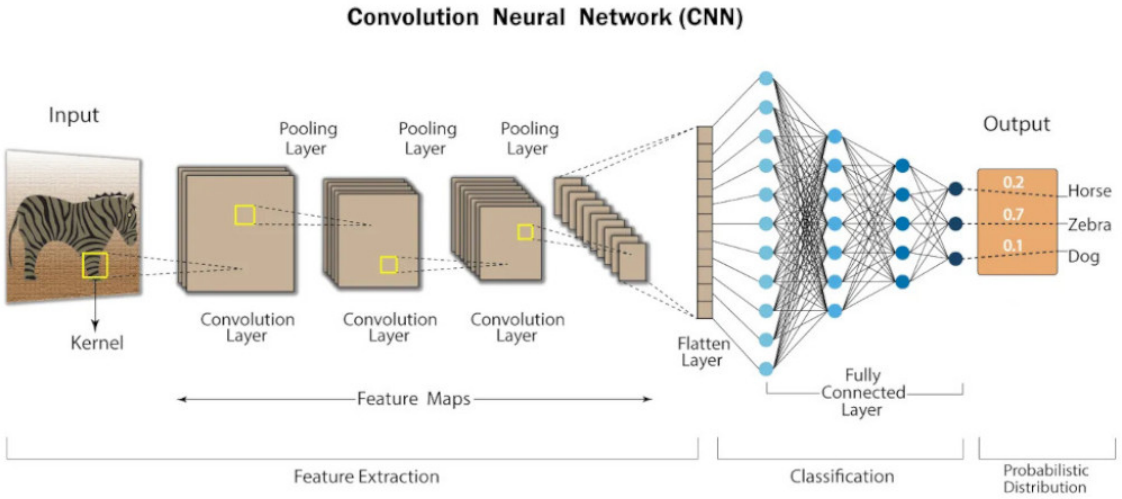

Figura 3.2: Rappresentazione schematica di una rete CNN

Concolutional Layer

Lo strato convoluzionale è il componente principale ed il più importante di una CNN perché è qui che avviene l'esecuzione della maggior parte dei calcoli.
In matematica, in particolare nell'ambito dell'analisi funzionale, una convoluzione è definata come un'operazione tra due funzioni di una variabile che consiste nell'integrare il prodotto tra la prima e la seconda traslata di un certo valore.
Nel caso specifico di una immagine una convoluzione è eseguita tra una matrice che rappresenta l'immagine ed una detta *kernel* che rappresenta un filtro, quindi questa operazione equivale essenzialmente ad applicare un filtro all'immagine. In figura 3.3 viene riportato un esempio di una operazione di convoluzione.

Il risultato dell'operazione di convoluzione dipende ovviamente dal contenuto numerico della matrice del kernel. Diversi kernel, infatti, tendono ad evidenziare o a nascondere alcune caratteristiche specifiche di un'immagine. Ad esempio un kernel può evidenziare i contorni orizzontali, un altro quelli verticali, un terzo evidenziare solo le parti circolari, altri ancora vengono usati per produrre un effetto blur[1] oppure, ancora, per traslare i pixel di un'immagine.
Convenzionalmente, nei primi layer convoluzionali i kernel sono predisposti per "estratte" delle feature di basso livello come ad esempio i bordi, il colore oppure l'orientamento delle forme. Andando poi sempre più in profondità invece la rete si "adatta" ad estrarre feature

[1] L'effetto *Blur* consiste semplicemente nello sfocare un'immagine o parte di essa.

sempre più specifiche cosi da poter classificare le immagini in modo del tutto simile a come faremo noi umani.
Il risultato di una operazione di convoluzione eseguita all'interno di una rete neurale è una matrice detta *feature map* o matrice delle caratteristiche (vedi figura 3.4).

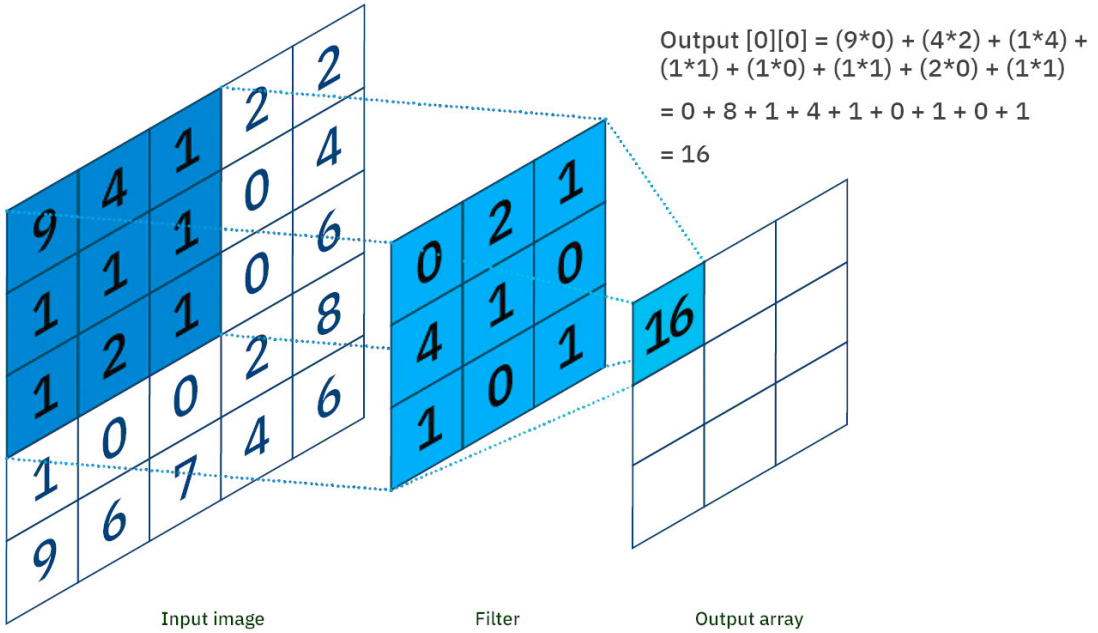

Figura 3.3: Esempio di una operazione di convoluzione tra due matrici, una rappresentante un'immagine e l'altra un filtro

Infine, per quanto riguarda la definizione di un layer convoluzionale i parametri principali in gioco sono i seguenti:

- **Numero di canali in input**: numero di canali dell'immagine di input (per esempio 3 per immagini RGB).
- **Numero di canali in output**: numero di canali che si voglio ottenere dopo la convoluzione.
- **Kernel size**: rappresenta le dimensioni della matrice del kernel di convoluzione.
- **Padding**: rappresenta le dimensioni di un eventuale bordo esterno applicato alla immagine di input.
- **Stride**: rappresenta la "velocità di traslazione" del kernel sull'immagine, misurata in pixel orizzontali e verticali.

Figura 3.4: Visualizzazione delle feature map estratte al passaggio di un'immagine attraverso 18 livelli convoluzionali

Pooling Layer

Il livello di pooling, noto anche come *downsampling layer*, riduce il numero di parametri dell'input eseguendo la riduzione della dimensionalità della feature map.
Solitamente un layer di pooling è utilizzato sempre dopo un layer convoluzionale. Il motivo è semplice: i filtri del layer convoluzionale forniscono informazioni più dense e pure, perché evidenziano alcune caratteristiche eliminandone altre che costituirebbero rumore. Queste informazioni sono quindi più facili da elaborare e non è necessario portarsi dietro tutta la pesantezza data da una immagine di grandi dimensioni. Il pooling serve proprio a questo: diminuire le dimensioni dell'immagine in input, mantenendo le caratteristiche principali della stessa.
Questa operazione viene effettuata scansionando la matrice delle caratteristiche tramite una matrice di dimensioni più piccole (solitamente 2×2). Man mano che la matrice scorre sulla feature map si estraggono i valori che andranno così a generare una nuova feature map di dimensioni minori.

In definitiva, quindi, il livello di pooling effettua uno "scalamento" della feature map. Questo scalamento può essere effettuato prendendo o i valori medi, in questo caso si parla di *average pooling*, oppure il valore maggiore, *max pooling* (vedi figura 3.5).

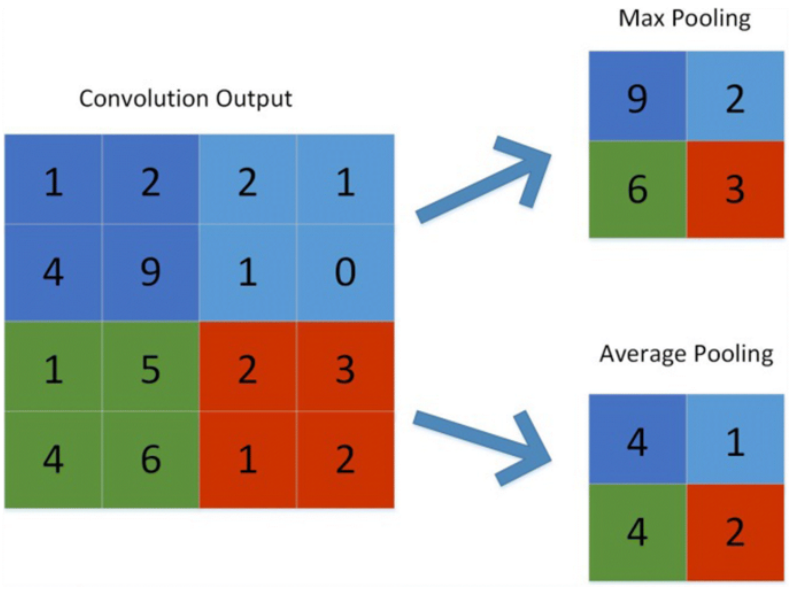

Figura 3.5: Esempio di max pooling e average pooling

Va sottolineato però che il risultato finale non è un semplice ridimensionamento dell'immagine, perchè, come mostrato in figura 3.6, viene mantenuta l'informazione massima o media di ogni feature map.

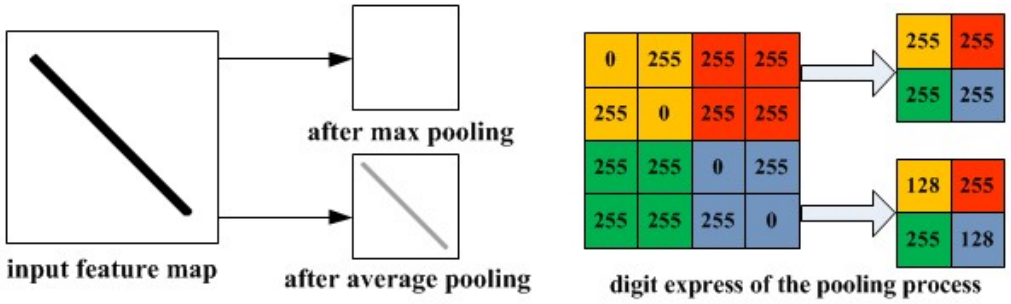

Figura 3.6: Differenze nell'utilizzo dell'average pooling e del max pooling

Per quanto riguarda la definizione di un layer di pooling i parametri principali sono sostanzialmente gli stessi di quelli vista per un livello convoluzionale:

- **Kernel size**: rappresenta le dimensioni della matrice che andrà a "scansionare" la feature map.
- **Padding**: rappresenta le dimensioni di un eventuale bordo esterno applicato alla immagine di input.
- **Stride**: rappresenta la "velocità di traslazione" del kernel sull'immagine, misurata in pixel orizzontali e verticali.

Fully-connected Layer

Gli strati completamente connessi sono i livelli finali dell'architettura CCN e sono quelli che, partendo dai valori ottenuti nei livelli precedenti, eseguono, tramite l'applicazione di trasformazioni lineari tra i vettori in input e i vari pesi aggiornati della rete, la classificazione finale delle informazioni.
I livelli completamente connessi lavorano avendo come input vettori unidimensionali quindi nella maggior parte dei casi prima di questi livelli è presente un livello di pooling che ridimensiona la feature map seguito da altre funzioni che ne effettuano il "reshape"[2].

[2]Il reshape consiste nel cambiare la "forma" di un array ossia il numero di elementi per ognuna delle sue dimensione senza modificare i dati da esso contenuti. In particolare è possibile aggiunge, rimuovere o cambiare il numero di elementi per ogni dimensione.

Un esempio del funzionamento degli strati completamente connessi è riportato in figura 3.7.

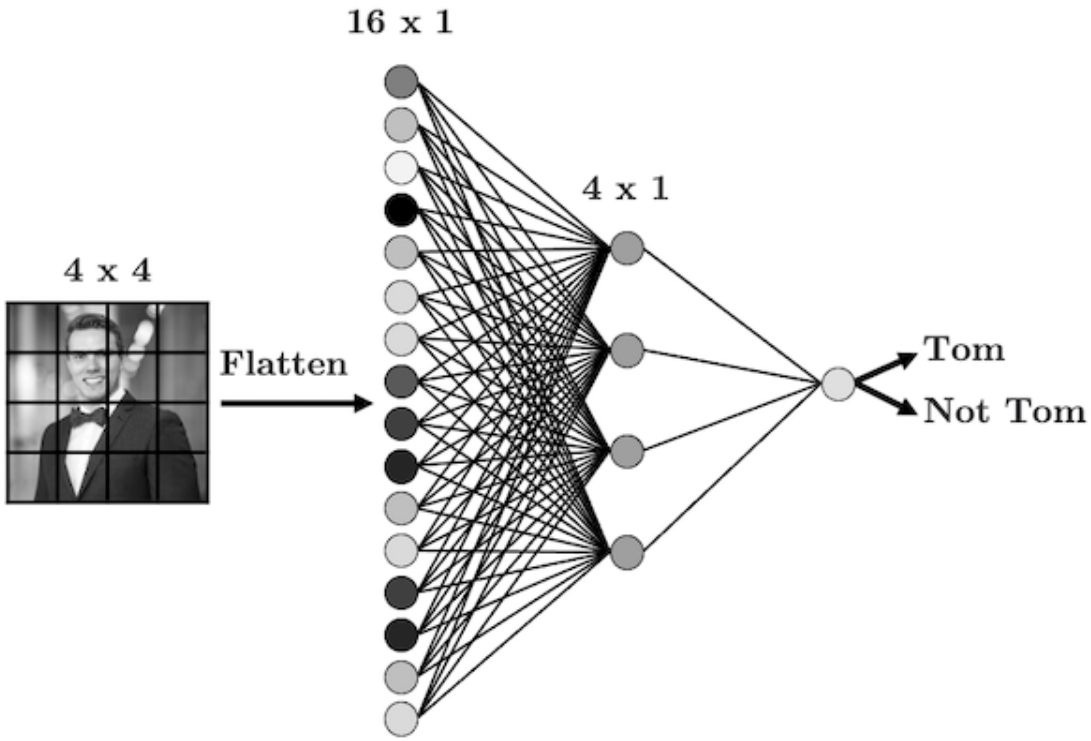

Figura 3.7: Esempio di funzionamento dei livelli completamente connessi

3.1.2 ResNet-18

ResNet-18 [35] è una particolare rete neurale convoluzionale.
ResNet sta per *Residual Network* in quanto viene sfruttato il concetto di "residuo" per migliore le prestazioni della rete.
Per comprendere meglio il concetto di residuo bisogna sapere che nelle prime reti CCN si tendeva ad utilizzare sempre più livelli (aumentando così la profondità della rete) per cercare di ridurre il tasso di errore.
Tutto questo funziona, come si può vedere in figura 3.8, solo fino ad un determinato numero di livelli oltre il qual le reti non miglioravano più anzi, all'aumentare del numero di livelli cresceva il tasso di errore di allenamento e di test[3].
Questo il larga parte era dovuto al fatto che arrivati ad un certo punto era molto più probabile che si venissero a creare degli errori e che tali errori venissero poi propagati per tutta la rete.

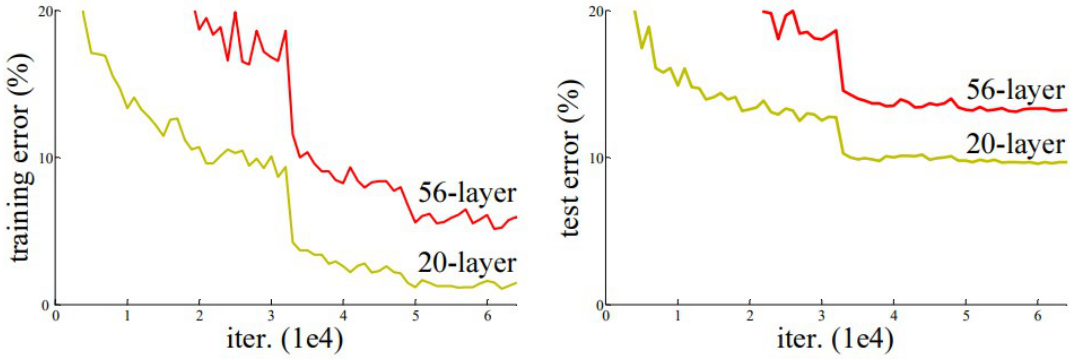

Figura 3.8: I grafici in figura mostrano l'andamento del training error e del validation error relativi a due reti profonde rispettivamente 20 e 56 livelli. Come si può notare in entrambi i grafici l'errore cresce al crescere della profondità della rete

Per far fronte a questo problema nel tempo è stato introdotto il concetto di blocco residuo.
L'approccio consiste nell'aggiungere una "scorciatoia" o "connessione di salto" che consenta alle informazioni di fluire, per esempio, più facilmente da un livello ad un altro livello posto "più in basso" (figura 3.9), ovvero di bypassare alcuni livelli rispetto al normale flusso da un livello al livello successivo.
Così facendo, se durante il normale flusso dei dati attraverso la rete CNN un livello danneggia le prestazioni dell'architettura sarà possibile poi "recuperare" utilizzando i valori

[3]Il *training error* indica quanto una rete stia effettivamente "imparando" durante la fase di training. In particolare se il processo di training sta procedendo per il meglio il training error assumerà valori sempre più piccoli. Il *test error* invece consiste nel valutare quanto una rete già allenata sia in grado di funzionare con dati generici differenti da quelli utilizzati per il training.

che non hanno "bypassato" il livello in questione.

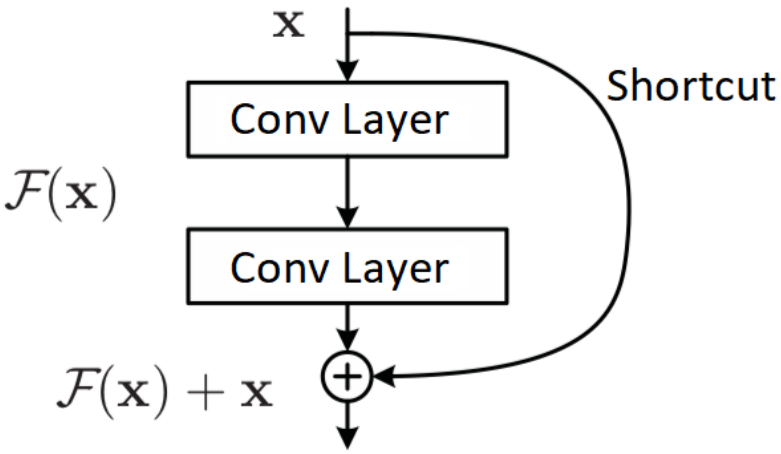

Figura 3.9: Funzionamento di un blocco residuo

Pertanto, aggiungendo nuovi layer, grazie alla “connessione residua”, si può formare una rete molto profonda garantendo, come mostra il grafico in figura 3.10, che le prestazioni del modello non diminuiscano ma che addirittura potrebbero aumentare leggermente.

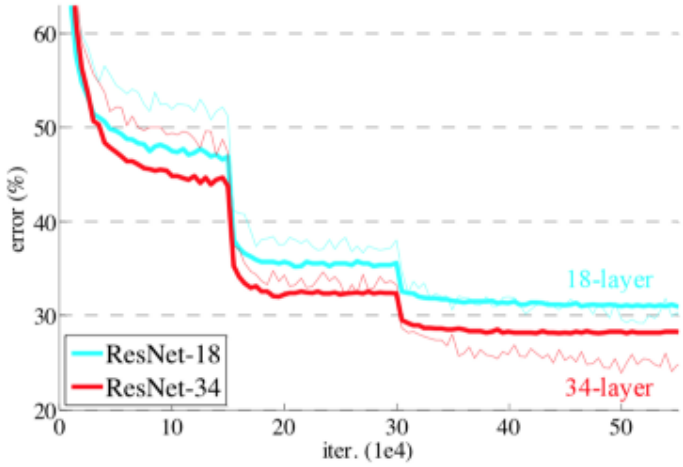

Figura 3.10: Performance di ResNet all’aumentare della profondità della rete

In particolare, ResNet-18 è “profonda” 18 livelli e presenta una struttura come quella mostrata in figura 3.11 e 3.12

layer name	output size	18-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
	1×1	average pool
		fc layer

Figura 3.11: Schema descrittivo di ResNet-18

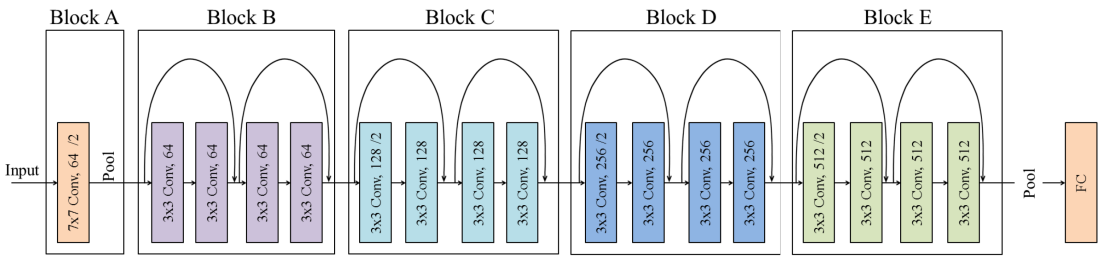

Figura 3.12: Schema a blocchi di ResNet-18

Di base ResNet-18 è progettata per lavorare con immagini RGB in input e di dimensioni 224×224px ma grazie al livello di averagepool finale è in grado di gestire anche immagini in input con risoluzioni differenti.

3.1.3 Fase di Training

Ogni rete neurale prima di poter svolger il compito per cui è stata progettata, prevede una sessione di training.
In questa fase una rete neurale è allenata mediante un dataset di dati specifico per il tipo di task da svolgere. Nel caso di reti dedicate alla ricostruzione di oggetti 3D i dataset usati per il training contengono immagini e modelli 3D di vari oggetti.

La fase di training ha una durata ed un consumo in termini di risorse molto variabile. Questo dipende sia dalla "profondità" della rete che dalle dimensioni del dataset di dati usati per il training. In generale si possono avere sessioni di training che durano meno di un'ora o anche sessioni che durano diversi giorni.

Una sessione di training consiste nel far "analizzare" alla rete neurale tutto il dataset, usato per l'allenamento, per un numero determinato di volte, scelto più o meno arbitrariamente, detto "*epoch*".
Per ogni epoch, infatti, la rete analizza tutti i dati presenti nel dataset cercando di apprendere qualcosa. Alla fine di ogni epoch viene anche calcolato un errore (training error) che ci indica la differenza tra ciò che la rete fornisce come output rispetto al dato in input, per esempio, nel caso di una rete dedicata alla ricostruzione di oggetti 3D a partire da una singola immagine, quanto il modello ricostruito sarà fedele a quello originale.
Se il processo di training funziona correttamente la rete tenderà a migliorare le proprie performance, riducendo l'errore di training, con il passare delle epoch.
Il training di una rete neurale termina o dopo aver eseguito un numero determinato di epoch o al raggiungimento di una determinata soglia di errore, solitamente settata ad un valore abbastanza basso (nell'ordine di 10^{-5}).

Alla fine di ogni sessione di training la rete neurale avrà così ottimizzato i propri parametri interni per i vari strati, detti comunemente "*pesi*". Tali pesi possono essere estratti e salvati evitando così di dover allenare da zero ogni volta tutta la rete. Al termine del processo di training la rete sarà quindi pronta a svolgere il suo compito.

In particolare, per quanto riguarda il training di BSP-net, essendo essa è composta da due "encoder": uno detto "*autoencoder*" usato per il training dei parametri relativi alla ricostruzione di modelli 3D ed uno detto "*image encoder*" usato per il training dedicato alla Single View Reconstruction; il processo di training si divide quindi in due fasi:

- la prima fase consiste nel training del solo autoecoder. In particolare, in questa fase la rete viene allenata tramite dei modelli 3D in formato voxel di oggetti appartenenti a varie classi. Tramite questo training la rete BSP-net, sfruttando una serie di convoluzioni nello spazio 3D, si allena ad estrarre per ogni voxel delle feature "spaziali" (o predittori). Tali feature saranno poi utili sia per riconoscere oggetti simili che per la ricostruzione dei vari modelli 3D.

- La seconda fase consiste nel training dell'image encoder. In questo caso BSP-net è

allenata usando i predittori ottenuti dall'autoecoder più un dataset di immagini rappresentanti vari oggetti. In questa fase sfruttando delle covoluzioni 2D si otterranno delle feature che saranno poi confrontate con i predittori ottenuti in pecedenza cercando di avere cosi un match tra l'immagine bidimensionale ed il modello tridimensionale. Tramite varie ottimizzazioni, dunque, la rete sarà così in grado di riconoscere da una singola immagine la classe e il tipo di modello da ricostruire.

Il processo di training di BSP-net è stato rappresento in figura 2.3.

Entrambi gli encoder di BSP-net usano un modello convoluzionale (CNN) ed in particolare l'image encoder sfrutta il modello ResNet-18. Nel caso di BSP-net però l'unica differenza è quella di non possedere i livelli di pooling e dunque, progettata in questo modo, la rete consente di lavorare solo con immagini di dimensione 128×128px.
In particolare, le immagini usate da BSP-net per il training sono immagini con risoluzione 128×128px in scala di grigio con un solo canale (L).

3.1.4 Ricostruzione degli oggetti 3D

Nella fase ricostruzione la rete neurale allenata, o inizializzata con dei pesi ottenuti da un precedente training, è in grado di prendere in input un'immagine di un oggetto ed attraverso un procedimento "inverso" a quello usato nella fase di training fornire in output un modello 3D (figura 3.13).

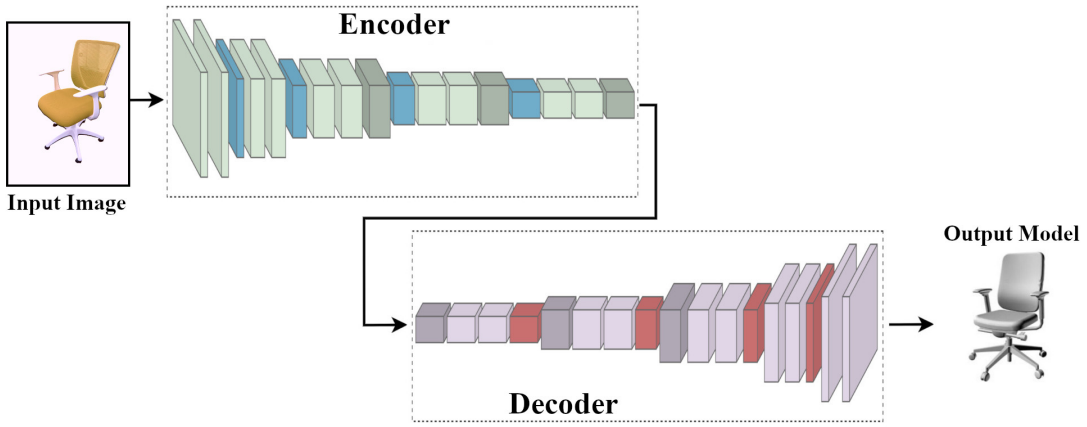

Figura 3.13: Ricostruzione di un oggetto 3D a partire da una singola immagine

Se la rete ha effettuato un processo di training ottimale usando un numero sufficiente di dati, e ottenuto dunque un errore di training molto basso, questa sarà in grado di ricostruire un oggetto da qualsiasi tipologia di immagine, reale o sintetica, a colori o in scala di grigio.
L'unica accortezza che bisogna avere è che le immagini in input per la ricostruzione devono

avere la stessa risoluzione e lo stesso numero di canali delle immagini utilizzate in fase di training. Questo perché, come detto prima, una rete neurale utilizza esattamente il processo inverso a quello utilizzato in fase di training per "decodificare" un'immagine. Quindi se la rete è stata allenata, per esempio, con immagini di dimensione 224×224px questa avrà gestito i pesi "sapendo" di dover lavorare con tale dimensione e di conseguenza si aspetta che in input abbia sempre delle immagini con risoluzione 224×224px.

3.2 Pix3D

Il dataset di immagini e modelli fornito dagli autori di *Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling* [10] è stato usato in questa ricerca nella fase di ricostruzione di modelli 3D. È stato scelto Pix3D in quanto offre delle categorie di modelli standard uguali a quelli utilizzati in altre studi relativi reti neurali dedicate alla ricostruzione di modelli 3D.

Nell'articolo riquadrante Pix3D gli autori sottolineano il fatto che al giorno d'oggi nel campo della Computer Vision, soprattutto in ambito 3D, si ha la necessita di avere a disposizione dei dataset in grado di offrire non solo modelli 3D ma anche immagini che mostrino questi modelli in situazioni reali. I dataset disponibili, a loro dire, presentano alcune limitazioni. In particolare esistono si grandi dataset che offrono molti modelli 3D ma questi associano sempre ai modelli 3D immagini sintetiche e non reali. Esistono, in alternativa, dataset che offrono sia modelli 3D che le corrispondenti immagini reali ma si tratta sempre di dataset di piccole dimensioni e molto specifici. Di conseguenza, chiunque avesse la necessità di trattare sia modelli 3D che le corrispondenti immagini reali deve per forza di cose crearsi il proprio dataset.
Per far fronte a questo problema gli autori hanno dunque costruito questo dataset che cerca di venire in contro a questa esigenza.

In particolare, Pix3D offre in totale circa 400 oggetti divisi in 9 categorie quali: sedie, tavoli, letti, scrivanie, librerie, guardaroba, divani, strumenti e oggetti vari.
Oltre ad offrire dei modelli 3D, per ogni singolo modello Pix3D offre anche una serie di immagini del corrispondente oggetto reale ripreso in contesti di comuni per un totale di circa 10.000 immagini (figura 3.14).

Sfruttando questo particolare dataset che fornisce sia le immagini dell'oggetto, utilizzate come input per la rete neurale, che il corrispondente modello 3D "originale", utilizzato come "ground truth", è stato possibile valutare, per ogni dataset sintetico usato per il training della rete neurale, quanto l'oggetto ricostruito fosse fedele al corrispondente ground truth.

Un altro aspetto importante da considerare, inoltre, è dato dal fatto che le immagini fornite da Pix3D non fossero sintetiche mentre quelle usate per il training della rete si. Utilizzando dunque le immagini reali di Pix3D si è confermato ulteriormente che è comunque possibile utilizzare dataset di immagini sintetiche per il training di una rete neurale che

andrà poi a lavorare con immagini reali.

Figura 3.14: Tipologia di dati forniti dal dataset Pix3D: un modello 3D dell'oggetto, delle immagini reali dell'oggetto e, per ogni immagine, una "maschera" da utilizzare per la rimozione del background

3.3 Blender e BlenderProc

Per la realizzazione di tutti i vari dataset utilizzati per il training della rete neurale si è proceduto utilizzando il software Blender [6] e la pipeline di BlenderProc [16] per automatizzare tutte le fasi: dal caricamento dei modelli, all'aggiunta dei materiali fino al render.
Blender in particolare è stato usato oltre che per la normale gestione e rifinitura dei modelli 3D, anche per l'esecuzione di tutto il codice relativo alla pipeline di BlenderProc che, tramite i suoi moduli e le relative funzioni, è stata utilizzata per la gestione di tutto il processo relativo alla creazione e gestione di materiali, luci, camere e parametri dei render finali.

3.3.1 Blender

Blender è un software dedicato alla modellazione 3D, rigging, animazione, texturing, rendering, composizione e montaggio video. Dispone inoltre di funzionalità per mappature UV, simulazioni fisiche, particellari e di fluidi.
Pur essendo open source le funzionalità messe a disposizione da Blender, per caratteristiche e potenzialità, sono paragonabili, se non a volte migliori, di quelle offerte da altri software più noti come *Cinema4D* [8], *3D Studio* Max [1] o *Maya* [20].

Uno dei punti di forza di Blender, oltre al fatto di essere distribuito con licenza open source, è quello di poter essere eseguito su molte piattaforme richiedendo poco spazio per essere installato.
Offre inoltre la possibilità di importare o esportare modelli di formati differenti come *.obj*, *.fbx*, *.dae*, *.svg* ed altri ancora.

L'interfaccia di Blender

L'interfaccia di Blender di base è suddivisa in precise sezioni, dove al centro è posizionata l'area di lavoro. Su tale area verranno collocati gli oggetti da modificare, mentre sulla destra del pannello, si trovano le selezioni per gestire le scene e le caratteristiche degli oggetti in lavorazione (figura 3.15).
L'ambiente di lavoro risulta essere comunque altamente personalizzabile in base alle esigenze di ciascun utente. L'utente, infatti, può definire più layout differenti per le varie finestre presenti Blender, chiamate schermate, e passare rapidamente da una all'altra selezionandole da un menu o attraverso delle scorciatoie da tastiera.
Quasi tutte le funzioni infatti, oltre che dai vari menù presenti nelle varie sezioni, possono essere richiamate con scorciatoie da tastiera e per questo motivo quasi tutti i tasti sono collegati a numerose funzioni.

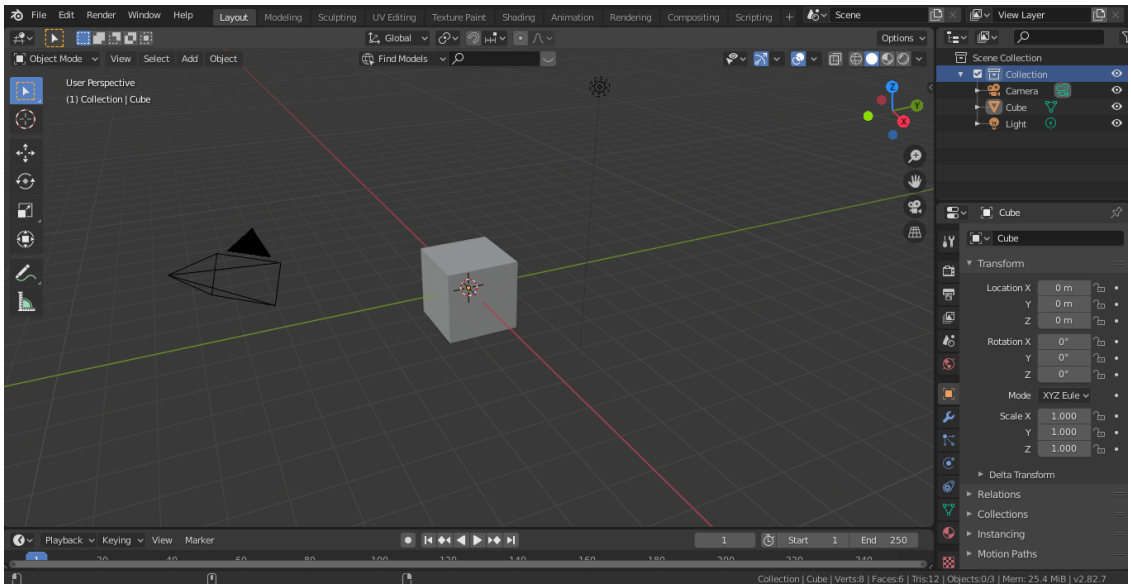

Figura 3.15: Interfaccia di Blender

Modellazione 3D

Blender è utilizzato soprattutto come strumento per la modellazione di oggetti 3D. Offre infatti una varietà di primitive geometriche, tra cui mesh poligonali, curve di Bézier, superfici NURBS, metaball, icosfere, testo e un sistema di modellazione n-gon chiamato B-mesh grazie alle quali è possibile modellare in maniera semplice qualsiasi cosa.
Possiede anche un avanzato sistema di modellazione poligonale a cui è possibile accedere tramite la modalità *edit mode* e supporta operazioni come l'estrusione, smussatura di angoli e la suddivisione.
Blender offre inoltre funzionalità che permettono di gestire lo sculpting 3D, il remesh e la retopology utilizzata per la semplificazione di modelli complessi.
Consente inoltre la gestione e creazione di materiali tramite texture procedurali o basate su nodi, nonché texture painting, o anche semplicemente l'importazione e applicazione di materiali esterni facilmente reperibili in rete.

Blender include molte modalità per interagire con gli oggetti, le due principali sono *Object Mode* ed *Edit Mode*, che vengono attivate con il tasto Tab. La Object Mode viene utilizzata per spostare, ridimensionare e ruotare intere mesh poligonali. La Edit Mode viene utilizzata invece per manipolare i dati dell'oggetto, per esempio per manipolare i singoli vertici di una singola mesh.
Ci sono anche molte altre modalità, come Vertex Paint, Weight Paint e Sculpt Mode.

Render

Per quanto riguarda la parte relativa al render Blender presenta al suo interno diversi motori di render. I più utilizzati sono *EEVEE* e *Cycles*.

EEVEE è un motore di rendering in real-time. Ciò significa che esegue il rendering così velocemente, che è praticamente istantaneo.
EEVEE, infatti, sfrutta un processo chiamato rasterizzazione per stimare il modo in cui la luce interagisce con oggetti e materiali. Tale processo però non tiene conto di fattori importanti come possibili riflessioni o "rimbalzi" di luce proveniente da altri oggetti che si trovano all'esterno della visuale della camera (*frustum*), portando spesso ad avere dei risultati poco realistici.

A differenza di EEVEE, Cycles è un motore di render ray-traicing. Cycles, infatti, è ottimizzato per riprodurre la scena nel modo più fedele possibile alla realtà gestendo tutti i fenomeni fisici della luce quali per esempio riflessioni e rifrazioni.
Per fare ciò l'algoritmo di ray-tracing traccia tutti i raggi luminosi partendo da una telecamera presente nella scena, finché non trovano una fonte di luce (una lampada, un materiale che emette luce o lo sfondo del mondo). Se nel suo percorso un raggio "colpisce" un oggetto, a seconda del materiale specifico di tale oggetto, il raggio può essere trasmesso o riflesso con intensità differente rispetto all'istante precedente all'impatto. Calcolando il percorso di tutti i raggi e come questi interagiscano con gli elementi presenti nella scena si avrà dunque una rappresentazione degli effetti della luce molto più accurata.
Ovviamente, un render eseguito con Cycles risulta essere più realistico ma richiede anche

molto più tempo per elaborare l'immagine finale rispetto ad render eseguito con EEVEE.

Python in Blender

Blender supporta inoltre lo scripting Python per la creazione di strumenti personalizzati, importazione o esportazione da altri formati e automazione delle attività. Ciò consente l'integrazione con diversi motori di rendering e software esterni tramite l'uso di plug-in e add-on.
Blender, infatti, presenta un interprete Python incorporato che viene caricato all'avvio di del programma e rimane attivo mentre questo è in esecuzione.
L'interprete incorporato di Blender fornisce un tipico ambiente Python, quindi qualsiasi codice Python può essere eseguito utilizzando questo interprete.
Blender fornisce anche i suoi moduli Python, come *bpy* e *mathutils* i quali, importati in uno script Python, consentono l'accesso ai dati, alle classi e alle funzioni di Blender.

3.3.2 BlenderProc

Come si legge nel proprio paper [16] BlenderProc è una pipeline procedurale modulare open source, che aiuta a generare immagini dall'aspetto reale per l'addestramento di reti neurali neurali.
BlenderProc offre infatti una serie di moduli e funzioni sviluppati in linguaggio Python, facilmente configurabili che consento, appoggiandosi all'ambiente di Blender, di gestire in modo semplice ed automatico tutta la fase di creazione del proprio dataset sintetico.

Tramite le funzioni offerte da Blenderproc è possibile gestire infatti il caricamento di modelli 3D, l'applicazione di texture e materiali, la creazione di luci e camere e più in generale tutto ciò che concerne la creazione del proprio dataset sintetico in modo semplice e altamente personalizzabile.
Offre inoltre la possibilità di ottenere in output immagini non solo standard a colori ma anche depth map, normal map ed anche maschere per la segmentazione delle immagini.

I principali moduli messi a disposizione sono:

- **Loader**: offre delle funzioni relative al caricamento di mesh, materiali, texture ed altri oggetti all'interno di una scena.
- **Camera**: offre funzioni relative la gestione della camera. Tramite queste funzioni è possibile posizione la camera in un determinato punto della scena, gestire la profondità di campo, la risoluzione, le dimensioni del sensore e tutti gli altri parametri relativi la camera.
- **Lightning**: offre funzioni dedicate all'illuminazione della scena.
- **Material**: offre funzioni relative la creazione o importazione di materiali e texture, la loro gestione e l'applicazione di questi ai vari oggetti presenti nella scena.

- **Types**: offre funzioni dedicate alla manipolazione di oggetti ed elementi della scena.
- **Render**: offre funzioni per gestire tutte impostazioni legate al render come l'applicazione di DeNoise per la riduzione del rumore, il numero di sample, la trasparenza il formato di output e molto altro ancora.

Inoltre, essendo BlenderProc progettata per lavorare in ambiente Blender, è anche possibile utilizzare tutte le altre funzioni messe a disposizione da Blender in maniera molto semplice. È possibile, infatti, inserire le funzioni "specifiche" di Blender direttamente nel codice scritto per BlenderProc, o viceversa, senza che questo porti alla comparsa di errori.

3.4 Python

Tutti gli script ed in generale tutta la parte relativa alla programmazione è stata sviluppata utilizzando il linguaggio di programmazione Python [26].

Python è un linguaggio di programmazione ampiamente utilizzato nelle applicazioni Web, nello sviluppo di software, nella data science e nel machine learning.
Gli sviluppatori utilizzano sempre più spesso Python in quanto risulta essere efficiente e facile da imparare e può essere eseguito su diverse piattaforme.

I vantaggi di Python sono inoltre:

- Una facilita di lettura e comprensione del codice in quanto possiede una sintassi base, simile a quella della lingua inglese.
- Possibilità di utilizzare meno righe di codice in quanto Python presenta pochi costrutti sintattici a differenza di altri linguaggi di programmazione.
- La presenza di una vasta libreria "standard" che contiene codici riutilizzabili per quasi tutte le attività evita agli sviluppatori dover scrivere i codici da zero. Inoltre, sono disponibili più di 137.000 librerie Python per varie applicazioni, incluso lo sviluppo web, la data science e il machine learning.
- La possibilità di integrare facilmente Python con altri linguaggi di programmazione noti come *Java*, *C*, e *C++*.
- L'esistenza di una community che include milioni di sviluppatori da tutto il mondo dalla quale è possibile ricevere sempre un aiuto in maniera rapida.
- La presenza di molte risorse utili reperibili in rete come ad esempio tutorial, documentazioni e guide degli sviluppatori.
- La possibilità di trasferire il codice scritto in Python su diversi sistemi operativi informatici, come *Windows*, *macOS*, *Linux* e *Unix*.

In particolare, per la ricerca condotta in questa tesi sono state utilizzate, per la parte relativa la creazione di dataset, tutte le librerie Python messe a disposizione da BlenderProc e da Blender.
Mentre per la gestione della rete neurale sono state utilizzate principalmente:

NumPy: NumPy [22] è il pacchetto fondamentale per il calcolo scientifico in Python. È una libreria molto usata dagli sviluppatori per creare e gestire facilmente matrici, modellare forme logiche ed eseguire operazioni rapide sugli array, tra cui manipolazione matematica, logica, di forma, ordinamento, trasformate di Fourier discrete, algebra lineare di base, operazioni statistiche di base, simulazione casuale e molto altro.
NumPy supporta inoltre l'integrazione con molti linguaggi, tra cui C e C++.

PyTorch: PyTorch [27] è un framework open-source di Python utilizzato soprattutto per applicazioni in ambito di Machine Learning. È costituito da una libreria di base chiamata Torch, la quale contiene funzioni e moduli per la gestione di tensori multidimensionali (ossia dei contenitori di dati, in generale dati numerici, che possono essere visti come una generalizzazione delle classiche matrici aventi un numero arbitrario di dimensioni) definendo le possibili operazioni matematiche che potranno essere svolte su questi tensori.
PyTorch risulta infatti particolarmente utile nel campo dell'elaborazione dei tensori in quanto permete di sfruttare l'accelerazione delle schede grafiche (GPU) riducendo di molto i tempi di elaborazione.
Grazie a questa sua caratteristica, inoltre, PyTorch viene utilizzato spesso anche come un valido sostituto di NumPy in quanto riesce a sfruttare al meglio tutta la potenza di calcolo fornita della GPU.
Questo framework offre inoltre altre librerie e moduli utili allo sviluppo di una rete neurale in modo molto semplice flessibile e veloce come, per esempio, *Torch.nn* o *Torch.utils*.

Open3D: Open3D [23] invece è una libreria open source che supporta lo sviluppo rapido di software che si occupano di dati 3D. Presenta infatti molti metodi per:

- l'elaborazione e visualizzazione di dati 3D
- la ricostruzione e visualizzazione di scene 3D
- la creazione di strutture dati 3D
- il rendering di modelli 3D
- il supporto per l'apprendimento automatico 3D con PyTorch e TensorFlow

Open3D è disponibile sia in linguaggio Python che C++

Capitolo 4

Pipeline proposta e definizione degli attributi di un'immagine da verificare

4.1 Descrizione della pipeline adottata per la creazione di dataset di immagini sintetiche

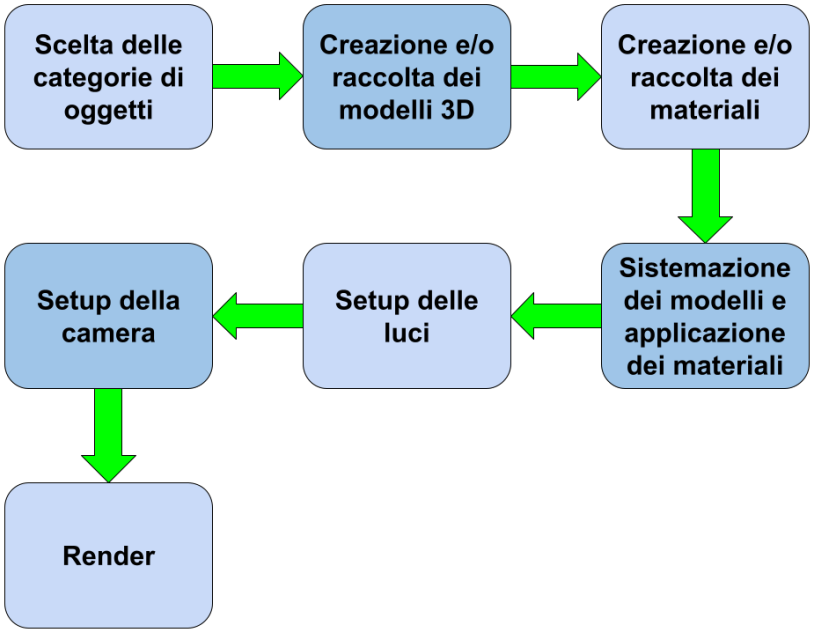

Figura 4.1: Schema della pipeline proposta in questa tesi

Per poter procedere ad indagare quali paramenti di un'immagine sintetica incidessero maggiormente nella ricostruzione di un oggetto 3D è stato necessario creare dei dataset custom i quali sono poi stati usati per il training della rete neurale.
Prima di procedere alla generazione dei dataset e alla conseguente fase di training e test della rete neurale si è quindi proceduto a identificare e definire una pipeline di lavoro da seguire per organizzare al meglio tutte le fasi legate alla generazione dei dataset (figura 4.1).

La pipeline adottata è stata pensata per essere utilizzata con il software Blender [6], ma in generale le indicazioni restano valide per qualsiasi software di modellazione 3D.
La pipeline proposta si suddivide in diverse fasi che verranno trattate di seguito:

1. **Scelta delle categorie di oggetti**. Il primo passo consiste nella scelta dei modelli che verranno utilizzati come base per la generazione del proprio dataset. In base alle varie esigenze vengono scelti i modelli reputati più utili ai fini dello scopo da raggiungere.

2. **Creazione e/o raccolta di modelli 3D**. Una volta scelti i modelli di base da usare per la creazione del dataset si passa ad una fase di ricerca o eventualmente di creazione dei vari modelli. Se si ha la necessita di avere a disposizione un numero molto grande di modelli è possibile sfruttare dataset già esistenti o eventualmente procedere ad una ricerca dei modelli dai vari siti web dedicati.
   La scelta dei modelli 3D è un'operazione che richiede una certa attenzione soprattutto se questi provengono da dataset e fonti diverse o se sono stati ottenuti da differenti siti web. Questo perchè si potrebbero presentare in seguito sia problemi di compatibilità con il software che tutta una serie di altri problemi legati ai differenti formati utilizzati ed alle eventuali conversioni che Blender applica (uno su tutti la presenza di normali invertite o l'inversione degli assi come mostrato in figura 4.2).

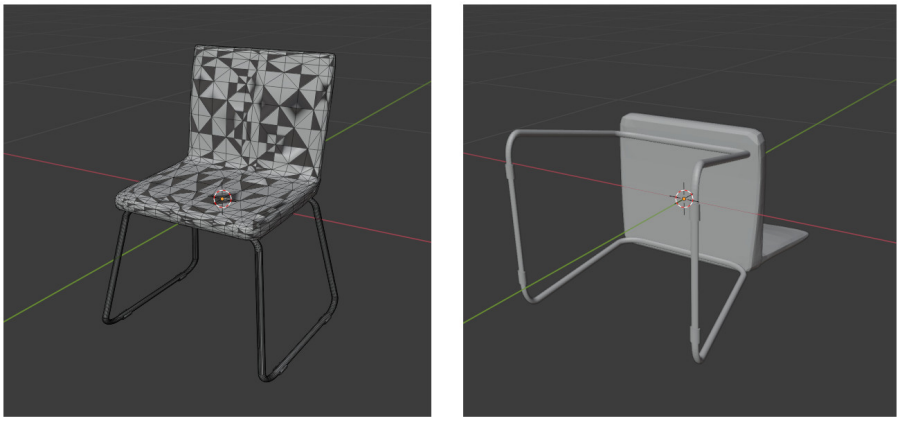

Figura 4.2: (a) Mesh con normali invertite, (b) mesh orientata in maniera differente rispetto agli assi di Blender

3. **Creazione e/o raccolta dei materiali**. Avendo ottenuto i modelli 3D necessari un altro passo da fare sarà quello di creare una raccolta di materiali da applicare ai vari modelli. Anche in questo caso è possibile sfruttare i vari dataset di texture presenti in rete o in alternativa creare manualmente le proprie texture.

4. **Sistemazione dei modelli e applicazione dei materiali**. A questo punto avendo sia i modelli che i materiali una prima cosa da fare sarà quella di normalizzare i modelli rispetto ad una determinata dimensione (per esempio facendo in modo che tutti i modelli risultino essere contenuti all'interno di un cubo o di una sfera unitaria come mostrato in figura 4.3) in modo tale da semplificare poi il processo di gestione delle luci e della camera.

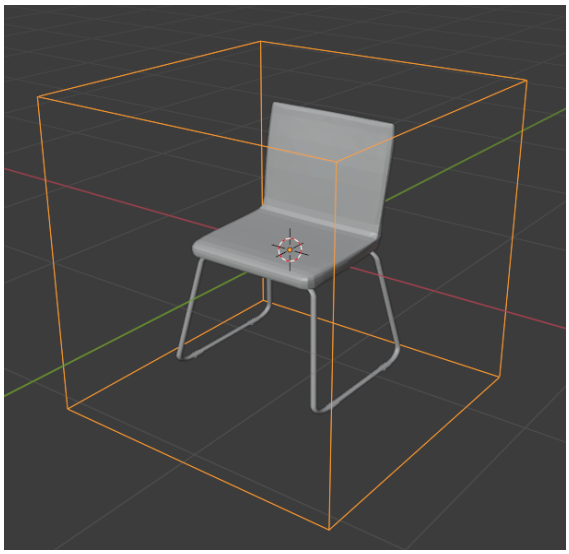

Figura 4.3: Modello 3D normalizzato in modo tale da essere posizionato all'interno di un cubo unitario

Avendo dunque sistemato tutti i modelli si passa all'applicazione dei materiali. Questa operazione può essere effettuata in diversi modi:

- In maniera casuale: tramite l'uso di apposti script al modello vengono applicati dei materiali casuali. Utile se si hanno dei modelli con uno o due soli materiali come potrebbero essere sedie o tavoli
- In maniera manuale: tramite le funzioni offerte da Blender si applicano manualmente i materiali. Utile se si vuole poter scegliere il tipo di materiale da usare per il singolo oggetto.
- Approccio ibrido: per ogni oggetto si indica una tipologia generica di materiale (es. legno o tessuto) e poi tramite un apposito script si applica casualmente un materiale della stessa tipologia.

Bisogna comunque tenere a mente che se si applica un materiale ad un oggetto senza che sia prima stato eseguito una *UV map*[1] spesso il risultato potrebbe non essere dei migliori, quindi, potrebbe essere necessario generare una mappa UV dei modelli manualmente (per una migliore resa) o in alternati sfruttare la funzione *Smart UV* offerta da Blender.

5. **Setup delle Luci**. Prima di procedere alla creazione dei render è necessario aggiungere un'illuminazione all'interno della scena. La scelta del tipo di luci, del colore e della potenza luminosa dipende esclusivamente dallo scopo per cui sarà poi utilizzato il dataset ma in linea generale si può pensare di posizionare le luci in modo tale da illuminare al meglio il modello 3D evitando però allo stesso tempo di generare effetti indesiderati sull'oggetto stesso come ad esempio ombre nette o riflessioni (almeno che queste non siano volute).

6. **Setup della Camera**. Avendo sistemato le luci si procede infine con l'aggiunta della camera. Anche in questo caso i paramenti della camera verranno settati in base all'utilizzo che si dovrà fare delle immagini ottenute in fase di render.
   Se si vuole avere una camera che riprenda l'oggetto da diverse angolazioni sarà inoltre necessario settare dei keyframe relativi alla posizione della camera facendo comunque in modo che questa inquadri sempre l'oggetto.
   Per quanto riguarda i parametri della camera come ad esempio la lunghezza focale, la profondità di campo o l'apertura del diaframma, anche in questo caso, possono essere settati in base alle proprie necessità. Di base Blender crea comunque una camera con dei parametri di default abbastanza standard i quali vanno bene in molti casi.

7. **Render**. Infine, si può procedere al render delle immagini. Anche in questo caso si può scegliere la tipologia di render da effettuare se a colori o in scala di grigi oppure se si vuole ottenere anche la depth map o la normal map relativa al modello da renderizzare.
   Se si vuole ottenere poi un risultato fotorealistico l'unica accortezza da avere nella fase di render è quella di utilizzare il motore di render Cycles al posto di EEVEE.

Per quanto riguarda i punti dal 4 al 7 della pipeline, questi possono essere gestiti in maniera molto semplice ed automatica sfruttando le funzioni messe a disposizione da BlenderProc [16]. In particolare, è possibile sia creare dei singoli script da eseguire in Blender per gestire singolarmente le varie fasi oppure, come è stato fatto nel caso specifico di questa tesi, scrivere un piccolo programma che si occupa di gestire in modo automatico il caricamento dei modelli, l'applicazione del materiale, la creazione delle luci, la creazione della camera con i relativi keyframe ed il render finale.

[1] L'*UV mapping* è una tecnica di texture mapping che permette di applicare correttamente delle texture su un modello 3D. Questa tecnica consiste nell'"appiattire" una mesh 3D su un piano 2D, sul quale giace un'immagine. A questo punto ogni vertice dell'oggetto tridimensionale disporrà di un set di coordinate bidimensionali condiviso con l'immagine, che potrà essere quindi associata alle sue facce, risultando visibile nello spazio 3D. Le UV map rappresentano quindi una mappatura delle coordinate tra lo spazio bidimensionale delle immagini (texture) e quello tridimensionale della mesh.

Seguendo gli step presentati in questa pipeline e sfruttando le potenzialità offerte sia da Blender che da BlenderProc è stato possibile infatti, una volta implementato il codice necessario, poter facilmente gestire la creazione di tutti i dataset utilizzati per i vari test passando solamente i parametri voluti, come il numero di render per modello o la distanza della camera, al programma di generazione di dataset realizzato per questo studio.

4.2 Feature che potrebbero incidere sul training

Avendo definito al meglio tutta la parte relativa alla creazione di dataset di immagini sintetiche a partire da modelli 3D il passo successivo è stato capire quali potessero essere le feature di un'immagine che maggiormente potrebbero incidere nella fase di training di una rete neurale.
Nell'ambito di questa ricerca ci si è concentrati principalmente sui seguenti parametri: dimensione delle immagini usate per il training, i materiali applicati ai modelli, colore delle immagini e illuminazione del modello.

4.2.1 Dimensioni

Per quanto riguarda le dimensioni delle immagini, analizzando il funzionamento di diverse reti neurali dedicate alla ricostruzione di oggetti 3D, si è osservato che queste usavano in fase di training dataset di immagini a bassa risoluzione, in generale 128×128px o 224×224px. Quindi partendo dall'assunto che immagini più grandi, ossia con risoluzione maggiore, offrissero potenzialmente maggiori informazioni "utili" rispetto ad immagini di dimensioni inferiori si è scelto di analizzare nello specifico se la risoluzione di un immagine fosse un parametro importante da tenere in considerazione nella creazione di un dataset.
Per prima cosa quindi si è cercato se in letteratura fossero stati condotti studi più approfonditi in materia ma gli unici studio condotti relativi alle dimensioni delle immagini usate per il training di reti neurali riguardavano solamente reti dedicate al riconoscimento di determinati elementi all'interno di un'immagine [30][34][32][18]. Dai risultati di questi studi si evinceva però come ci fosse effettivamente una relazione tra dimensione dell'immagine e prestazioni della rete quindi si è deciso di procedere alla creazione di dataset di immagini a differenti risoluzioni con cui poi andare ad effettuare i vari test.

4.2.2 Texture e Materiali

Un altro aspetto che è stato scelto come caso da investigare è il contributo svolto dalla presenza o meno di un materiale applicato ad un oggetto. In particolare, avendo la possibilità di poter gestire i vari materiali a proprio piacimento, come prima cosa ci si è domandati se una rete neurale fosse comunque in grado di ottenere buoni risultati in fase di ricostruzione allenandosi utilizzando solamente immagini di oggetti senza nessun materiale applicato,

dove le uniche feature utili potevano essere rappresentati dai bordi o dalla forma dell'oggetto.
Inoltre, si è scelto di fare anche delle analisi piu specifiche riguardo i vari tipi di materiali utilizzati cercando di capire quali potessero portare miglioramenti alla rete e quali no.

Si è deciso poi di indagare quanto la presenza di parametri relativi il render fotorealistico di immagini come, per esempio, l'applicazione di determinate "mappe[2]" come mormal map o bump map, roughness, specular map ed altro ancora andassero ad influire sul training della rete.
Quest'ultimo caso in particolare nasce dall'assunto che più un materiale risulta essere "complesso" maggiore sarà il numero di calcoli in fase di render con un conseguente aumento dei tempi di generazione dell'immagine. Quindi, per esempio, se potenzialmente l'applicazione o meno di una normal map ad un oggetto non influisse più di tanto sul processo di training, e in generale sulla buona resa della ricostruzione di un oggetto 3D, banalmente si potrebbero utilizzare materiali senza normal map e far così risparmiare tempo e risorse in fase di creazione del dataset.
In ultima analisi si è posta l'attenzione riguardo la scelta di utilizzare immagini a colori rispetto ad immagini in scala di grigi. In particolare, infatti analizzando diverse reti neurali si è osservato che non tutte utilizzavano immagini a colori per il training (come, per esempio, anche BSP-net [39]) dunque si è deciso di capire quanto la scelta in un senso piuttosto che nell'altro potesse poi incidere sul risultato finale.

4.2.3 Illuminazione

Un'ulteriore analisi è stata fatta riguardo l'illuminazione del modello 3D in fase di render. In questo caso si è voluto capire se la presenza di ombre o riflessioni dovuta ad una determinata configurazione delle luci nella scena andasse ad impattare sul training della rete neurale e sulla buona riuscita della ricostruzione.
Anche in questo non sono stati trovati studi specifici a riguardo quindi si è proceduto a analizzare due differenti tipologie di illuminazione: una consiste nell'illuminare il modello con una luce abbastanza omogenea mentre la seconda di illuminare il modello sfruttando il classico sistema detto *three-point lighting*[3] (figura 4.4).

[2]Per la generazione di materiali che siano altamente fotorealistici spesso si fa ricorso a partiolari *texture map* ognuna delle quali aggiunge dei dettagli oltre al semplice colore di base di un materiale. In particolare una Normal map contiene tutti i dettagli relativi i rilievi o le protuberanze di un materiale, la Roughness map contiene informazioni circa la "ruvidità" di un materiale mentre la Specular map contiene informazioni riguardanti le interazioni con la luce di un determinato materiale.

[3]Il sistema *three-point lighting* o "illuminazione a tre punti" è una particolare tipologia di illuminazione professionale standard utilizzata in fotografia, nel cinema o in atri media visivi che si basa sull'utilizzo di tre fonti di luce per illuminare un oggetto o un personaggio. Facendo variare l'intensità, la distanza, la posizione e l'inclinazione delle varie luci è possibile infatti gestire al meglio le eventuali ombre e riflessioni in modo semplice e veloce.

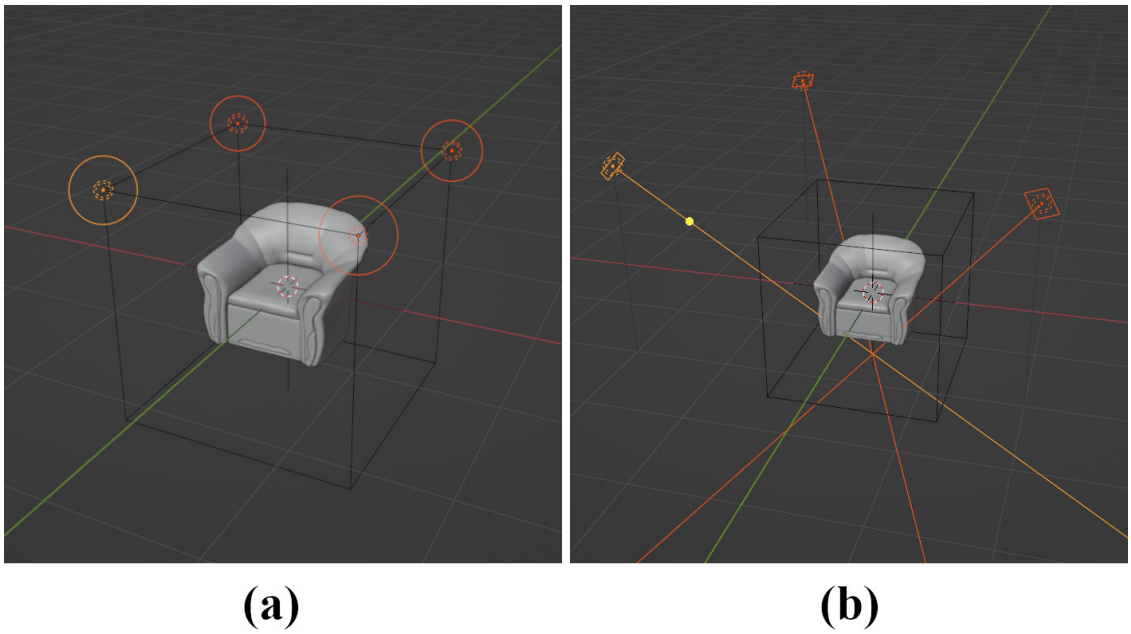

Figura 4.4: Posizione delle luci utilizzate. (a) Luci posizionate in modo tale da illuminare in maniera omogenea il modello 3D, (b) luci posizionate secondo il sistema three-point linghting

4.3 Chamfer Distance

Avendo stabilito quali potessero essere i parametri che potenzialmente potessero influire di più sul processo di training di una rete neurale e successivamente sulla ricostruzione di un oggetto 3D si è dovuto poi identificare quale fosse il metodo più efficace per eseguire una valutazione sulle prestazioni ottenute dalla rete neurale in termine di precisione nella ricostruzione di un oggetto 3D.

Come metodo per valutare l'effettiva fedeltà del modello 3D ricostruito dalla rete rispetto all'originale si è scelto di utilizzare la *Chamfer Distance*, una metrica ampiamente utilizzata in letteratura soprattutto per valutare le prestazioni di reti neurali che si occupano della ricostruzione di oggetti 3D.
La Chamfer Distance (CD), infatti, è una metrica di valutazione che tiene conto della distanza di ogni singolo punto appartenente ad una nuvola di punti A rispetto al corrispondente punto "più vicino" appartenente ad un'altra nuvola di punti B.
In figura 4.5 è riportato un esempio grafico del calcolo della Chamfer Distance tra due nuvole di punti.

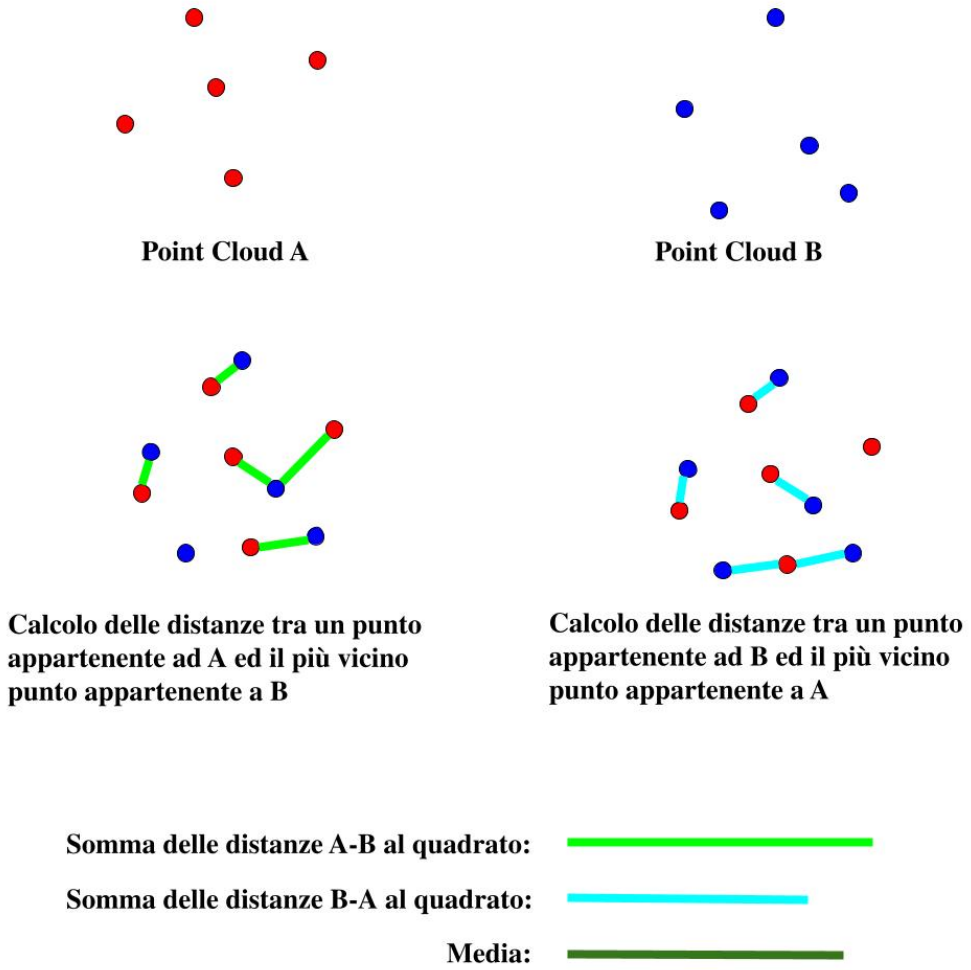

Figura 4.5: Rappresentazione grafica del calcolo della Chamfer Distance

La Chamfer Distance è ottenuta facendo la media delle distanze al quadrato tra le coppie di punti più vicini appartenenti a due nuvole di punti differenti utilizzando la formula mostrata in figura 4.6. Quindi, semplicemente, più è basso il valore della Chamfer Distanze minore saranno le differenze tra le due nuvole di punti e di conseguenza tra i due modelli 3D analizzati.

$$\mathrm{CD}(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

Figura 4.6: Formula matematica per il calcolo della Chamfer Distance

Dato che per il calcolo della Chamfer Distance occorre avere due nuvole di punti come input è stato necessario creare delle nuvole di punti dei modelli di Pix3D utilizzati come groud truth.
Per fare ciò è stata utilizzata una funzione della libreria Open3D [23] di Python che consente di convertire una mesh in una nuvola di punti. Mentre per quanto riguarda i modelli ricostruiti ottenuti tramite BSP-net non c'è stato bisogno di fare nulla in quanto questa rete è in grado di fornire in output oltre che le mesh 3D anche le nuvole di punti degli oggetti ricostruiti. Per il calcolo effettivo della Chamfer Distance invece si è sviluppato un script nel quale venivano utilizzate delle funzioni messe a disposizione dalle librerie PyTorch [27], Scikit-learn [31] e Open3D [23].

Capitolo 5

Lavoro svolto

Dopo aver fatto una analisi di quanto già presente in letteratura, chiarito quali fossero gli obietti da raggiungere, definita una pipeline per la creazione dei database e trovata una metrica per la valutazione dei modelli ricostruiti si è proceduto alla parte pratica del lavoro svolto in questa tesi.

5.1 Creazione dei dataset

Il primo compito da svolgere è stato quello di creare dei dataset di immagini sintetiche da utilizzare per il training della rete neurale.
Questi dataset in particolare dovevano contenere oltre alle immagini dei vari modelli 3D anche una rappresentazione in voxel di tali modelli che sarebbe poi servita alla rete neurale nella fase di training.
Per la creazione delle immagini sintetiche si è proceduto applicando in toto la pipeline proposta e sviluppando tutto il codice necessario alla creazione delle immagini sfruttando le funzioni di offerte da BlenderProc [16].
Per la creazione dei modelli "voxel" ci si è appoggiati ad un semplicissimo tool esterno chiamato *Binvox* [5] che consente di trasformare un qualsiasi modello 3D nel corrispondente modello in formato voxel.

5.1.1 Ricerca dei modelli

La prima parte del lavoro ha riguardato la ricerca di modelli 3D di oggetti appartenenti alle classi scelte da testare.
Si è scelto dunque di utilizzare circa 40 modelli di oggetti per ogni classe per un totale di 120 modelli. Data la necessita di avere un numero così grande di modelli si è pensato fin da subito di sfruttare i vari siti online che offrivano modelli 3D di oggetti.
In particolare, i modelli utilizzati per la creazione dei dataset provengono in gran parte dal sito web di Turbosquid [37] e da BlenderKit [7].

Per la scelta dei modelli si è preferito, quando possibile, utilizzare quelli già disponibili in formato *.blend*, ossia il formato con cui Blender salva i propri file, rispetto ad altri formati come *.dae*, *.fbx* o *.obj*. Questa scelta è dovuta principalmente al fatto che molto spesso quando si importano tali formati in Blender questi possono presentare alcuni problemi come la presenza di normali "flippate" o di assi invertiti che necessitano spesso di una correzione manuale. Utilizzando invece formati .blend, se il modello 3D è ben realizzato, si evitano gran parte di questi problemi (vedi figura 4.2).

Un ulteriore scelta è stata fatta prediligendo modelli 3D che avessero già dei materiali e texture applicati. Tale scelta è stata fatta non perché si voleva utilizzare il materiale specifico di quell'oggetto ma per un semplice motivo pratico: la presenza dei materiali implica quasi certamente la presenza di una UV map fatta all'oggetto. Avendo dunque una mappa UV già pronta, anche se spesso questa necessita di qualche piccola modifica, si risparmia comunque del tempo rispetto a doverne creare una da zero.
Avere i materiali già applicati aiuta inoltre perché ci evita di dovere assegnare manualmente i singoli materiali alle varie parti di una mesh. In questo caso, infatti, tramite un semplice script è possibile andare a sostituire i materiali esistenti con quelli scelti per svolgere le varie prove.

5.1.2 Aggiustamento dei modelli

Pur facendo attenzione nella scelta dei modelli da utilizzare si è dovuti comunque passare per una fase di analisi ed eventuale correzione di errori o di aggiustamento delle varie mesh.

Come prima cosa ogni modello è stato analizzato per controllare la presenza di errori di modellazione. Si è verificato che non vi fosse la presenza di "buchi" nella mesh, vertici duplicati o facce sovrapposte ed in generale tutto ciò che poteva portare alla creazione di effetti non desiderati sia fase di render sia nel successivo processo di trasformazione della mesh in voxel, i quali poi sarebbero stati utilizzati dalla rete neurale nel processo di training.
In particolare, per i modelli con formato diverso dal .blend si è provveduto a controllare l'orientamento delle normali e, quando necessario, al riorientamento dell'oggetto rispetto gli assi di Blender.
Per ogni modello si è verificata anche la presenza di eventuali "modificatori[1]" e si è provveduto all'applicazione di questi. Infatti, dato che per ogni modello serviva anche il suo corrispettivo in formato voxel, la presenza dei modificatori portava sempre alla creazione di modelli voxel spesso incompleti o del tutto errati.

[1]In Blender utilizzare un modificatore o *modifier* consiste nel far eseguire al software una serie di operazioni automatiche che agiscono sulla geometria di un oggetto in modo non distruttivo. Tramite l'uso di modificatori si possono così creare automaticamente molti effetti che altrimenti richiederebbero molto tempo per essere generati manualmente (come per esempio la smussatura dei bordi o la riduzione dei vertici), senza influire sulla geometria di base dell'oggetto. L'effetto di un modificatore su un oggetto, infatti, è sempre reversibile a patto che questo non venga effettivamente "applicato" all'oggetto.

Tutti i modelli sono poi stati normalizzati e scalati in modo tale da essere contenuti dentro un "*empty*[2]" rappresentante un cubo unitario. Questo è stato fatto per semplificare poi il processo successivo di posizionamento delle luci e della camera, in quanto in questo modo mettendo le luci e la camera al di fuori dell'empty è stato possibile, tramite script far muovere la camera o variare la posizione delle luci in modo indipendente dal modello che si stava utilizzando (vedi figura 4.3).

5.1.3 Materiali

Per quanto riguarda la gestione dei materiali questa è stata fatta utilizzando interamente le librerie messe a disposizione da BlenderProc [16].

Come prima cosa per ogni modello si è verificata la presenza di materiali e di conseguenza la presenza di una mappa UV. Nel caso in cui questa non fosse presente, se il modello era particolarmente complesso si è proceduto alla creazione manuale della mappa UV, in caso contrario si è sfruttata la funzione *Smart UV* offerta da Blender.

L'approccio usato per la gestione dei materiali è stato quello di assegnare preventivamente ad ogni oggetto non dei materiali specifici ma dei materiali "vuoti" che avessero solo il nome di una determinata "categoria" di materiali. Per esempio, nel caso di un tavolo le gambe potevano avere come materiale uno chiamato "*Metal*" mentre la parte superiore uno chiamato "*Wood*" o "*Plastic*". Così facendo tramite un apposito script è stato possibile leggere le "categorie" di materiali assegnate all'oggetto e in base a queste andare ad applicare uno specifico materiale, appartenente alla categoria letta, preso in maniera casuale da un dataset, appositamente creato, contenente solamente texture e materiali vari. In alternativa, era possibile fornire anche più di una categoria di materiali da cui scegliere assegnando all'oggetto un materiale contenente il nome di due o più categorie separate da "_". Per esempio "*Metal_Plastic*" prevedeva la possibilità di applicare o materiali metallici o materiali plastici (figura 5.1).

Tramite script è stata sviluppata inoltre la possibilità di far variare i materiali per ogni cambio di inquadratura ottenendo così dei render dello stesso oggetto ma ripreso da angolazioni differenti e con differenti materiali, oppure di mantenere sempre gli stessi materiali per ogni singolo render.

Il dataset utilizzato per la gestione dei materiali è stato ottenuto "unendo" due dataset contenenti materiali e texture, quello fornito da *Haven* [12] e quello fornito da *AmbientCG* [3]. Dal dataset ottenuto, per motivi di spazio, sono poi stati tolti tutti quei materiali che non sarebbero serviti per i nostri modelli come asfalto, cemento, ed altri ancora.
Il dataset finale contiene in totale circa 134 materiali comprendenti materiali come legno,

[2]In Blender un *empty* è un oggetto senza geometria aggiuntiva. Un empty non ha ne volume ne superficie e non può essere renderizzato. Tuttavia può essere utilizzato come oggetto "padre" a cui imparentare altri oggetti o per altri scopi ancora.

metalli, tessuti, pelle e plastica.

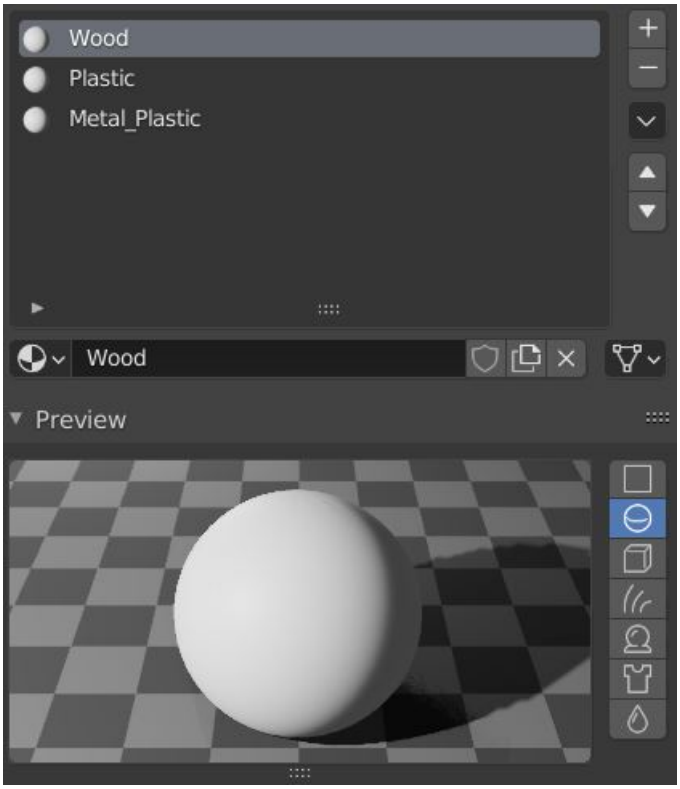

Figura 5.1: Esempio di gestione dei vari materiali di un modello

Tutto questo è stato possibile utilizzando in particolare la funzioni offerte da BlenderProc quali:

- ***materials.collectall()*** per l'acquisizione di tutti i materiali presenti in una scena.
- ***load_ccmaterilas()*** per il caricamento dei materiali da un dataset esterno.
- ***set_material()*** per l' applicazione dei nuovi materiali.

Inoltre, sfruttando le funzioni proprie di Blender è stato anche possibile, sempre tramite script, modificare il colore di alcuni materiali tramite l'uso di nodi *MixRGB*, avendo così un ulteriore grado di casualità nella scelta dei materiali.

5.1.4 Luci, Camera e Render

La gestione delle luci, della camera e dei render è stata eseguita interamente in maniera automatizzata sfruttando le funzioni di BlenderProc.

Per le luci, in particolare, tramite script si poteva sceglie quale delle due configurazioni viste precedentemente (figura 4.4) utilizzare ed in base alla scelta effetuata queste venivano poi inserite all'interno della scena.

Di seguito è riportato il codice utilizzato per la creazione delle luci per una illuminazione omogenea del modello:

```
1     # luci posizionate in (-1,-1,1) (1,-1,1) (1,1,1) (-1,1,1)
2
3     for x in [-1, 1]:
4         for y in [-1, 1]:
5             light = bproc.types.Light()
6             light.set_location([x, y, 1])
7             light.set_energy(100)
```

E quello usato per il metodo *three-point lighting*:

```
8     # Posizione della KeyLight, BackLight e FillLight
9     light_location = [(-2, -2, 2), (-1.5, 2, 2), (2, -2, 2)]
10
11    # Inclinazione delle luci
12    euler_angle = [
13    (45*(np.pi/180),0,-45*(np.pi/180)),
14    (-45*(np.pi/180),0,35*(np.pi/180)),
15    (45*(np.pi/180),0,45*(np.pi/180))]
16
17    # Potenza della KeyLight, BackLight e FillLight
18    energy = [(80),(10),(25)]
19
20    for loc in range(len(light_location)):
21        light = bproc.types.Light()
22        light.set_type(type="AREA")
23        light.set_location(light_location[loc])
24        light.set_rotation_euler(euler_angle[loc])
25        light.set_energy(energy[loc])
```

Per quanto riguarda la camera si è sviluppato un codice che le permettesse di "muoverla" attorno ad un determinato modello mantenendolo comunque all'interno dell'inquadratura. Per fare ciò lo script generava randomicamente un numero prestabilito di posizioni della camera, ottenute come coordinate punti *(x,y,z)* appartenenti alla superficie di una sfera come mostrato in figura 5.2. Agendo sui parametri per la creazione di questi punti come l'angolo di inclinazione, quello di rotazione o il raggio si poteva vincolare la camera ad

assume determinate posizioni.

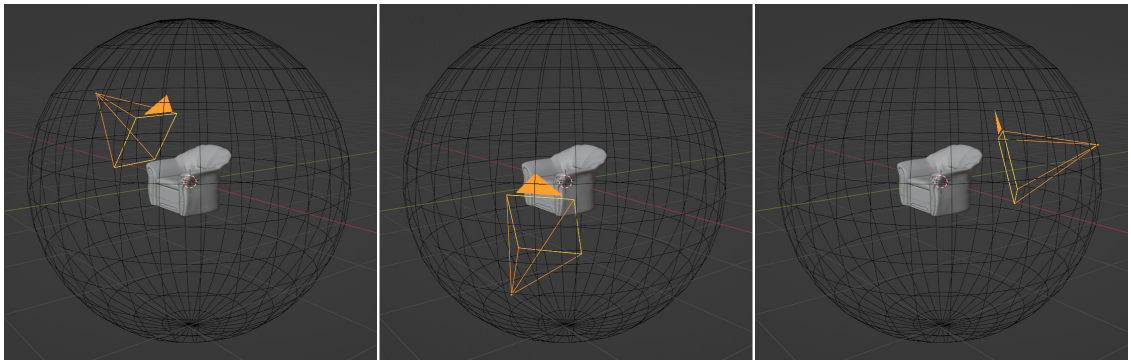

Figura 5.2: La camera si muove sopra la superficie di una sfera inquadrando sempre l'oggetto

Per ogni posizione in cui veniva spostata la camera veniva inoltre applicato un keyframe in modo tale da potere poi eseguire tutti i vari render in sequenza.

Infine, per la fase di render si è sfruttata solamente la funzione "*render*" di BlenderProc la quale fa partire un singolo render della scena se non sono presenti keyframe oppure un render per ogni keyframe. Gli unici parametri da impostare, in questo caso, sono stati le dimensioni delle immagini in output (quelle usate in questa tesi sono stati 128×128px, 224×224px, 512×512px e 1024×1024px), il formato di immagine in output settato come *.png*, ed infine abilitare la trasparenza del background.

5.1.5 Voxel

La generazione dei modelli in formato voxel è stata effettuata utilizzando un semplice tool chiamato *Binvox* [5].
Binvox è un semplice programma che dato in ingresso un modello 3D, lo "rasterizza" in una griglia voxel 3D binaria e successivamente salva il file "voxel" risultante (figura 5.3).
Si esegue tramite terminale specificando il percorso dei file di input, quello di output, la dimensione dell'output (dimensione della griglia 3D che conterrà il modello voxel), il formato di output ed altri parametri vari come la rotazione, lo scalamento ed altro ancora.
Binvox lavora con diversi formati in input ma non con file .blend quindi per prima cosa si è dovuto convertire i file .blend in un formato che andasse bene per Binvox.
Il formato scelto è stato il *.ply*. Per eseguire questa conversione si è sfruttata la libreria di Python Open3D [23] e tramite un semplice script si sono ottenuti tutti file necessari in formato .ply.
Avendo i file nel formato giusto si è proceduto alla conversione in formato voxel dei vari modelli.
Gli unici parametri da settare in Binvox sono stati la dimensione finale della griglia contenente il modello voxel, settata a 64×64×64 in quanto BSP-net lavora con questa dimensione, ed il formato di output scelto come "*raw*" *.npy* ossia il formato di output della libreria

NumPy [22] di Python.

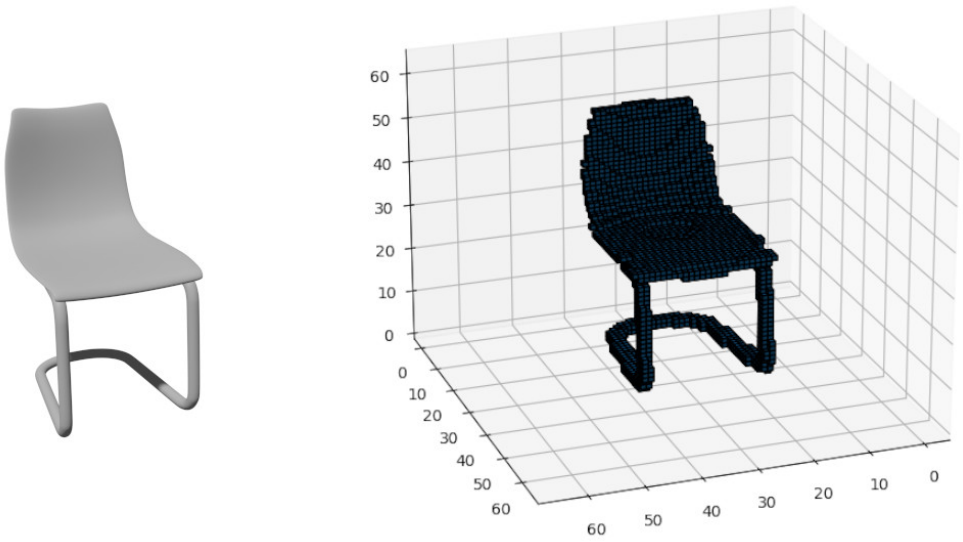

Figura 5.3: Modello di una sedia in formato voxel ottenuto grazie al tool Binvox

5.2 Rete Neurale

Dopo la parte relativa la creazione dei dataset si è passati alla gestione della rete neurale. La rete neurale utilizzata si basa sul modello proposto da BSP-net [39] e ne utilizza in larga parte lo stesso codice, consultabile sull'apposita pagina github.

Data la grande quantità di risorse di cui una rete neurale ha bisogno soprattutto in fase di training, BSP-net è stata utilizzata su una workstation contente una GPU Nvidia GeForce 1060 con 6GB di memoria dedicata, in grado di fornire alla rete una buona potenza di calcolo necessaria alle varie elaborazioni.

5.2.1 Paramentri della rete neurale

I parametri principali della rete neurale necessari a BSP-net per funzionare sono i seguenti:

- **Pesi dell'autencoder e dell'image encoder**. I "*pesi*" rappresentano un insieme di valori che la rete associa a ciascuna caratteristica estratta da un'immagine di input

durante un processo di training. In generale questi valori vengono generati durante la fase di training e successivamente salvati all'interno di specifici file creati dalla rete stessa. Questi parametri sono necessari in fase di ricostruzione perché contengono tutte le informazioni utili che la rete ha appreso durante il processo di training. Senza di questi la rete risulterebbe non allenata e quindi non in grado di funzionare. I pesi inoltre possono essere utili anche in fase di training in quanto se salvati periodicamente (per esempio ogni 10 o 100 epoch), in caso di crash del sistema, possono essere utilizzati come "backup" consentendo alla rete di continuare il training da dove è stato interrotto senza dover ricominciare tutto da capo.

- **Device da utilizzare**. Essendo il codice di BSP-net realizzato utilizzando il framework PyTorch è possibile scegliere quale risorsa hardware utilizzata per svolgere i calcoli come la CPU o la GPU. Nel caso di questa tesi, BSP-net è stata utilizzata sfruttando la GPU.

- **img_ef_dim**. Questo valore rappresenta il numero di canali in uscita per ogni livello di convoluzione. Di base è stato settato su 64 come nel caso di ResNet-18 [35].

- **z_dim**. Questo parametro riguarda la dimensione dell'array contente le feature estratte dai modelli in formato voxel e dalle immagini. In questo caso si è utilizzata una dimensione pari a 256.

- **Learning rate**. Il tasso di apprendimento o learning rate definisce quanto velocemente procederà l'addestramento della rete. Con un valore alto, l'errore di training scenderà più velocemente ma di contro più alto è il valore del learning rate, più alto è il rischio che dei dettagli vengano trascurati. Il valore utilizzato in questo caso è stato quello proposto dagli autori di BSP-net ossia 0,0001.

- **Batch_size**. Dato che il dataset usato per il training potrebbe essere troppo grande per essere elaborato tutto in una volta, si può pensare suddividerlo in sottogruppi uniformi, chiamati batch. Il batch_size rappresenta le dimensioni di questi sottogruppi. In base alla dimensione del batch si avranno quindi un determinato numero di iterazioni per eseguire un training completo su tutto il dataset (epoch). Per esempio, se si usasse un batch size di 20 per un dataset di 1000 elementi sarebbero necessarie 50 iterazioni per analizzare l'intero dataset, cioè per completare un epoch.
  In linea di massima il batch_size influisce soprattutto sulla velocità di training della rete. Bisogna comunque fare attenzione al fatto che, se si sfrutta l'accelerazione GPU, e il batch_size fosse troppo grande si potrebbero avere problemi di esaurimento della memoria.
  Di base non esiste una regola fissa per la scelta del batch_size, ma dipende dal singolo caso e da diversi fattori quali la dimensione del dataset, la disponibilità di memoria e perfino la tipologia di rete neurale. Batch size tipici sono 32, 64 o 128 (solitamente potenze di 2 per motivi di allocazione della memoria). BSP-net usa un batch size pari a 64 ma nel caso di questa tesi il valore utilizzato è stato 16 in quanto la GPU utilizzata non era in grado di allocare abbastanza memoria usando valori maggiori.

5.2.2 Estrazione dei predittori

Come detto in precedenza BSP-net sfrutta una doppia fase di training, una per l'autoencoder ed una per l'image encoder. Nella fase di training dell'autencoder, infatti, la rete impara a riconoscere, mediante l'estrazione di particolari feature, e a ricostruire oggetti a partire da semplici modelli in formato voxel. Le feature ottenute in questa fase vengono poi utilizzate per il training dell'image endcoder.

Nei test effettuati in questa tesi però non è stato effettuato il training dell'autoencoder ma, dato che insieme al codice di BSP-net vengono forniti i pesi relativi ad un modello di autoencoder pre-allenato su migliaia di modelli, si è deciso di utilizzare questi pesi e di saltare tutta la prima fase di training.
In particolare, per quanto riguarda le classi di oggetti scelte in questo studio per effettuare i vari test, l'autoencoder fornito dagli autori di BSP-net risulta essere allenato su un dataset contente 5422 modelli 3D di sedie, 6807 modelli 3D di tavoli e 1256 modelli 3D di armadi/librerie. Quindi dato che i pesi dell'autoencoder dati erano stati ottenuti su una grande quantità di modelli dopo una intensa fase di training, e che quindi la rete era già in grado di riconoscere ed estrarre delle feature da migliaia di modelli 3D si è deciso di utilizzare questi pesi e di non effettuare un nuovo training dell'autoencoder.

Avendo dunque l'autoencoder "già allenato" si è passati all'estrazione dei "predittori". In particolare, sfruttando la funzione *encode_voxel()*, creata partendo dalla funzione *get_z()* di BSP-net, la rete è stata in grado di fornire per ogni modello voxel in input un array contenete delle feature utili alla ricostruzione del modello 3D.
Infatti, fornendo come input alla rete questi predittori essa sarà in grado di ricostruire il corrispondente modello 3D come mostrato in figura 5.4.

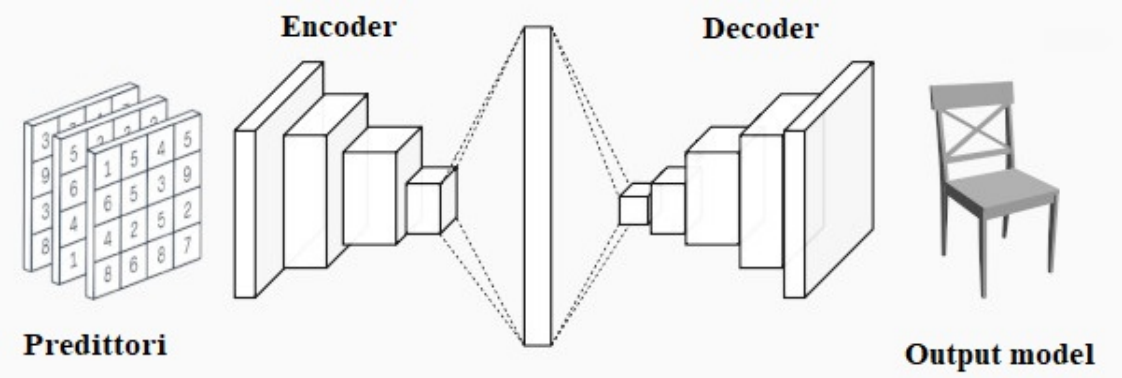

Figura 5.4: Ricostruzione di un modello 3D a partire dai predittori ottenuti dall'autoencoder di BSP-net

In questo caso come prova che l'autoencoder fornito dagli autori di BSP-net funzionasse, una volta estratti i predittori di tutti i modelli 3D appartenenti alle classi scelte si è subito proceduto alla ricostruzione di questi modelli a partire dai predittori verificando, tramite il calcolo della Chamfer Distance, che i modelli ottenuti fossero molti simili a quelli originali. I risultati ottenuti sono riportati in tabella 5.1.

Classi di modelli	**Chamfer Distance**
Sedie	0.00120928
Tavoli	0.00110767
Armadi e Librerie	0.00156328

Tabella 5.1: Chamfer Distance media per ogni classe di modelli scelta calcolata tra i modelli ricostruiti a partire dai predittori estratti usando l'autoencoder pre-allenato fornito dagli autori di BSP-net ed i modelli "originali"

5.2.3 Training Image Encoder

Avendo estratto i predittori dai modelli in formato voxel si è passati poi al training dell'image encoder di BSP-net.

Prima di poter effettuare il training però è stato necessario effettuare alcuni aggiustamenti al codice di BSP-net. In particolare, come evidenziato in precedenza, questa rete è stata progettata e sviluppata per lavorare con immagini in input di dimensione 128×128px, in scala di grigi ad un solo canale (L). Quindi, per far fronte a questo limite è stato studiato il codice di BSP-net per capire come e cosa fare per poter utilizzare come input immagini con dimensione variabile e a più canali.
La soluzione al problema è stata abbastanza semplice e veloce da trovare. In particolare, gli autori di BSP-net nel loro studio [39] specificano che il modello su cui si basa la loro rete BSP-net è quello di ResNet-18 ma "senza utilizzare livelli di polling". Quindi BSP-net, nella sua versione originale sfrutta solamente dei livelli convoluzionali per ridurre le dimensioni delle varie feature map in modo tale da arrivare ai livelli completamente connessi con una feature map rappresentata da un tensore di dimensione 1×1. Per riuscire a fare questo utilizzando lo stesso numero di livelli e la stessa tipologia di convoluzioni usate da ResNet-18, BSP-net necessita dunque di avere come input immagini di dimensione pari a 128×128 pixel.
Una soluzione a questo problema è stata quella di aggiungere semplicemente i due livelli di pooling presenti in ResNet-18 e mancanti in BSP-net. Così facendo la rete è stata in grado

di gestire immagini di differenti dimensioni.
Per quanto riguarda il numero di canali invece è bastato solamente agire sul primo livello convoluzionale cambiando il parametro relativo ai canali in input da 1 a 3.

Di seguito è riportato il codice originale di BSP-net con le modifiche apportate per consentire la gestione di immagini a differenti risoluzioni ed immagini a colori.

```
26 class ImgEncoder(nn.Module):
27     def __init__(self, img_ef_dim, z_dim):
28         super(ImgEncoder, self).__init__()
29         self.img_ef_dim = img_ef_dim
30         self.z_dim = z_dim
31 
32         # Cambiato numero di canali in input da 1 a 3
33         self.conv_0 = nn.Conv2d(3, self.img_ef_dim, 7, stride=2,
               padding=3, bias=False)
34 
35         # Aggiunto MaxPool2D
36         self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2,
               padding=1)
37 
38         self.res_1 = resnet_block(self.img_ef_dim,
               self.img_ef_dim)
39         self.res_2 = resnet_block(self.img_ef_dim,
               self.img_ef_dim)
40         self.res_3 = resnet_block(self.img_ef_dim,
               self.img_ef_dim*2)
41         self.res_4 = resnet_block(self.img_ef_dim*2,
               self.img_ef_dim*2)
42         self.res_5 = resnet_block(self.img_ef_dim*2,
               self.img_ef_dim*4)
43         self.res_6 = resnet_block(self.img_ef_dim*4,
               self.img_ef_dim*4)
44         self.res_7 = resnet_block(self.img_ef_dim*4,
               self.img_ef_dim*8)
45         self.res_8 = resnet_block(self.img_ef_dim*8,
               self.img_ef_dim*8)
46 
47         # Aggiunto AdaptiveAvgPool2D
48         self.avgpool = nn.AdaptiveAvgPool2d((1,1))
49 
50         self.linear_1 = nn.Linear(self.img_ef_dim*8,
               self.img_ef_dim*8, bias=True)
51         self.linear_2 = nn.Linear(self.img_ef_dim*8,
               self.img_ef_dim*8, bias=True)
52         self.linear_3 = nn.Linear(self.img_ef_dim*8,
               self.img_ef_dim*8, bias=True)
53         self.linear_4 = nn.Linear(self.img_ef_dim*8, self.z_dim,
               bias=True)
```

Avendo provveduto a sistemare il codice di BSP-net adattandolo al caso specifico di questa tesi si è proceduto con il training della rete neurale utilizzando i dataset generati per le varie analisi da effettuare.
Per quanto riguarda l'allenamento della rete, di base si è deciso di eseguire un training per 1000 epoch ma osservando che comunque si ottenevano valori di training molto bassi ancora prima di raggiungere le 1000 epoch per evitare fenomeni di *overfitting* [3] della rete neurale si è implementata una funzione di "*early stopping*" (arresto anticipato) che terminasse il training o quando l'errore di training scendesse al di sotto di 10^{-5} oppure se il training error non decrescesse più per oltre 50 epoch.

Finita la fase di training la rete salvava in automatico un file contente i pesi e lo stato del modello da usare poi per la fase di ricostruzione.

5.2.4 Ricostruzione di oggetti 3D e calcolo della Chamfer Distance

Terminata la fase di training la rete era pronta per la ricostruzione dei modelli 3D.
Per la ricostruzione dei vari modelli sono state utilizzate le immagini "reali" provenienti dal dataset di Pix3D (figura 5.5). Per ogni oggetto del datatset di Pix3D dono state prese casualmente 3 immagini e da queste tre sono stati ricostruiti tre modelli 3D dell'oggetto.

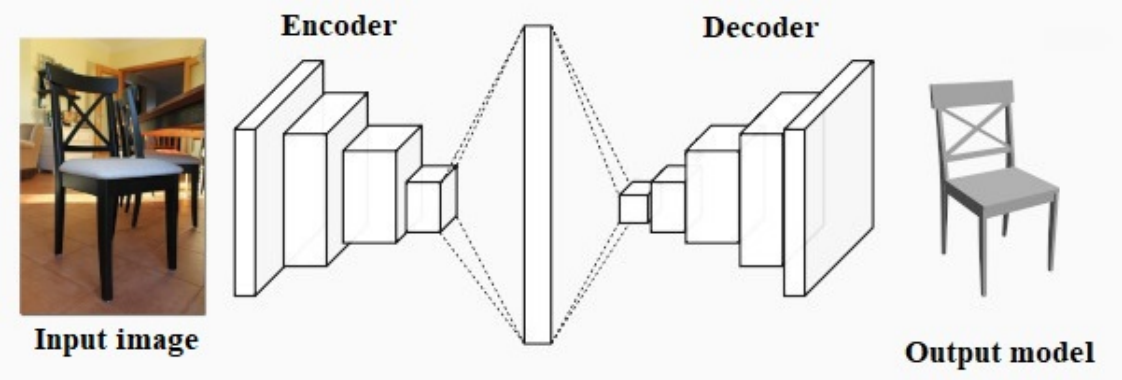

Figura 5.5: Ricostruzione di un oggetto 3D a partire da una singola immagine "reale" in input

Gli oggetti sono stati ricostruiti direttamente come nuvole di punti per facilitare il lavoro relativo al calcolo della Chamfer Distance. Avendo così i modelli come nuvole di punti si è passato al calcolo della Chamfer Distance tra i modelli ricostruiti e i modelli ground

[3] Overfitting: fenomeno che si verifica quando un modello si adatta ai dati su cui sta effettuando il training con il risultato che la rete non sarà in grado di generalizzare fornendo dunque previsioni accurate solo per dati molto simili a quelli di addestramento.

truth di Pix3D (figura 5.6).

Per il calcolo della Chamfer Distance è stato creato uno script in Python il quale sfruttava alcune funzioni offerte delle librerie Scikit-learn [31] e PyTorch [27] di Python per semplificare i calcoli necessari.
Una volta ottenuti i valori di Chamfer Distance per ogni modello questi sono stati inseriti in un foglio di calcolo dove hanno subito tutto il processo di analisi.

Figura 5.6: (a) Nuvola di punti di un modello ricostruito e (b) del corrispondente modello "ground truth" utilizzate nel calcolo della Chamfer Distance. (c) Sovrapposizione delle due nuvole di punti per poter visualizzare le differenze tra i due modelli.

Il lavoro di estrazione dei predittori, training dell'image encoder, ricostruzione e calcolo della Chamfer Distance è stato poi ripetuto per ogni singolo dataset di immagini sintetiche da testare.

Capitolo 6

Esperimenti e risultati ottenuti

6.1 Scelta dei modelli

La scelta delle categorie di oggetti da utilizzare per la generazione dei dataset sintetici è stata fatta sulla base di alcuni vincoli legati in particolare a tre aspetti fondamentali:

- Categorie di oggetti utilizzati in diversi studi e quanto visto in letteratura.
- Scelta della rete neurale da utilizzare per la ricostruzione degli oggetti 3D
- Come effettuare una precisa analisi della ricostruzione di un oggetto 3D

In generale, in base a quanto trovato in letteratura, la maggior parte dei dataset di immagini sintetiche utilizzati in vari studi offrono sempre delle categorie "standard" di oggetti tra le quali: sedie, tavoli, armadi, librerie, fucili, auto, aerei, letti e barche. Tra i vari dataset utilizzati in altre ricerche, Shapenet [38] è il dataset che ne offre di più con circa 270 categorie di oggetti. Analizzando nello specifico studi riguardanti reti neurali che si occupano di ricostruzione di oggetti 3D si è visto che anche in questo caso le categorie di oggetti utilizzati erano sempre più o meno le stesse. Questo perché spesso ogni nuovo studio va a confrontare i propri risultati con quelli ottenuti da studi precedenti e quindi si tende sempre ad utilizzare le stesse categorie di oggetti. In particolare, quelle più utilizzate in assoluto nei vari studi sono sedie, aerei ed automobili. Quindi partendo da questa prima analisi si è deciso di concentrarsi su dei modelli standard già utilizzati in altre ricerche evitando invece oggetti molto specifici.

Per quanto riguarda la rete neurale, dato che si era deciso di adottare il modello di BSP-net [39] perché in grado di ricostruire direttamente una mesh 3D di un oggetto, e data anche l'intensione di utilizzare l'autoencoder pre-allenato fornito dagli sviluppatori, si è dovuta limitare la scelta delle categorie a quelle sulle quali era stato allenato l'autoencoder. In particolare, l'autoencoder di BSP-net utilizzato in questa ricerca risulta essere stato allenato su 13 categorie di oggetti quali: sedie, tavoli, armadi/librerie, divani, aerei, automobili, fucili, lampade, display, telefoni, panchine, barche e speaker.

Infine, volendo utilizzare delle immagini reali di oggetti come input per la ricostruzione di un modello 3D e la conseguente scelta di utilizzare il dataset Pix3D [10] che ci dava la possibilità di avere contemporaneamente dei modelli 3D, da usare come gruond truth per testare la qualità della ricostruzione di un oggetto, che delle corrispondenti immagini reali da utilizzare come input si è dovuta effettuare una ulteriore cernita delle possibili categorie da utilizzare.
In particolare, le possibili categorie da utilizzare si sono infine ridotte a tutte quelle categorie che fossero presenti nel dataset Pix3D e contemporaneamente sulle quali fosse stato allenato l'autoencoder di BSP-net. Quindi alla fine le possibili categorie da scegliere risultavano essere: sedie, tavoli, armadi, librerie e divani. Si è deciso dunque di utilizzare sedie, tavoli e librerie/armadi e di scartare i divani per concentrarci su categorie di oggetti con forme molto diverse tra loro.

6.2 Esperimenti effettuati

6.2.1 Risoluzione delle immagini

I primi esperimenti effettuati hanno riguardato principalmente la risoluzione delle immagini utilizzati per il training della rete neurale. Le dimensioni scelte da testare sono state le seguenti: 128×128px, 224×224px, 512×512px e 1024×1024px (figura 6.1).

Figura 6.1: Immagine sintetica di una sedia con risoluzione 128×128px, 224×224px, 512×512px e 1024×1024px

La scelta di queste dimensioni è stata fatta analizzando vari studi riguardo reti neurali che utilizzano dataset di immagini sintetiche per il training. In particolare, si è visto che la maggior parte delle reti neurali utilizza per il training immagini di dimensione 128×128px quindi questa risoluzione è stata utilizzata come base di partenza per le nostre analisi. Partendo da una risoluzione di 128×128px si è proceduto poi ad effettuare degli esperimenti

con risoluzioni maggiori per capire se effettivamente immagini più grandi potessero fornire maggiori informazioni utili alla rete per la ricostruzione di un oggetto 3D.
Per quanto riguarda nello specifico la risoluzione 224×224px questa è stata scelta perché è la risoluzione su cui lavora ResNet-18 e dato che il modello utilizzato in questa tesi si rifà al modello di ResNet-18 è sembrato utile effettuare delle valutazioni anche su questa specifica risoluzione delle immagini.
Per ogni categoria di oggetti sono stati creati quattro dataset di immagini sintetiche uguali in tutto (stessi materiali, posizione delle luci e della camera) tranne che per la risoluzione delle immagini in modo tale da evitare che altri parametri andassero ad influire poi sui risultati della ricostruzione.

6.2.2 Distanza della camera

Un altro parametro analizzato riguarda la posizione della camera utilizzata per il render dei modelli 3D dei vari oggetti. In particolare, si è posta l'attenzione riguardo l'effettiva distanza della camera da un modello 3D.
Dato che tutti i dataset analizzati presentavano sempre delle immagini in cui la camera si muove attorno all'oggetto ma rimane sempre alla stessa distanza si è pensato di generare dei dataset di immagini in cui la camera oltre a muoversi attorno all'oggetto cambiasse anche la sua distanza da esso in modo del tutto casuale (figura 6.2).
Anche in questo caso sono stati generati due dataset di immagini per ogni categoria di oggetto in cui l'unica differenza stava nella distanza della camera nelle varie immagini e si è poi proceduto con i vari test.

Figura 6.2: Render con camera a distanza fissa (in alto) e render con distanza della camera che varia (in basso)

6.2.3 Materiali e Texture

Per quanto riguarda l'analisi dei vari materiali e le texture applicate ai modelli 3D si è proceduto ad eseguire diversi esperimenti. In primo luogo, si è cercato di capire quanto questi influissero sulla ricostruzione ed in particolare quanto la presenza o meno di un materiale influisse sul risultato finale. Per fare ciò si è deciso di generare, per ogni categoria di oggetti, due dataset uno contenente immagini di oggetti aventi dei materiali applicati e un altro con immagini di oggetti senza materiali (figura 6.3).

Figura 6.3: Render di un modello con materiale e render dello stesso modello senza materiale

Dopo i primi test si è passati ad una analisi più dettagliata riguardante più nello specifico le varie texture ed i vari materiali. In particolare dato che i vari materiali utilizzati erano formati da un insieme di texture rappresentanti le varie "mappe" (figura 6.4) utilizzate per la generazione di determinati effetti in fase di render, come delle normal map per simulare deformazioni della superficie o delle roughness map per gli effetti di opacità o riflessione, si è proceduto a generare dei dataset in cui di volta in volta veniva rimossa una singola mappa fino ad vere solamente una texture con solo il colore del materaile. Così facendo si è cercato di capire quanto la presenza o meno di queste texture, utilizzate per la creazione di effetti in fase di render, potesse incidere sul buon funzionamento della rete.

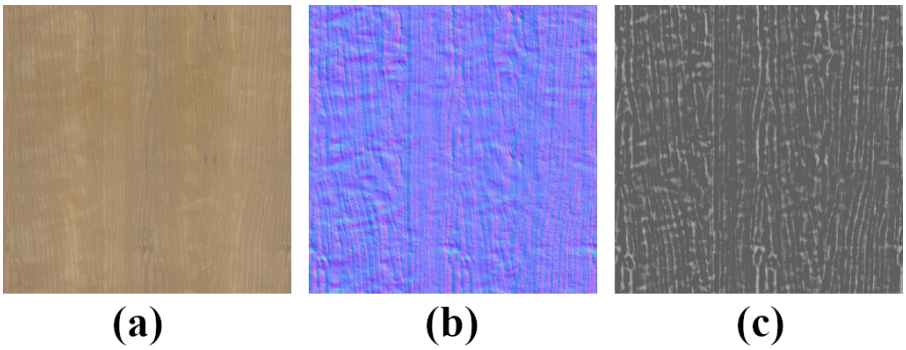

Figura 6.4: (a) Color map, (b) Normal map, (c) Roughness map

Un ulteriore analisi è stata fatta per cercare di capire se vi fossero determinati materiali "migliori" da utilizzare per i modelli 3D rispetto a ad altri e in questo caso un parametro scelto è stata la presenza di texture aventi determinati pattern (figura 6.5) che potessero impattare negativamente soprattutto nella fase di training della rete neurale.

Figura 6.5: Esempio di materiale con pattern che potenzialmente potrebbe incidere in maniera negativa sul training della rete neurale

Infine, come ultima analisi, si è provato ad allenare la rete utilizzando delle immagini in scala di grigi al posto di immagini a colori. Anche in questo caso per ogni dataset che veniva generato l'unico parametro delle immagini a variare era solo il materiale applicato all'oggetto.

6.2.4 Illuminazione

Anche l'impatto che l'illuminazione dell'oggetto poteva avere sulle prestazioni della rete neurale è stato analizzato.
In particolare, sono stati definiti due modelli di illuminazione da utilizzare in fase di creazione del dataset. Uno era una configurazione di luci disposte in modo tale da illuminare in maniera omogenea l'intero modello 3D cercando di ridurre il più possibile le zone d'ombra, mentre nella seconda configurazione le luci sono state secondo il modello three-point lighting con due luci poste di fronte il modello 3D ed una dietro in modo tale da simulare un effetto più realistico grazie anche alla presenza di ombre.

Un ulteriore analisi è stata poi fatta analizzando invece quanto la variazione di intensità delle luci influisse sul risultato finale. In questo caso, infatti, a differenza dei primi due in cui le luci avevano sempre un'intensità fissa, è stato creato un dataset dove per ogni singola immagine l'intensità della luce era differente avendo come risultato delle immagini più "scure" ed altre più "luminose".

6.3 Riassunto dei risultati ottenuti

Di seguito sono riportati i risultati ottenuti nei vari esperimenti per le categorie di oggetti analizzati.

6.3.1 Dimensioni

I primi esperimenti effettuati hanno riguardato le dimensioni delle immagini utilizzate per il training della rete neurale. In particolare, per questi primi test sono stati testati tre dataset di immagini:

1. Il primo conteneva delle immagini sintetiche ottenute mantenendo la camera ad una distanza fissa dal modello 3D.
2. Il secondo conteneva, invece, delle immagini ottenute applicando ai vari modelli 3D gli stessi materiali usati per il primo dataset ma facendo variare la distanza della camera dal modello.
3. Il terzo invece conteneva delle immagini ottenute utilizzando le pose della camera del secondo dateset ma ai modelli 3D sono stati rimossi tutti i material.

Un esempio di questi tre dataset utilizzati è mostrato in figura 6.7.

Per effettuare i test rispetto alla risoluzione delle immagini, ognuno dei tre dataset è stato generato con immagini a risoluzione: 128×128px, 224×224px, 512×512px e 1024×1024px.

Avendo generato tutti i dataset necessari si è proceduto ad effettuare i vari training della rete neurale per le varie risoluzioni. Si è passati poi alla ricostruzione dei modelli 3D a partire dalle immagini reali di Pix3D ed infine all'analisi della somiglianza rispetto ai modelli originali attraverso la valutazione della Chamfer Distance.
I risultati ottenuti utilizzando il primo dataset sono riportati in tabella 6.1, quelli relativi al secondo dataset in tabella 6.2 e quelli relativi al terzo dataset in tabella 6.3.

	Chamfer Distance			
	128×128px	**224×224px**	**512×512px**	**1024×1024px**
sedie	0,01216544 0,0122	0,01096158 0,0110	0,01085494 0,0109	0,01051179 0,0105
tavoli	0,01700497 0,0170	0,01533876 0,0153	0,01512623 0,0151	0,01291376 0,0129
librerie e armadi	0,01532416 0,0153	0,01773648 0,0177	0,01763111 0,0176	0,01199550 0,0120

Tabella 6.1: Valori di Chamfer Distance ottenuti utilizzando il primo dataset di immagini sintetiche (arrotondati all'ottava ed alla quarta cifra decimale)

	Chamfer Distance			
	128×128px	**224×224px**	**512×512px**	**1024×1024px**
sedie	0,00533475 0,0053	0,00473552 0,0047	0,00459662 0,0046	0,00421961 0,0042
tavoli	0,01497073 0,0150	0,01401991 0,0140	0,01378746 0,0138	0,01354580 0,0135
librerie e armadi	0,00714980 0,0071	0,00516797 0,0052	0,00396632 0,0040	0,00403917 0,0040

Tabella 6.2: Valori di Chamfer Distance ottenuti utilizzando il secondo dataset di immagini sintetiche (arrotondati all'ottava ed alla quarta cifra decimale)

	Chamfer Distance			
	128×128px	**224×224px**	**512×512px**	**1024×1024px**
sedie	0,00607146 0,0061	0,00546857 0,0055	0,00522626 0,0052	0,00523146 0,0052
tavoli	0,01449611 0,0143	0,01503572 0,0150	0,01465384 0,0147	0,01700933 0,0170
librerie e armadi	0,00832706 0,0083	0,00873554 0,0087	0,00864010 0,0086	0,00932117 0,0093

Tabella 6.3: Valori di Chamfer Distance ottenuti utilizzando il terzo dataset immagini sintetiche (arrotondati all'ottava ed alla quarta cifra decimale)

Analizzando in dettaglio i risultati ottenuti per le varie dimensioni si può osservare che, nella maggior parte dei casi, all'aumentare della risoluzione il valore della Chamfer Distance diminuisce e quindi, in media, la ricostruzione dell'oggetto 3D risulta essere migliore. In particolare, passando da una risoluzione di 128×128px ad una di 224×224px il valore della Chamfer Distance "migliora" di circa il 9% per le sedie ed i tavoli. Mentre, aumentando ancora le dimensioni e passando da una risoluzione di 224×224px ad una di 512×512px la Chamfer Distance comincia ad essere un po' più eterogenea "migliorando" dallo 0,3% al 4,4% rispetto alla risoluzione di 224×244px. Passando invece da una risoluzione di 512×512px ad una di 1024×1024px i valori di Chamfer Distance ottenuti variano di molto tra i vari test effettuati e addirittura in alcuni casi la ricostruzione potrebbe essere peggiore come nel caso dei test effettuati con modelli senza materiali applicati. Il motivo in questo caso potrebbe essere legato alla progettazione della rete stessa. In particolare, ResNet-18 di base è progettata per lavorare con immagini di dimensioni di 224×224px; utilizzando tale dimensione, infatti, arrivati al layer di pooling finale, prima dei livelli completamente connessi, si ha una matrice delle feature di dimensioni 7×7 che tramite una operazione di *adaptive average pool* verrà ridotta ad un "vettore unitario" (questo perché i livelli completamente connessi lavorano solo su "vettori" ad una dimensione). Al crescere della dimensione delle immagini di input cresce anche la matrice delle feature finale avendo quindi una matrice di dimensioni 16×16 per immagini di risoluzione 512×512px e 32×32 per immagini di risoluzione 1024x1024px. Aumentando le dimensioni in input quindi si

avranno più feature ma poi quando verrà effettuato l'average pool la presenza di queste feature in più non sempre potrebbe portare risultati migliori (figura 6.6).

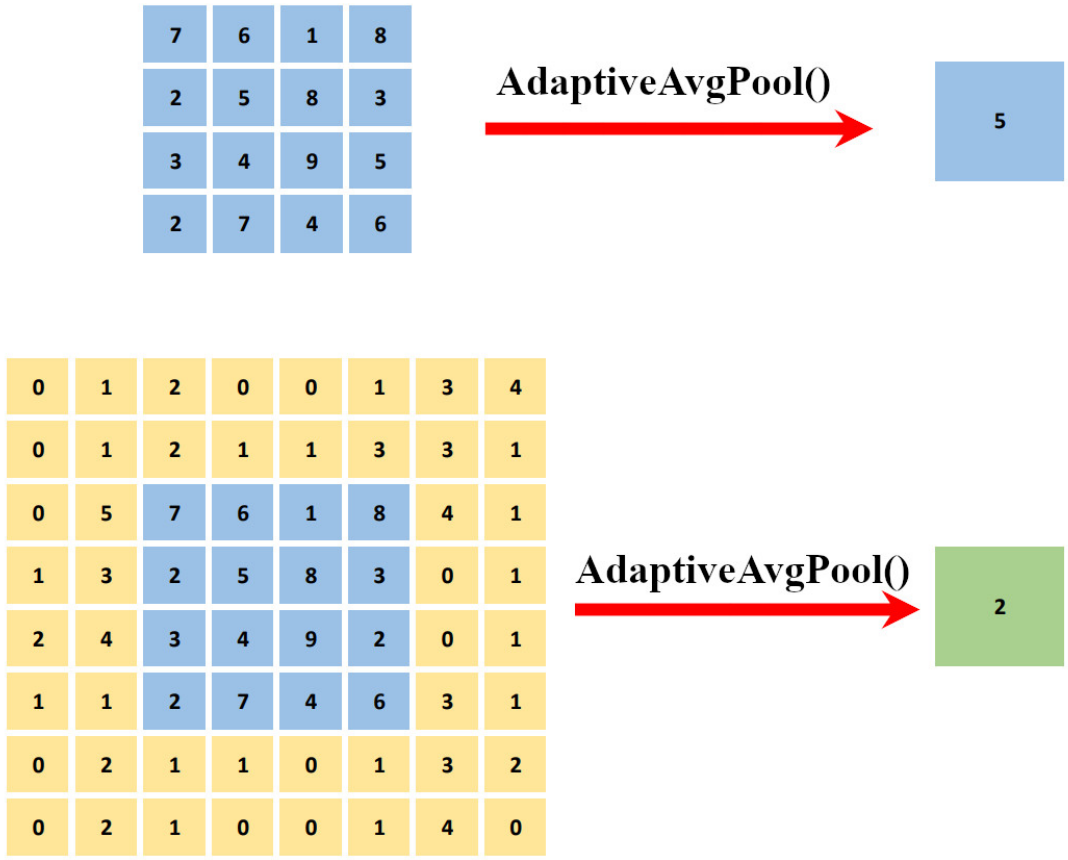

Figura 6.6: Esempio di Adaptive Average Pool effettuato su matrici di dimensioni differenti. Come si può notare il risultato potrebbe variare al crescere delle dimensioni della matrice

Un altro motivo che potrebbe spiegare il peggioramento della Chamfer Distance per la risoluzione 1024×1024px potrebbe essere quello che le immagini reali forniti da Pix3D hanno risoluzione variabile e spesso minore di 1024×1024px. In questo caso quindi le immagini vengono prima riscalate e ciò potrebbe comportare una perdita di qualità dell'immagine stessa utilizzata come input per la rete che ne andrebbe a compromettere il risultato della ricostruzione.

In generale, comunque, si è visto che aumentando le dimensioni delle immagini i risultati tendono ad essere migliori confermando in parte quanto visto in altri studi [30][34][32][18].

Figura 6.7: Tipologie di immagini presenti (a) nel primo dataset, (b) nel secondo dataset e (c) nel terzo

Un discorso a parte va fatto per gli armadi e le librerie. In questo caso i valori ottenuti sembrano essere leggermente differenti da quelli di sedie e tavoli. Questo è dovuto principalmente a due fattori:

- L'autoencoder di BSP-net. In particolare, a differenza di sedie e tavoli, per quanto riguarda le librerie e gli armadi l'auto-encoder di BSP-net risulta essere allenato "solo" su circa 1200 modelli (mentre per sedie e tavoli sono oltre 5000 ciascuno). Questo fatto influisce molto in fase di training in quanto potenzialmente la rete non sarà in grado di generalizzare abbastanza e quindi si avranno delle ricostruzioni meno accurate.
- Le immagini di Pix3D. Un altro fattore che potrebbe spiegare questa differenza tra i valori ottenuti con sedie e tavoli e quelli ottenuti con librerie e armadi è dato dal fatto che anche in questo caso Pix3D offre solamente 27 modelli di librerie o armadi, troppo pochi per valutare l'effettiva capacita di generalizzazione della rete neurale utilizzata.

Unendo insieme questi due fattori si potrebbe spiegare il motivo del perché i risultati dei test effettuati sulle librerie e sugli armadi siano un po' più eterogenei.

Di seguito sono riportate in figura 6.8 e in figura 6.9 alcuni esempi di ricostruzione di oggetti 3D per le varie risoluzioni testate.

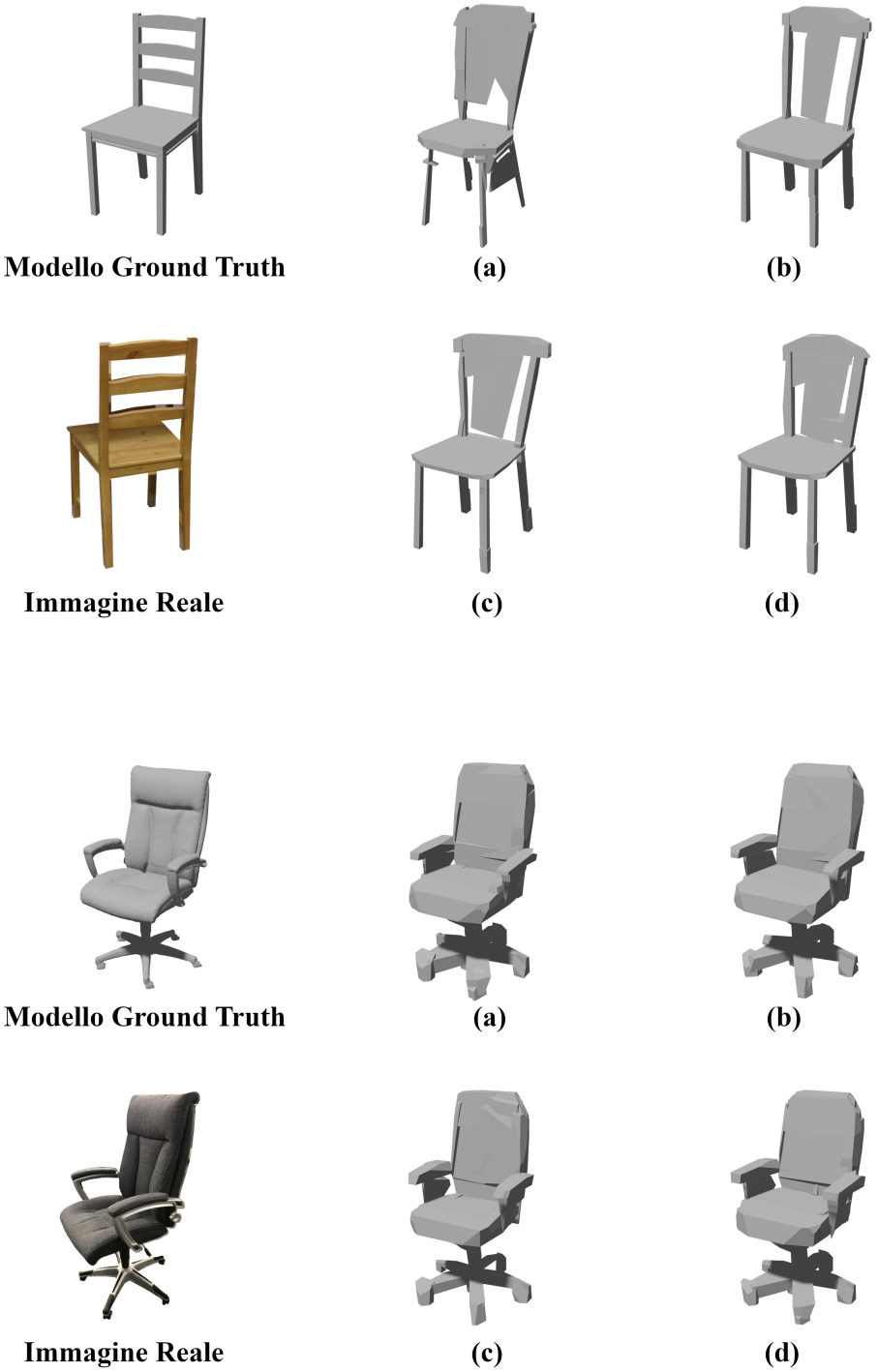

Figura 6.8: Esempio di sedie ricostruite dalla rete neurale allenata tramite l'utilizzo di dataset di immagini sintetiche con risoluzione: (a) 128×128px, (b) 224×224px, (c) 512×512px e (d) 1024×1024px

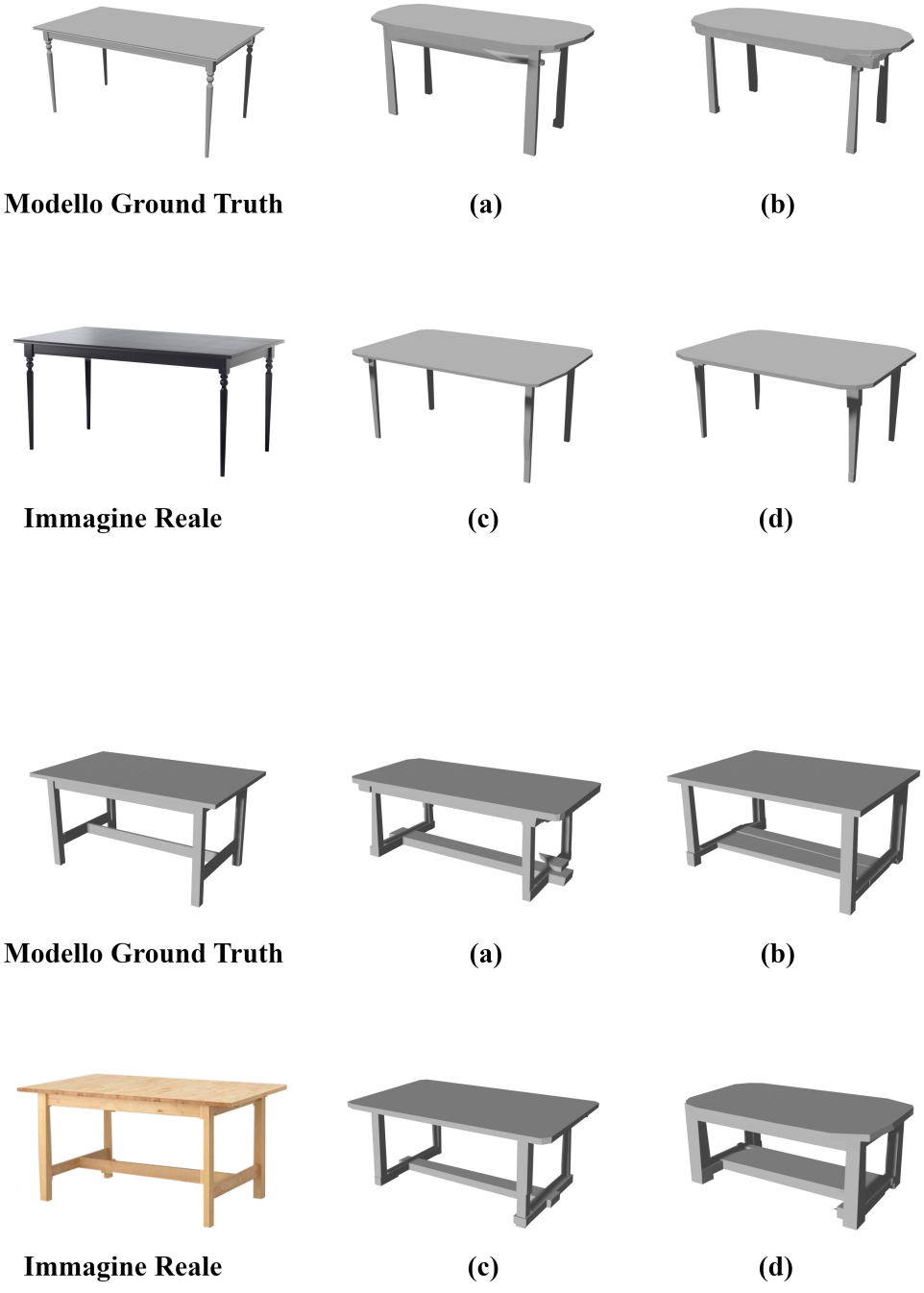

Figura 6.9: Esempio di tavoli ricostruiti dalla rete neurale allenata tramite l'utilizzo di dataset di immagini sintetiche con risoluzione: (a) 128×128px, (b) 224×224px, (c) 512×512px e (d) 1024×1024px

Per quanto riguarda invece le ottimizzazioni legate ai tempi di generazione dei dataset e di training della rete si è osservato che, per la generazione dei dataset sintetici le immagini a dimensione minore richiedono molto meno tempo di render rispetto a quelle a dimensione maggiore, come è logico che fosse.
Per la creazione dei dataset utilizzati in questa ricerca, contenenti ognuno 1000 immagini i tempi di render variavano da circa 25/30 minuti per immagini con risoluzione di 128×128px e 224×224px, poco meno di 2 ore per immagini di dimensioni di 512×512px ed oltre 4 ore per immagini di dimensione 1024×1024px.

Per quanto riguarda il tempo di training invece, utilizzando immagini di dimensioni 128×128px o 224×224px questo è stato di circa 5 secondi per epoch per un totale di circa 25/30 minuti necessari per completare una sessione di training. Aumentando le dimensioni invece il tempo per epoch cresce a circa 17/18 secondi per immagini di dimensioni di 512×512px avendo dunque sessioni di training della durata di circa 3 ore. Per dimensioni di 1024×1024px invece il tempo di training per epoch è stato di circa 68 secondi con delle durate totali delle sessioni di training maggiori di 12 ore. In particolare, il numero di epoch necessarie per terminare una sessione di training nei primi tre esperimenti è stato in media di 260 per immagini di dimensione 128×128px, 350 per immagini di dimensione 224×224px, 545 per immagini di dimensione 512×512px e di 708 per immagini di dimensioni 1024×1024px. I risultati ottenuti sono riportati in tabella 6.4.

	Tempi di training medi			
	128×128px	**224×224px**	**512×512px**	**1024×1024px**
tempo per singola epoch (secondi)	5,4	5	17	68
numero medio di epoch	260	350	545	709
Tempo totale di training (minuti)	24,27	29,17	154,41	803,53

Tabella 6.4: Tempi medi di training per i tre esperimenti riguardanti le dimensioni delle immagini

Nel caso specifico del dataset contenente immagini sintetiche di modelli senza materiali i tempi di render sono risultati leggermente inferiori mentre i tempi di training sono risultati leggermente superiori a quelli degli altri due esperimenti questo potrebbe confermare che la mancanza di materiali applicati ai modelli non aiuta la rete in quanto necessita di più tempo per il training.

Va sottolineato comunque che sia il tempo di render che il tempo di training per singola epoch dipendono soprattutto dalle risorse hardware della macchina utilizzata ma anche dal numero di altri programmi che sono in esecuzione nel momento in cui si stanno effettuando le operazioni di render o di training della rete, risultando quindi altamente variabili. Dunque, i risultati ottenuti sono da considerarsi come dei “tempi medi” su tutti i test effettuati. Durante lo svolgimento degli esperimenti si è cercato comunque, dove possibile,

di mantenere in esecuzione solo i programmi strettamente necessari alla ricerca evitando di eseguire ulteriori applicazioni per tutto il tempo in cui si effettuavano i render per il dataset o il training della rete neurale.
In particolare, per quanto riguarda la parte hardware, il computer utilizzato per la creazione dei dataset e per l'implementazione della rete neurale era equipaggiato con una scheda grafica Nvdia GeForce GTX 1060 con 6Gb di memoria dedicata che è stata utilizzata sia per i render che per i calcoli che venivano fatti dalla rete neurale per il training.

Avendo questi primi dati si è deciso di effettuare gli ulteriori test utilizzando solamente immagini di dimensioni 224×224px e 512×512px essendo queste ultime un buon compromesso tra tempi di generazione dei dataset, tempi di training della rete e qualità del risultato finale della ricostruzione.

6.3.2 Distanza della camera e presenza di materiali

Un altro risultato che è venuto fuori dai primi test effettuati è legato alla posizione della camera (tabella 6.5). In particolare, dai primi test si è visto che facendo variare la distanza della camera dall'oggetto il risultato della ricostruzione migliorava di circa il 50% per le sedie, il 7% per i tavoli e del 70% per gli armadi e le librerie (anche in questo caso però il risultato potrebbe essere falsato per i motivi elencati in precedenza).

	Distanza camera fissa		**Distanza camera variabile**	
	Chamfer Distance		**Chamfer Distance**	
	224×224px	**512×512px**	**224×224px**	**512×512px**
sedie	0,01096158 0,0110	0,01085494 0,0109	0,00473552 0,0047	0,00459662 0,0046
tavoli	0,01533876 0,0153	0,01512623 0,0151	0,01401991 0,0140	0,01378746 0,0138
librerie e armadi	0,01773648 0,0177	0,01763111 0,0176	0,00516797 0,0052	0,00396632 0,0040

Tabella 6.5: Valori della Chamfer Distance relativi agli esperimenti con dataset di immagini sintetiche dove la camera che inquadra il modello 3D è fissa ad una determinata distanza e dataset in cui la camera si allontana o si avvicina al modello

Questo miglioramento potrebbe essere legato al fatto di utilizzare delle immagini reali in fase di ricostruzione. Le immagini reali utilizzate infatti non mostrano quasi mai un oggetto alla stessa distanza o perfettamente al centro dell'immagine (figura 6.10). Di conseguenza utilizzando in fase di training delle immagini in cui la distanza della camera dall'oggetto varia potrebbe aiutare la rete neurale a performare meglio.

Un altro motivo potrebbe essere legato al fatto che facendo variare la distanza della camera dall'oggetto si tende a prevenire il fenomeno dell'overfitting permettendo alla rete

neurale una migliore generalizzazione.

In particolare, il fatto che i risultati delle sedie siano migliori rispetto a quelli dei tavoli potrebbe essere dovuto alla forma dell'oggetto in quanto un tavolo presenta comunque una forma molto più semplice rispetto a quella di una sedia e quindi, anche al variare della distanza dall'oggetto, la rete potrebbe comunque incorrere in problemi di overfitting.

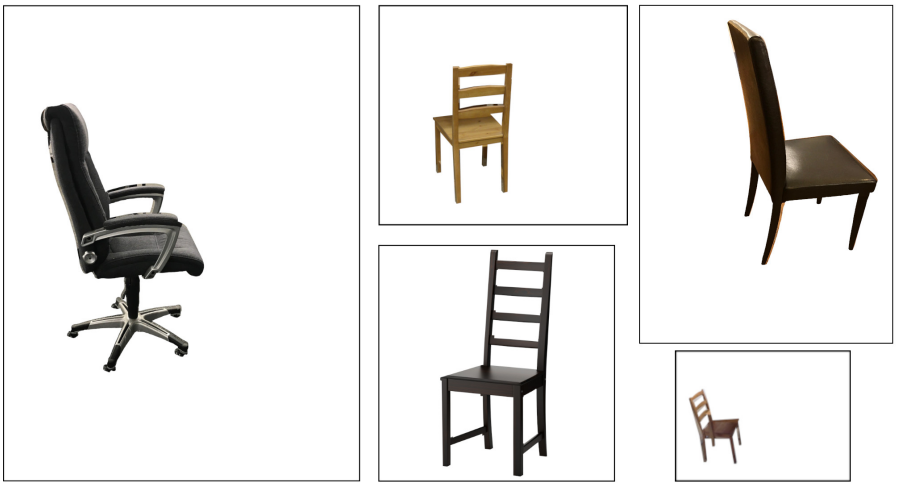

Figura 6.10: Esempio di immagini prese dal dataset di Pix3D ed utilizzate per la ricostruzione degli oggetti. Da notare che le varie immagini non hanno quasi mai dimensioni simili

Per quanto riguarda i materiali invece in questi primi test si è notato che la presenza di un qualsiasi materiale applicato all'oggetto migliora i risultati rispetto ad usare immagini di oggetti senza materiali (tabella 6.6).

	Modelli con materiali e texture		**Modelli senza materiali**	
	Chamfer Distance		**Chamfer Distance**	
	224×224px	**512×512px**	**224×224px**	**512×512px**
sedie	0,00473552 0,0047	0,00459662 0,0046	0,00546857 0,0055	0,00522626 0,0052
tavoli	0,01401991 0,0140	0,01378746 0,0138	0,01503572 0,0150	0,01465384 0,0147
librerie e armadi	0,00516797 0,0052	0,00396632 0,0040	0,00873554 0,0087	0,00864010 0,0086

Tabella 6.6: Valori della Chamfer Distance relativi agli esperimenti con dataset di immagini sintetiche ottenuti da modelli 3D a cui è stato applicato un materiale e dataset di immagini ottenute utilizzando modelli senza materiali applicati

6.3.3 Illuminazione

I test successivi hanno riguardato l'illuminazione della scena e dei vari modelli 3D utilizzati per la creazione delle immagini sintetiche. Come detto in precedenza per prima cosa sono state provate due configurazioni di luci e per ognuna della quali si è effettuato un'analisi dei risultati relativi alla ricostruzione di un oggetto 3D.

Da questi esperimenti si è notato che un'illuminazione abbastanza omogenea del modello 3D tende a far migliorare le prestazioni della rete neurale in fase di ricostruzione a differenza di una tipologia di illuminazione "più" realistica come potrebbe essere quella ottenuta mediante il posizionamento delle luci secondo il metodo "three-point lighting" (tabella 6.7). Questo potrebbe essere dovuto al fatto che un'illuminazione omogenea evita la creazione di zone "più scure" o ombre nette sugli oggetti, ombre che potrebbero essere interpretate dalla rete come possibili feature in fase di training e di conseguenza compromettere la buona riuscita di una ricostruzione.

	Illuminazione omogenea		**Three-Point Lighting**	
	Chamfer Distance		**Chamfer Distance**	
	224×224px	**512×512px**	**224×224px**	**512×512px**
sedie	0,00473552 0,0047	0,00459662 0,0046	0,00481142 0,0048	0,00486408 0,0049
tavoli	0,01401991 0,0140	0,01378746 0,0138	0,01453354 0,0145	0,01488299 0,0149
librerie e armadi	0,00516797 0,0052	0,00396632 0,0040	0,00620252 0,0062	0,00457752 0,0046

Tabella 6.7: Chamfer Distance relativa agli esperimenti fatti con una configurazione di luci posizionate in modo tale da illuminare in maniera omogenea il modello 3D e una configurazione di luci secondo il modello del three-point lighting

In questo caso si è notato comunque che, a differenza degli altri esperimenti, per quanto riguarda le sedie ed i tavoli utilizzando una diversa tipologia di illuminazione i risultati "peggiorano" aumentando la risoluzione. Questo potrebbe essere spiegato dal fatto che a basse risoluzioni l'impatto di particolari ombre o riflessioni, nel complesso, tende a notarsi di meno che a risoluzioni maggiori.

Altri esperimenti sono stati fatti facendo variare l'intensità delle luci per verificare se, come nel caso dello spostamento della camera, questo portasse a qualche tipo di miglioramento (tabella 6.8).
In questo caso però non si è osservato alcun tipo di miglioramento sostanziale anzi si è visto un leggero peggioramento nella maggior parte dei casi rispetto alle prove fatte con luci con intensità luminosa costante.

	Luci con intensità fissa		Luci con intensità variabile	
	Chamfer Distance		Chamfer Distance	
	224×224px	512×512px	224×224px	512×512px
sedie	0,00473552 0,0047	0,00459662 0,0046	0,00475790 0,0048	0,00483455 0,0048
tavoli	0,01401991 0,0140	0,01378746 0,0138	0,01412432 0,0141	0,01360294 0,0136
librerie e armadi	0,00516797 0,0052	0,00396632 0,0040	0,00563830 0,0056	0,00409192 0,0041

Tabella 6.8: Chamfer Distance relativa agli esperimenti fatti con una configurazione di luci posizionate in modo tale da illuminare in maniera omogenea il modello 3D con intensità costante e luci posizionate allo stesso modo ma con intensità variabile

6.3.4 Scala di Grigi

Per quanto riguarda la scala di grigi si è proceduto a generare dei dateset di immagini sintetiche che avessero un solo canale (L) in scala di grigio. Anche in questo caso i risultati ottenuti sono stati in media "peggiori" anche se non di molto rispetto ad utilizzare immagini a colori RGB (tabella 6.9).

	Immagini a colori (RGB)		Immagini in scala di grigi (L)	
	Chamfer Distance		Chamfer Distance	
	224×224px	512×512px	224×224px	512×512px
sedie	0,00473552 0,0047	0,00459662 0,0046	0,00478368 0,0048	0,00493321 0,0049
tavoli	0,01401991 0,0140	0,01378746 0,0138	0,01504626 0,0150	0,01437739 0,0144
librerie e armadi	0,00516797 0,0052	0,00396632 0,0040	0,00581470 0,0058	0,00376179 0,0038

Tabella 6.9: Risultati relativi l'utilizzo di un dataset con immagini a colori e lo stesso dataset ma con immagini in scala di grigi

Una cosa da sottolineare in questo caso è che i tempi di training per epoch utilizzando immagini in scala di grigi ad un solo canale sono in media di 1 o 2 secondi più veloci ma sono state necessarie in media circa un centinaio di epoch in più rispetto al training effettuato con immagini a colori (RGB). Il motivo del "risparmio" di tempo per la singola epoch è dovuto al fatto che le convoluzioni in questo caso non vengono effettuate su 3 canali come nel caso RGB ma solamente su 1 facendo così risparmiare qualche secondo. Anche in questo caso un aumento del numero di epoch necessarie per il training potrebbe essere un indicatore del fatto che questa tipologia di immagini non aiuti la rete in fase di training e che quindi si ha bisogno di un numero maggiore di epoch per ottenere un buon risultato.

Inoltre, va detto anche che in questo caso come input alla rete neurale per la ricostruzione sono state utilizzate immagini anch'esse in scala di grigio (L) proprio per il fatto che la rete era stata allenata con immagini ad un solo canale e quindi è stato necessario convertire le immagini reali da RGB a scala di grigio (L). Questo potenzialmente potrebbe anche avere influito sul risultato ottenuto.

6.3.5 Materiali

Altri esperimenti hanno riguardato nello specifico quali caratteristiche dei materiali e delle texture applicate agli oggetti potessero influire in qualche modo sul corretto funzionamento della rete neurale.
In particolare, sono stati creati dei dataset di immagini sintetiche dove di volta in volta veniva tolta o aggiunta una determinata texture dai materiali per vedere come la presenza o l'assenza di questa specifica mappa andasse ad influire sul training della rete neurale e poi sulla conseguente ricostruzione degli oggetti 3D.

Il primo dataset è stato realizzato utilizzando come texture solo il colore di base del materiale, color map, senza aggiungere altre mappe che andassero ad aggiungere effetti (tabella 6.10).

	Materiale realistico		**Solo Color Map**	
	CD		**CD**	
	224×224px	**512×512px**	**224×224px**	**512×512px**
sedie	0,00473552 0,0047	0,00459662 0,0046	0,00458514 0,0046	0,00451234 0,0045
tavoli	0,01401991 0,0140	0,01378746 0,0138	0,01380615 0,0138	0,01328260 0,0133
librerie e armadi	0,00516797 0,0052	0,00396632 0,0040	0,00474027 0,0047	0,00310177 0,0031

Tabella 6.10: Valori ottenuti utilizzando immagini prodotte da render di modelli con materiali "realistici" e immagini ottenute utilizzando come texture solo la color map

In questo caso si è visto un piccolo miglioramento dovuto probabilmente al fatto che non vi fossero presenti sui modelli le varie riflessioni della luce dovute a determinati valori del parametro "specular" dei vari materiali, in questo caso impostato a 0. Questo lo si nota soprattutto nei tavoli dove la presenza di una superficie piana parallela al "soffitto" e l'utilizzo di materiali come plastica o legno tendono a far comparire maggiormente i riflessi delle luci come mostrato in figura 6.11.

Successivamente si è proceduto aggiungendo man mano le altre "mappe" come la normal map e la roughnes map e si è visto che anche in questi casi i risultati erano abbastanza simili a quelli ottenuti utilizzando solo la color map (tabella 6.11).

Figura 6.11: Riflessioni dovute alla posizione delle luci che potrebbero influire sul processo di training

	Color & Normal Map		**Color & Normal & Rough Map**	
	CD		**CD**	
	224×224px	**512×512px**	**224×224px**	**512×512px**
sedie	0,00468042 0,0047	0,00466631 0,0047	0,00470219 0,0047	0,00472538 0,0047
tavoli	0,01348998 0,0135	0,01348868 0,0135	0,01367404 0,0137	0,01349073 0,0135
librerie e armadi	0,00508019 0,0051	0,00396016 0,0040	0,00463013 0,0046	0,00343041 0,0034

Tabella 6.11: Risultati ottenuti utilizzando dataset di immagini generati aggiungendo le altre mappe ai materiali

Una ulteriore analisi è stata fatta utilizzando materiali che presentassero delle texture con determinati pattern che potevano creare problemi in fase di training ed in questo caso si è notato che effettivamente i risultati peggiorano se si utilizzano materiali con pattern molto evidenti (tabella 6.12).

	Materiali con Pattern	
	CD	
	224×224px	**512×512px**
sedie	0,0047766 0,0048	0,00526175 0,0053
tavoli	0,01488817 0,0149	0,01443899 0,0144
librerie e armadi	0,00592099 0,0059	0,00420944 0,0042

Tabella 6.12: Risultati ottenuti utilizzando dataset di immagini ottenute da modelli che presentano dei materiali con specifici pattern

6.3.6 Altri esperimenti

Per quanto riguarda poi le sedie si è proceduto a fare ulteriori test.
In particolare, si è proceduto aumentato il numero di immagini per modello utilizzate per il training passando da 25 per modello a 50 e poi a 100 (tabella 6.13). In questo caso si è notato che i risultati tendevano a migliorare al crescere del numero di immagini utilizzate.

Inoltre, si è visto che se pur il tempo di training per epoch aumentasse in maniera proporzionale al numero delle immagini utilizzate per il training, il numero di epoch necessarie per concludere la fase di training diminuiva al crescere delle immagini. L'unica cosa negativa è che il tempo di render delle immagini cresce anche esso in maniera proporzionale al numero di immagini da generare. Di conseguenza, per generare 50 immagini per modello ci vuole all'incirca il doppio del tempo rispetto a generarne 25.

25 Render per modello		**50 Render per modello**		**100 render per modello**	
CD		**CD**		**CD**	
224×224px	**512×512px**	**224×224px**	**512×512px**	**224×224px**	**512×512px**
0,00473552	0,00459662	0,00435512	0,00420133	0,00410967	0,00387922
0,0047	0,0046	0,0044	0,0042	0,0041	0,0039

Tabella 6.13: Risultati ottenuti aumentando il numero di immagini sintetiche utilizzate per il training della rete neurale

Capitolo 7

Conclusioni e sviluppi futuri

7.1 Conclusioni

Lo scopo di questo elaborato è stato quello di indagare se ci fossero dei parametri di un'immagine 2D, usata per il training di una rete neurale dedicata alla ricostruzione di oggetti 3D, che andassero ad impattare maggiormente sulla qualità del risultato finale.
Dalle prove effettuate in questa tesi, se pur limitate a specifiche caratteristiche di una immagine sintetica e a sole tre categorie di oggetti, si evince che effettivamente alcuni parametri delle immagini sintetiche utilizzate per il training di una rete neurale possono avere un impatto anche solo in termini di tempi di calcolo sia sul processo di training che sulla buona riuscita della ricostruzione di un oggetto 3D.

In generale, dai vari esperimenti effettuati si è osservato che i parametri di un'immagine 2D che maggiormente influiscono sulla buona riuscita di una ricostruzione sono in particolare: la risoluzione delle immagini utilizzate per il training, la presenza di materiali e texture applicate ai modelli 3D, l'illuminazione dei modelli 3D usati per la generazione delle immagini e il far variare la distanza della camera dal modello 3D quando si effettuano i render. Questi sono stati i tre paramenti che hanno fatto notare un effettivo miglioramento o peggioramento dei valori medi di Chamfer Distance calcolati tra i modelli ground truth e quelli ricostruiti.

Analizzando anche i tempi, per capire quali parametri ottimizzassero al meglio tutto il processo, dalla generazione delle immagini dei dataset fino alla ricostruzione di un oggetto 3D, si è visto che al crescere della risoluzione delle immagini aumentano sia i tempi di training che il tempo necessario alla generazione dei dataset. Dai dati ottenuti in questi esperimenti si evince che un buon compromesso sarebbe quello di utilizzare risoluzioni di 224×224px o 512×512px le quali forniscono dei risultati abbastanza soddisfacenti in tempi relativamente brevi rispetto, per esempio, ad immagini con dimensioni maggiori, le quali invece non sempre forniscono prestazioni migliori. Nel caso specifico della dimensione delle immagini utilizzate, comunque, un'ulteriore analisi potrebbe essere effettuata andando a modificare il codice della rete neurale, per esempio, aumentando la "profondità" della rete, magari inserendo o eliminando qualche livello convoluzionale in più in relazione alle

dimensioni delle immagini utilizzate per il training. In questo modo si eviterebbe il problema di arrivare al livello di pooling finale con una matrice delle feature di dimensioni quasi cinque volte più grandi, come nel caso di immagini di dimensioni 1024×1024px (32×32), rispetto a quello che è lo standard per una rete profonda 18 livelli come ResNet-18 (7×7) [35].

Avendo infine ottenuto anche in dettaglio i tempi di training e di rendering è possibile, inoltre, per chi ne avesse la necessità, disporre dei dati necessari per ottimizzare il processo al fine di ottenere i risultati voluti con il minor spreco di tempo e di risorse.

Per quanto riguarda gli altri esperimenti effettuati riguardati i diversi parametri individuati, ed in particolare gli esperimenti relativi le varie "mappe" applicate ai materiali, pur osservando effettivamente una variazione della Chamfer Distance, queste variazioni in alcuni casi non sono tali da permettere di affermare con assoluta certezza che vi siano dei miglioramenti o dei peggioramenti abbastanza rilevanti. Infatti, alcune variazioni nei valori medi di Chamfer Distance ottenuti potrebbero dipendere dal "percorso" che il flusso di informazioni fa attraverso i neuroni della rete in fase di training.
Questo perchè, per come sono sviluppati gli algoritmi di machine learning, il processo di training di un modello di rete neurale risulta essere intrinsecamente stocastico, nel quale diversi fattori possono incidere sulle prestazioni finali della rete.
In particolare, gli algoritmi utilizzati nell'ambito del machine learning sono algoritmi "non deterministici", cioè algoritmi che utilizzano elementi di casualità quando vengono prese "decisioni" durante l'esecuzione di un processo.
La scelta di utilizzare questa tipologia di algoritmi al posto dei classici algoritmi "deterministici" è dettata dal fatto che gli algoritmi non deterministici si adattano meglio a lavorare con grandi quantità di dati riuscendo ad ottenere dei buoni risultati in tempi ragionevoli. Tali algoritmi, infatti, sfruttano la "casualità" sia in fase di inizializzazione che durate tutto lo svolgimento del processo riuscendo, in questo modo, ad ottimizzare i tempi di calcolo. Questo però implica che verrà sempre eseguito un diverso ordine di passaggi anche quando lo stesso algoritmo viene lanciato nuovamente sugli stessi dati, portando inevitabilmente ad avere risultati differenti.
Per quanto riguarda una rete neurale i fattori che introducono della "casualità" nel processo di training sono in generale:

- L'inizializzazione dei pesi. I pesi in una rete neurale sono sempre inizializzati in maniera casuale, in generale con valori molto piccoli prossimi allo zero.

- L'ordine di processamento dei dati. I dati passati alla rete neurale in fase di training sono passati in maniera randomica e non in una determinata sequenza. Questo implica che in base a quale dato la rete analizzerà per primo questo potrebbe poi andare ad influenzare poi tutto il processo di training.

- Il processo di ottimizzazione dei pesi. Durante la fase di training vengono individuati i pesi che ottimizzano al meglio la rete neurale nello svolgimento del compito per cui è stata progettata. Tale operazione è svolta utilizzando appositi algoritmi di ottimizzazione non deterministici come ad esempio ADAM [4] introducendo anche in questo caso altri elementi di casualità.

Altri elementi che potrebbe incidere inoltre sul processo di training di una rete neurale sono anche: la CPU o GPU utilizzata per i calcoli, la differenza tra le versioni delle varie librerie utilizzate o anche l'uso di alcune librerie al posto di altre.

Per limitare questo problema esistono alcuni accorgimenti che si potrebbero prendere per cercare di ridurre al massimo l'utilizzo di fonti di casualità come, ad esempio, provare a sostituire alcuni algoritmi non deterministici con altri di tipo deterministico. Tutto questo però non garantisce in assoluto che i risultati siano sempre gli stessi ed anzi potrebbe potenzialmente portare ad una riduzione delle prestazioni generali della rete [28].
Nella ricerca svolta in questa tesi si anche provveduto ad eseguire più volte degli esprimenti utilizzando lo stesso dataset di immagini sintetiche per osservare quanto fosse effettivamente la variazione dei risultati dovuta ai vari fattori elencati in precedenza. Dai risultati ottenuti si evince che effettivamente ci sono delle variazioni nei risultati ottenuti ed in particolare la Chamfer Distance ottenuta, eseguendo lo stesso esperimento più volte, varia in un range che va da $\pm 2\%$ a $\pm 8\%$.

Infine, per quanto riguarda la creazione di dataset di immagini sintetiche, la pipeline proposta in questa tesi resta comunque valida non solo per la creazione di dataset sintetici da utilizzare per il training di una rete neurale ma anche per la creazione di qualsiasi tipologia di dataset contenente immagini 2D sintetiche da utilizzare in qualsiasi ambito.

7.2 Possibili miglioramenti e sviluppi futuri

Per quanto riguarda lo studio specifico svolto in questa tesi, gli esperimenti effettuati sono da considerarsi come un punto di partenza per ulteriori revisioni e ricerche in questo campo. Ulteriori analisi dovrebbero essere effettuate per validare in maniera rigorosa i risultati ottenuti in questa tesi, per esempio, mediante l'utilizzo di dataset di immagini sintetiche di oggetti appartenenti a categorie diverse da quelle viste in questa tesi.

Di seguito sono elencati dei possibili miglioramenti ed integrazioni che potranno essere di aiuto per migliorare o validare ulteriormente questo studio:

- Aggiunta di più modelli ai dataset utilizzati. In questo studio, come visto in precedenza, sono stati utilizzati 120 modelli (40 per categoria di oggetto scelto), quindi un'ulteriore analisi potrebbe essere fatta aumentando sensibilmente il numero di modelli. Questo perché avendo a disposizione più modelli da cui generare immagini da usare per il training, si potrebbe allenare una rete in modo tale che sia in grado di generalizzare meglio rispetto a quanto visto in queste analisi.

- Analisi riguardo le immagini reali utilizzati per la ricostruzione. In questa tesi le immagini che sono state utilizzate per la ricostruzione degli oggetti sono state prese in maniera casuale da quelle offerte dal dataset di Pix3D. Un'ulteriore analisi potrebbe riguardare come i parametri di queste immagini reali possono influire sulla ricostruzione di un oggetto 3D. Quindi si potrebbe porre l'attenzione, in primo luogo, sulle

dimensioni di tali immagini ed in particolare quanto incide il fatto di dover scalare un'immagine per portarla alle dimensioni richieste dalla rete sulla qualità della ricostruzione di un oggetto.

- Analisi più dettagliate su materiali e texture. In generale si potrebbe indagare ancora più in profondità riguardo l'influenza che i vari materiali e le texture hanno sul buon funzionamento di una rete neurale magari valutando anche l'impatto che i vari colori potrebbero avere sul risultato della ricostruzione.

- Provare a variare i diversi "iperparametri[1]" specifici della rete neurale. In tutte le prove effettuate in questa tesi sono stati mantenuti sempre gli stessi parametri "interni" della rete neurale come, per esempio, il batch size o il learning rate. Una ulteriore analisi potrebbe essere effettuata facendo variare questi iperparametri anche, per esempio, in relazione alle dimensioni delle immagini utilizzate per il training scegliendo magari di abbassare il learning rate per risoluzioni più piccole o aumentarlo per risoluzioni maggiori. Stessa cosa potrebbe essere fatta per le dimensioni del batch anche se queste dipendono in particolare dal tipo di GPU utilizzata e dalla memoria disponibile (nel caso specifico di questa tesi il valore del batch size per motivi di hardware è stato sempre 16)

- Aumentare la "profondità" della rete. In questo studio si è lavorato utilizzando una rete profonda 18 livelli. Si potrebbe pensare di aggiungere dei livelli in più e valutare se i risultati ottenuti restano comunque validi. Oppure far variare la profondità in relazione alla dimensione delle immagini utilizzate per il training.

- Testare i dataset utilizzati in questa tesi su reti neurali diverse da BSP-net per verificare che i risultati ottenuti siano gli stessi anche utilizzando reti differenti. Gli esperimenti fatti in questa tesi infatti sono stati effettuati sul modello di rete neurale proposto dagli autori di BSP-net e quindi i risultati ottenuti potrebbero essere validi solo per questo determinato modello di rete neurale. In linea teorica qualsiasi rete il cui image encoder sia basato su ResNet-18 dovrebbe dare risultati simili ma non è da escludere che su reti diverse da BSP-net i risultati siano completamente opposti.

In conclusione, il crescente utilizzo di reti neurali sia per la ricostruzione di oggetti 3D che, in generale, in qualsiasi ambito legato alla Computer Vision può portare nei prossimi anni allo sviluppo di diversi algoritmi e modelli ognuno dei quali potrebbe dare più peso a determinate caratteristiche di un'immagine rispetto ad altre. È quindi necessario riuscire a comprendere in maniera più generale possibile quali parametri di una immagine sintetica utilizzata per il training di una rete neurale siano in grado di far lavorare al meglio qualsiasi modello proposto.

[1]Per *iperparametri* si intendono tutti quei parametri propri di una rete neurale che potrebbero incidere sul processo di training della rete stessa. Rientrano tra gli iperparamentri di una rete: il numero di livelli, la dimensione del batch, il learning rate, la tipologia di ininzializzazione dei pesi, la scelta di un determinato ottimizzatore ed altri ancora.

Un auspicio, infine, sarebbe quello che attraverso lo studio dei parametri di un'immagine 2D, utilizzata per il training di una rete neurale, che maggiormente incidono sulle prestazioni di una rete neurale si arrivi alla creazione di un modello che sia in grado di ottenere sempre ottimi risultati indipendentemente dalla tipologia di immagine utilizzata per il training.

Appendice A

Requisiti della workstation utilizzata e versione dei software

A.1 Workstation

Il lavoro svolto in questa tesi è stato effettuato utilizzando una workstation con le seguenti caratteristiche:

- Sistema Operativo: Windows 10 Pro - 64 bit
- Processore (CPU): Intel Core i7-8700 3.20GHz
- Memoria RAM: 16GB
- Scheda Grafica (GPU): NVIDIA GeForce GTX 1060 con 6GB di memoria dedicata

A.2 Software

Lato software sono stati utilizzati i seguenti programmi con le relative versioni:

- Blender - versione 2.83.13 LTS
- BlenderProc - versione 2.3.0
- Binvox - versione 1.35
- VSCode - versione 1.73.1

- Python - versione 3.6.8 con le seguenti librerie:
  - PyTorch - versione 1.6.0 + CUDA versione 10.1
  - Torchvision - versione 0.7.0
  - NumPy - versione 1.19.5
  - Open3D - versione 0.13.0
  - Scikit-learn - versione 0.24.0
  - Pillow - versione 8.1.2

Bibliografia

[1] *3D Studio Max*. URL: https://www.autodesk.it/products/3ds-max.

[2] *3D Warehouse*. URL: https://3dwarehouse.sketchup.com.

[3] *AmbientCG*. URL: https://ambientcg.com.

[4] Diederik P. Kingma Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: (2014). DOI: 10.48550/arXiv.1412.6980.

[5] *Binvox*. URL: https://www.patrickmin.com/binvox.

[6] *Blender*. URL: https://www.blender.org.

[7] *Blenderkit*. URL: https://www.blenderkit.com.

[8] *Cinema 4D*. URL: https://www.maxon.net/it/cinema-4d.

[9] Olga Russakovsky Jia Deng Hao Su Jonathan Krause Sanjeev Satheesh Sean Ma Zhiheng Huang Andrej Karpathy Aditya Khosla Michael Bernstein Alexander C. Berg Li Fei-Fei. «ImageNet Large Scale Visual Recognition Challenge». In: (2014). DOI: 10.48550/arXiv.1409.0575. URL: https://www.image-net.org.

[10] Xingyuan Sun Jiajun Wu Xiuming Zhang Zhoutong Zhang Chengkai Zhang Tianfan Xue Joshua B. Tenenbaum William T. Freeman. «Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling». In: (2018). DOI: 10.48550/arXiv.1804.04610. URL: http://pix3d.csail.mit.edu.

[11] Artem Rozantsev Vincent Lepetit Pascal Fua. «On rendering synthetic images for training an object detector». In: (2015). DOI: 10.1016/j.cviu.2014.12.006.

[12] *Haven*. URL: https://polyhaven.com/textures.

[13] Param S. Rajpura Hristo Bojinov Ravi S. Hegde. «Object Detection Using Deep CNNs Trained on Synthetic Images». In: (2017). DOI: 10.48550/arXiv.1706.06782.

[14] Junyi Pan Xiaoguang Han Weikai Chen Jiapeng Tang Kui Jia. «Deep Mesh Reconstruction from Single RGB Images via Topology Modification Networks». In: (2019). DOI: 10.48550/arXiv.1909.00321. URL: https://github.com/jnypan/TMNet.

[15] Nanyang Wang Yinda Zhang Zhuwen Li Yanwei Fu Wei Liu Yu-Gang Jiang. «Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images». In: (2018). DOI: 10.48550/arXiv.1804.01654. URL: https://github.com/nywang16/Pixel2Mesh.

[16] Maximilian Denninger Martin Sundermeyer Dominik Winkelbauer Youssef Zidan Dmitry Olefir Mohamad Elbadrawy Ahsan Lodhi Harinandan Katam. «BlenderProc». In: (2019). DOI: 10.48550/arXiv.1911.01911. URL: https://dlr-rm.github.io/BlenderProc.

[17] *Keras*. URL: https://keras.io.

[18] Mingxing Tan Quoc V. Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». In: (2019). DOI: 10.48550/arXiv.1905.11946.

[19] Xiangyun Hua Zuxun Zhanga Yansong Duana Yongjun Zhanga Junfeng Zhua Huaping Long. «Lidar Photogrammetry and its data organization». In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2011). DOI: 10.5194/isprsarchives-XXXVIII-5-W12-181-2011.

[20] *Maya*. URL: https://www.autodesk.it/products/maya.

[21] Mehendale Ninad, Mehendale Ninad e Neoge Srushti. «Review on Lidar Technology». In: (2020). DOI: 10.2139/ssrn.3604309.

[22] *NumPy*. URL: https://numpy.org.

[23] *Open3D*. URL: http://www.open3d.org.

[24] *Photogrammetry Pipeline*. URL: https://alicevision.org/#photogrammetry.

[25] *Poliigon*. URL: https://www.poliigon.com.

[26] *Python*. URL: https://www.python.org.

[27] *PyTorch*. URL: https://pytorch.org.

[28] *PyTorch - Reproducibility*. URL: https://pytorch.org/docs/stable/notes/randomness.html.

[29] *Rhinoceros 3D*. URL: https://www.rhino3d.com.

[30] Vajira Thambawita Inga Strumke Steven A. Hicks Pal Halvorsen Sravanthi Parasa Michael A. Riegler. «Impact of Image Resolution on Deep Learning Performance in Endoscopy Image Classification: An Experimental Study Using a Large Dataset of Endoscopic Images». In: (2021). DOI: 10.3390/diagnostics11122183.

[31] *Scikit-learn*. URL: https://scikit-learn.org.

[32] Mats L. Richter Wolf Byttner Ulf Krumnack Ludwdig Schallner Justin Shenk. «Size Matters». In: (2021). DOI: 10.1007/978-3-030-86340-1_11.

[33] *Sketchfab*. URL: https://sketchfab.com.

[34] Carl F. Sabottke Bradley M. Spieler. «The Effect of Image Resolution on Deep Learning in Radiography». In: (2020). DOI: 10.1148/ryai.2019190015.

[35] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. «Deep Residual Learning for Image Recognition». In: (2015). DOI: 10.48550/arXiv.1512.03385.

[36] *TensorFlow*. URL: https://www.tensorflow.org.

[37] *Turbosquid*. URL: https://www.turbosquid.com.

[38] Angel X. Chang Thomas Funkhouser Leonidas Guibas Pat Hanrahan Qixing Huang Zimo Li Silvio Savarese Manolis Savva Shuran Song Hao Su Jianxiong Xiao Li Yi Fisher Yu. «ShapeNet: An Information-Rich 3D Model Repository». In: (2015). DOI: 10.48550/arXiv.1512.03012. URL: https://shapenet.org.

[39] Zhiqin Chen Andrea Tagliasacchi Hao Zhang. «BSP-Net: Generating Compact Meshes via Binary Space Partitioning». In: (2019). DOI: 10.48550/arXiv.1911.06971. URL: https://bsp-net.github.io.