UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria informatica e dell'automazione

Approcci generativi di Deep Learning applicati alle nuvole di punti, come supporto nell'ambito dei beni culturali.

Candidato:
Alessandro ZAMPONI

Relatore:
Prof. Emanuele FRONTONI

Correlatore:
Dott. Massimo MARTINI

Anno Accademico 2020-2021

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria informatica e dell'automazione

Generative Deep Learning approaches applied to point clouds, as a support in the field of cultural heritage.

Candidate:
Alessandro ZAMPONI

Advisor:
Prof. Emanuele FRONTONI

Coadvisor:
Dott. Massimo MARTINI

Academic Year 2020-2021

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria informatica e dell'automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

L'ambito dei beni culturali rappresenta un campo in continua espansione, soprattutto negli ultimi anni, dove, grazie alle tecnologie di rilevazione, è stato possibile ottenere rappresentazioni 3D di opere d'arte, edifici e monumenti. Questo, oltre che concedere un nuovo tipo di visita dei musei, apre la strada alla ristrutturazione di edifici o al completamento opere d'arte prive di alcuni dettagli. Tutto questo è reso possibile dalle tecniche e algoritmi di *Deep Learning*: tuttavia non si dispone ancora di un quantitativo di dati sufficiente per ottenere dei risultati considerevoli. Attualmente l'unico dataset di cui si può usufruire liberamente in quest'ambito è *ArCH (Architectural Cultural Heritage)*, dove le varie scene sono rappresentate attraverso le nuvole di punti, un particolare formato con cui è possibile visualizzare gli oggetti in 3D e navigare all'interno di essi. Per cercare di colmare la mancanza di dati sono stati applicati approcci generativi, allenando perciò delle reti neurali e rendendole capaci di generare dei nuovi oggetti, da aggiungere in un secondo momento al dataset di partenza, calcolando alcune metriche che possano aiutare nella valutazione quantitativa dei risultati ottenuti.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto

L'ambito dei beni culturali è sicuramente uno tra quelli dove, negli ultimi anni, si stanno verificando notevoli sviluppi: l'obiettivo principale è quello di reinventare le modalità di visita di musei o luoghi d'interesse, grazie a delle tecnologie che permettano la ricerca e il riconoscimento delle immagini. La mancanza di un buon numero di dati non consente però lo sviluppo di soluzioni capaci di manipolare gli oggetti e visualizzarli tridimensionalmente. Grazie all'utilizzo di tecniche e modelli di *deep learning [1]* è stato possibile sfruttare varie classi di algoritmi capaci di offrire delle soluzioni valide e utili allo sviluppo di quest'ambito: come esempio viene riportata la segmentazione semantica con la quale vengono riconosciuti i singoli elementi all'interno delle immagini. Attraverso l'utilizzo di approcci generativi si possono compensare parzialmente le mancanze sopracitate: creando nuovi oggetti, rappresentati attraverso le nuvole di punti, è possibile completare alcune scene carenti di dettagli o rendere disponibili delle figure di riferimento per la ristrutturazioni di opere d'arte o monumenti. Si sfruttano perciò quelle che vengono definite *GAN (Generative Adversarial Network)*, ovvero coppie di reti neurali in cui una delle due svolge il ruolo di **generatore(G)** e l'altra di **discriminatore(D)**. Queste due reti vengono allenate all'interno di un framework così che la rete G possa apprendere come generare dei nuovi oggetti, assicurandosi che questi abbiano la stessa distribuzione dei dati usati nella fase di addestramento. La rete D apprende invece come distinguere i dati reali da quelli generati dalla rete G. Attraverso le GAN può essere effettuata la simulazione di task realistici o generare dati quando quelli già presenti, come in questo caso, non sono sufficienti per il training. Grazie agli sviluppi tecnologici dei Mobile Mapping System (MMS) e all'utilizzo dei Sistemi Aeromobili a Pilotaggio Remoto (SAPR), usati nell'ambito urbano o architettonico per rilevare le diverse strutture è stato possibile generare delle grandi moli di dati 3D, che nello specifico vengono rappresentate dalle *nuvole di punti*. Tutto ciò ha portato alla creazione di un dataset specifico di questo ambito, *ArCH – Architectural Cultural Heritage [2]*, su cui è stata effettuata la classificazione e la data augmentation attraverso diverse reti neurali e relative tecniche di deep learning. La carenza dei dati di scene 3D in quest'ambito impedisce la creazione di reti in grado di classificare autonomamente

i vari oggetti presenti nelle scene. Inoltre non è possibile sfruttare dei metodi preesistenti o utilizzati in altri ambiti, poichè le rappresentazioni di queste scene includono oggetti non comuni, con strutture piuttosto complesse. Queste sono le motivazioni per cui le varie scene del dataset sono state suddivise in singoli oggetti e raggruppati poi nelle relative classi di appartenenza. Attraverso poi l'utilizzo di due reti neurali, la *Pointnet[3]* e la *setVAE[4]*, è stato possibile portare a termine rispettivamente i task di classificazione e di data augmentation sul dataset *ArCH*. Di seguito vengono proposte le soluzioni adottate sia per quanto riguarda l'etichettatura del dataset, sia per la parte relativa all'utilizzo del dataset con le varie reti.

1.2 Obiettivi e contributi principali

L'obiettivo principale della tesi è quello di proporre degli approcci generativi, sfruttando reti neurali piuttosto recenti, per la creazione di nuovi oggetti nell'ambito dei beni culturali così da rendere disponibile un framework in grado di colmare la mancanza di dati in questo specifico ambito. Alla base degli ottimi risultati che si ottengono attraverso le tecniche di deep learning c'è proprio la grande disponibilità di dati per allenare le reti. Il primo contributo rilevante, di questo lavoro di tesi, è stata la ristrutturazione del dataset ArCH, etichettando i singoli oggetti delle varie scene, in modo tale da renderlo utilizzabile all'interno delle reti. Successivamente sono stati creati dei nuovi oggetti, grazie all'utilizzo delle reti neurali offerte dalla comunità scientifica: una volta generati gli oggetti di interesse, sono stati studiati qualitativamente e quantitativamente, confrontandoli con quelli originali per poi, come ultimo step, effettuare una classificazione. I dataset considerati e con cui è stato svolto il lavoro sono *ArCH* e *Shapenet[5]*: il primo rappresenta quello di reale interesse, mentre il secondo è stato usato per verificare il corretto funzionamento delle reti. Infine sono stati effettuati dei confronti sulla classificazione di entrambi i dataset, prima e dopo aver effettuato la data augmentation.

1.3 Struttura della tesi

Dopo una prima presentazione dello stato dell'arte degli studi effettuati in quest'ambito, si illustreranno quali sono le prospettive del lavoro che si ha intenzione di sviluppare. Si parlerà del lavoro svolto, illustrando anche le modalità con cui sono stati completati i vari task e presentando brevemente i vari risultati ottenuti, descritti nel dettaglio in un secondo momento. Infine verrà riservata una sezione per indicare quali lavori futuri potrebbero essere svolti a partire da questa tesi, focalizzando l'attenzione sul come migliorare i risultati e proponendo approcci diversi.

Capitolo 2

Stato dell'arte

Dopo aver presentato e illustrato il modello di rappresentazione degli oggetti in quest'ambito, cioè le *nuvole di punti*, vengono riassunte quelle che sono le conoscenze e le tecniche allo stato dell'arte del *deep learning*. Viene poi descritto il task di classificazione e la rete utilizzata per completarlo, per poi terminare con la parte di generazione. Dopo averne spiegato lo scopo e l'utilità, vengono illustrate le reti utilizzate per generare dei nuovi oggetti con cui completare la data augmentation.

2.1 Point Clouds

Le nuvole di punti(point clouds)[6] sono un insieme di punti definiti su 3 assi(X,Y,Z) con eventuali valori di intensità come RGB o altre proprietà scalari: vengono utilizzate per la rappresentazione tridimensionale di strutture e per rilevazioni territoriali.

L'interesse è ricaduto su questa particolare rappresentazione, in quanto il dataset *ArCH* è stato creato a partire da numerose scansioni, salvate poi come nuvole di punti, come si può vedere in figura 2.1, così da avere la possibilità di lavorare con oggetti 3D. La generazione delle nuvole di punti avviene perciò in seguito al completamento di scansioni delle scene su cui si desidera lavorare. A scansioni più durature corrispondono una qualità e densità che crescono all'aumentare del tempo. L'utilizzo di questo tipo di rappresentazione ha garantito una diminuzione dei tempi di acquisizione dei dati, mantenendo comunque una buona qualità. Un limite però riscontrato è stata l'incapacità di definire automaticamente le varie caratteristiche delle componenti, rappresentando perciò solo la parte più esterna degli oggetti rilevati. Le point clouds permettono inoltre una navigazione all'interno degli oggetti scansionati, selezionando le parti di maggiore interesse da analizzare nel dettaglio.

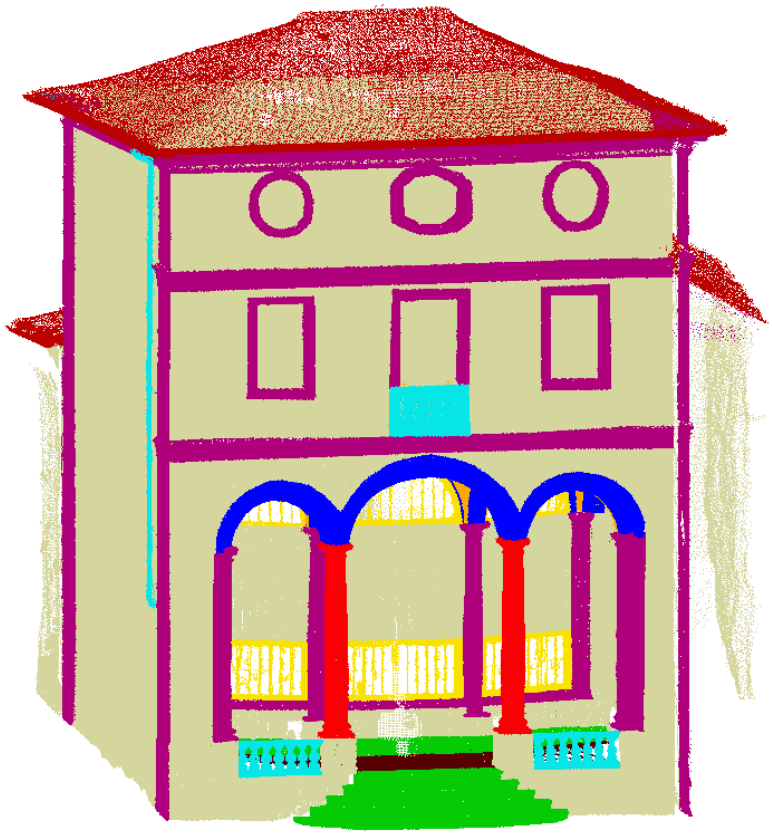

Figura 2.1: Esempio di una nuvola di punti del dataset ArCH

2.2 Deep Learning

Il deep learning[7][8], sotto categoria del machine learning[9][10], ha come scopo quello di creare modelli di apprendimento su più livelli, andando ad astrarre sempre di più le informazioni e sfruttando i valori prodotti in output nei livelli precedenti come valori di input per quelli successivi come si può vedere nella figura 2.2. La scalabilità del deep learning, resa possibile mediante l'aumento dei dati disponibili, è ciò che la differenzia dal machine learning: questo aspetto permette quindi ai sistemi di deep learning di migliorare le prestazioni all'aumentare dei dati, contrariamente a quanto succede nel machine learning, dove una volta raggiunti determinati livelli di performance non sono più scalabili.

Lo sviluppo di componenti software sempre più efficienti ed efficaci ha permesso un contemporaneo sviluppo di quest'ambito. Grazie all'introduzione e il conseguente utilizzo delle *GPUs* è stato possibile lavorare su grandi quantità di dati, merito anche delle crescenti disponibilità di dataset, in tempi ragionevoli. La rete può essere allenata in due diversi modi:

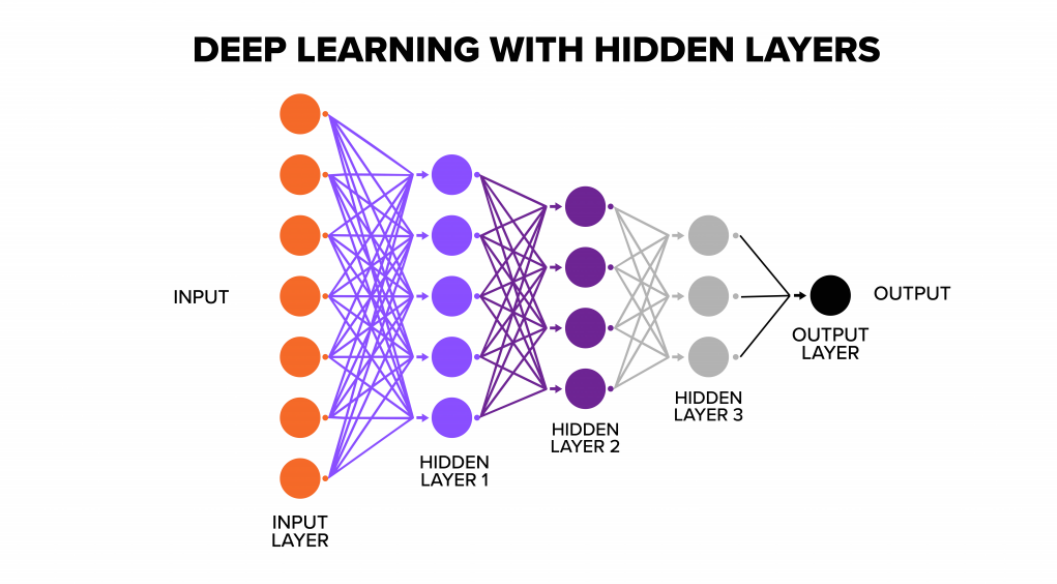

Figura 2.2: Esempio di una rete di Deep Learning

1. Apprendimento supervisionato – al sistema vengono forniti dei dati in input insieme alle caratteristiche che dovranno avere i risultati prodotti. In questo modo il sistema dovrà apprendere il legame presente così da renderlo riutilizzabile.

2. Apprendimento non supervisionato – all'algoritmo vengono forniti esclusivamente un set di dati in input così che il sistema identifichi nelle informazioni in ingresso autonomamente delle caratteristiche.

Con l'apprendimento automatico i dispositivi possono modificare i propri algoritmi così da apprendere nuove caratteristiche da sfruttare. In questo senso, terminata la fase di training, il modello allenato è capace di classificare i dati appartenenti al dataset, secondo le loro classi di appartenenza, realizzando così la *classificazione* degli oggetti. Il processo di apprendimento ha quindi principalmente due passaggi:

- *Training*: una parte del dataset che verrà usato deve essere dedicato esclusivamente all'allenamento della rete.

- *Testing*: un'ulteriore porzione del dataset non utilizzato deve essere dedicato a questo passaggio, per testare la rete e il suo corretto funzionamento.

Per avere un modello attendibile l'errore che viene calcolato durante queste due fasi deve essere basso. Per permettere l'ottimizzazione della fase di training una porzione del dataset viene riservata per il *Validation Set*: in questo modo è possibile controllare che la rete non impari a memoria i dati, generalizzando bene gli oggetti e ottenendo quindi delle buone performance. Risulta quindi necessario effettuare uno split iniziale del dataset così da attribuire un quantitativo di dati consono per questi 3 set:

- Training Set: 70%
- Testing Set: 20%
- Validation Set: 10%

Questo rapporto è quello più indicato, essendo il più utilizzato allo stato dell'arte: può essere variato in base alle esigenze, mantenendo però una proporzione simile a quella sopra indicata.

Effettuata la classificazione del dataset, si passa ad un altro task che si può considerare a tutti gli effetti un'evoluzione della classificazione: la *segmentazione semantica*, figura 2.3. Attraverso questo task è possibile associare ogni punto ad una determinata classe. Nel passaggio tra la classificazione e la segmentazione, è

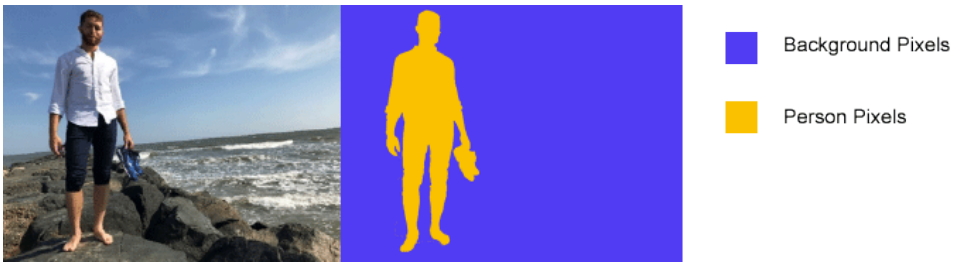

Figura 2.3: Esempio di segmentazione semantica

possibile trovare anche il task di *object detection*. Solitamente si assume che ci sia un elemento principale all'interno dell'immagine e che l'attenzione debba essere posta esclusivamente su di esso, riconoscendo la sua classe di appartenenza. Non è raro però avere più oggetti di interesse all'interno delle immagini: per questo motivo si deve conoscere, oltre alla classe, anche l'esatta posizione dell'oggetto nell'immagine. Questo task viene utilizzato in numerosi contesti, come la guida automatica, dove la rete deve poter riconoscere la posizione dei veicoli, dei pedoni, la strada ed altri ostacoli. In questo studio non sono stati effettuati task di *segmentazione semantica* e *object detection*, che possono quindi rappresentare dei lavori futuri inerenti a questo ambito.

2.3 Classificazione di nuvole di punti

Avendo a disposizione la rappresentazione degli oggetti presenti all'interno del dataset, si procede con la classificazione: pertanto viene attribuita una classe di appartenenza, tra quelle predefinite, ad ogni singolo oggetto. In questo senso è importante che la rete impari a riconoscere le caratteristiche che accomunano gli oggetti delle stesse classi così da poter fornire in output una probabilità di appartenenza ad una di esse. Le nuvole di punti, presentando dei formati irregolari, molto spesso vengono manipolate dai ricercatori. Grazie alla Pointnet[3], una delle reti più utilizzate in questo contesto, è possibile usufruire di queste nuvole direttamente,

rispettando l'invarianza di permutazione dei punti. La *PointNet* permette quindi di elaborare in input direttamente la nuvola di punti grezza, senza effettuare operazioni di preprocessing sui dati, secondo la struttura rappresentata in figura 2.4. Inoltre si può sfruttare queste rete per diversi task: la classificazione, la segmentazione delle parti di un oggetto e la segmentazione semantica di una scena. La struttura della rete è piuttosto semplice poichè inizialmente ogni punto viene processato singolarmente ed indipendentemente dagli altri, per poi essere rappresentato attraverso le sue 3 coordinate X, Y, Z. Con l'addestramento questa rete riesce poi a selezionare i punti più significativi o quelli con più contenuto informativo, codificando i motivi della scelta.

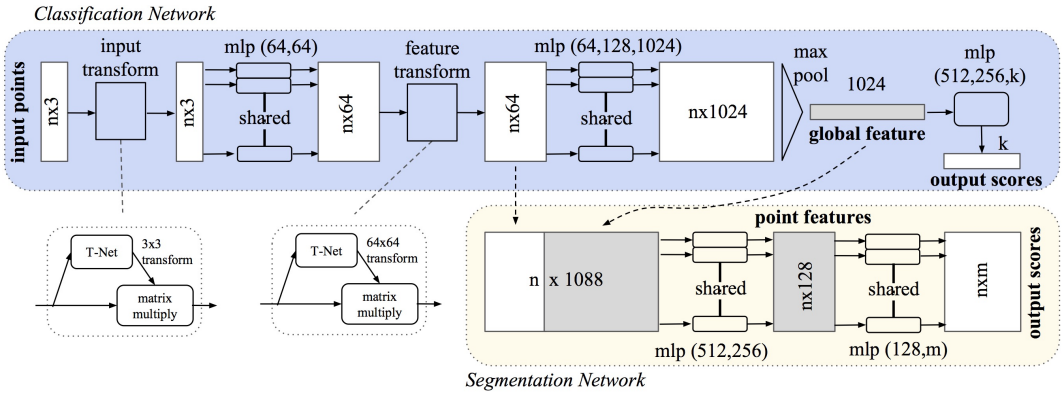

Figura 2.4: Architettura della PointNet

La *PointNet* viene poi usata ricorsivamente all'interno della *PointNet++[11]*,la versione successiva della *PointNet* dove si vanno a dividere le nuvole in più set dai quali vengono poi estratte le caratteristiche. Queste verranno poi raggruppate per produrre delle caratteristiche ad un livello superiore fino ad ottenere una classificazione dell'intero dataset. All'interno di questo studio è stata utilizzata la *PointNet* per effettuare la classificazione poichè presenta un ottimo *tradeoff*, nonostante attualmente siano disponibili delle reti più recenti e con performance migliori, come la *DGCNN[12]*. Il tradeoff indica il compromesso che deve essere fatto in termini di complessità, durata temporale e accuratezza. Infatti aumentando la complessità la rete diventa più sensibile alle piccole variazioni ma non appena deve generalizzare su degli oggetti mai viste prima i risultati sono pessimi. Questo determinerà una cattiva accuratezza del modello. Viceversa, diminuire la complessità ed aumentare l'accuratezza fa si che i risultati in fase di training siano piuttosto bassi.

2.4 Approcci generativi per le nuvole di punti

Gli approcci generativi [13] hanno come scopo principale quello di aumentare il set di dati su cui si lavora, ottenendo perciò una maggiore qualità. Tra le tecniche più semplici e comuni degli approcci generativi o, più in generale della data augmentation, si possono trovare le trasformazioni geometriche:

- Traslazione
- Rotazione
- Capovolgimento
- Trasformazioni spaziali
- Variazione della luminosità
- Iniezione del rumore

di cui vengono riportati degli esempi visivi nella figura 2.5.

Figura 2.5: Esempio di trasformazioni geometriche

In questo modo possono essere generati dei nuovi dati da riutilizzare per l'addestramento delle reti neurali. Un dettaglio da chiarire è che non ha molto senso applicare questo tipo di approccio "semplificato" alle nuvole di punti: risulta infatti molto più difficile utilizzarli rispetto alle immagini ed è anche concettualmente errato. Infatti ruotare un oggetto qualsiasi, come per esempio una colonna, è un'operazione priva di senso in quest'ambito. Per questo motivo sono state impiegate le GAN, grazie alle quali sono state prodotte delle nuvole verosimili alle originali. Le *Generative Adversarial Network* sono essenzialmente una classe di algoritmi che sfrutta due reti neurali, portate a scontrarsi tra loro. Sono un framework che sfrutta un processo avversario, allenando il generatore e il discriminatore: il primo ha come scopo quello di produrre degli oggetti il più realistici possibili che verranno poi dati come input al discriminatore che, come il nome suggerisce, dovrà distinguere quali sono gli oggetti generati e quali quelli reali. Più precisamente, una volta creata una distribuzione gaussiana dei dati durante l'addestramento, è possibile utilizzare il generatore: inizialmente questo è alimentato da un vettore di numeri casuali, che viene poi trasformato in base alla distribuzione prodotta in precedenza. Quando il modello viene addestrato, lo spazio vettoriale sarà una versione compressa della distribuzione dei dati, definita spazio latente. Il modello GAN sfrutterà questo spazio per estrarre dei punti che verranno usati come dei nuovi dati per il generatore. Perciò il discriminatore è alimentato dall'intero insieme di dati con cui si effettua l'addestramento in cui appunto compaiono sia elementi reali che generati. Lo schema generale di funzionamento è rappresentato dalla figura 2.6. Lo scopo è quello di riuscire a produrre degli oggetti che non siano distinguibili da quelli reali.

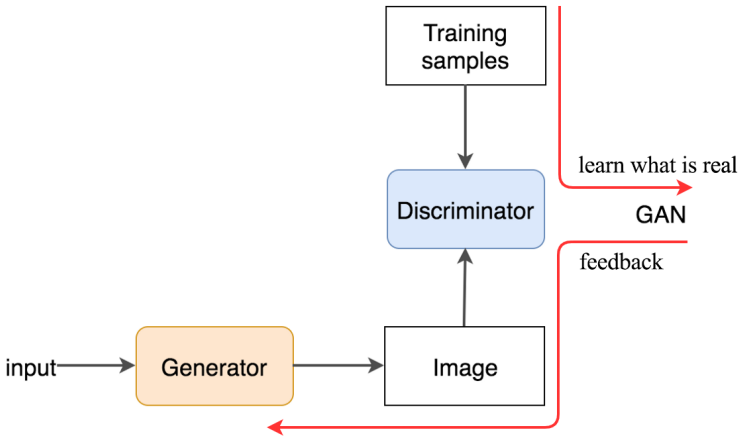

Figura 2.6: Schema di funzionamento delle GAN

Uno dei motivi principali per cui vengono utilizzati questi approcci generativi è rappresentato dalla differenza tra le GAN stesse e gli altri tipi di reti: con le prime è possibile creare dei nuovi esempi per le diversi classi, completando i task definiti come data augmentation, senza limitarsi perciò alla classificazione degli oggetti. Un problema che molto spesso si riscontra lavorando con queste reti è la saturazione: in

questo caso il generatore non riuscirà più a produrre degli esempi realistici, per una progettazione poco efficiente della funzione di loss.

Un altro approccio generativo che viene usato molto spesso in alternativa alle GAN è rappresentato dai *Variational Autoencoders (VAE)*.

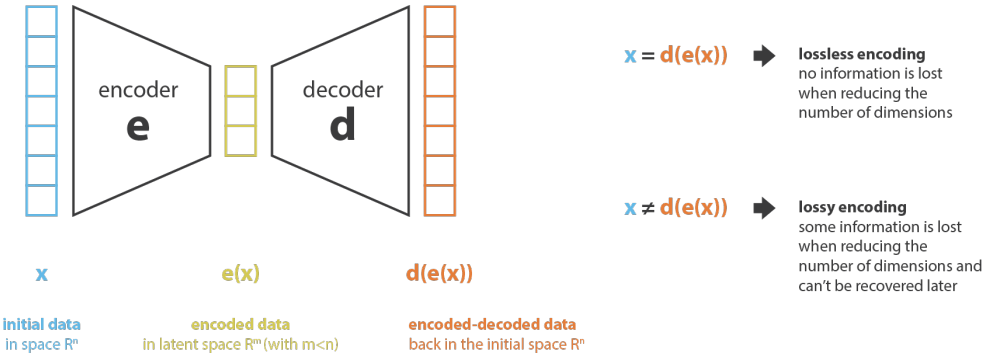

Figura 2.7: Schema di funzionamento delle VAE

Come illustrato nella figura 2.7, vengono sfruttati un encoder e un decoder come due reti neurali con lo scopo rispettivamente di massimizzare il contenuto informativo nella fase di encoding e di minimizzare l'errore nella fase di decoding. Si cerca perciò la loro combinazione migliore attraverso un processo di ottimizzazione iterativo. La generazione di nuovi oggetti può essere fatta prendendo casualmente dei punti dallo spazio latente, ovvero quello dove l'encoder va a comprimere i dati che prende in input, per poi decodificarli. Il problema principale di questo metodo è che per fare una cosa del genere si necessita di uno spazio latente che possa essere considerato regolare, risultato difficile da raggiungere. Per farlo si deve introdurre una regolarizzazione durante la fase di training in modo tale da ottenere uno spazio latente utilizzabile per la generazione.

Come affermato in precedenza, lavorando con le nuvole di punti , molti degli approcci sviluppati per le immagini non sono applicabili con questo tipo di rappresentazione, rendendo questo task il più impegnativo dello studio.

2.4.1 TreeGAN

In un primo momento sono state approfondite la *TreeGAN[14]* e la *SpectralGAN[15]* , per poi sostituire quest'ultima con la *SetVAE[4]* per problemi di documentazione del codice. Attraverso la *TreeGAN* è stato effettuato un primo task di data augmentation sul dataset proposto di default, lo *ShapeNet*[5], per vedere la qualità degli oggetti generati, controllando anche la variazione delle metriche.

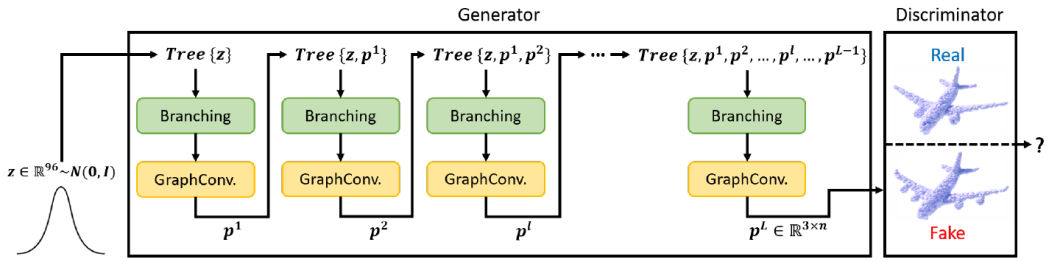

Figura 2.8: Architettura della TreeGAN

Questa rete, la cui struttura è presentata nella figura 2.8 sfrutta le informazioni degli *"antenati"* dei punti per definire delle features, estratte tra i vari strati convoluzionali, più dettagliate, che fungono poi da supporto per la rappresentazione delle nuvole di punti. Una particolarità di questa rete è che si possono generare gli oggetti senza doverla allenare separatamente su ogni classe. Viene introdotta la *Frechet Point Cloud Distance (FPD)*, una metrica specifica di questa rete per valutarne l'aspetto quantitativo, con la quale si misura la distanza nel dominio delle features tra la distribuzione reale e quelle generata.

$$FPD = \|m_p - m_Q\|_2^2 + Tr(\Sigma_P + \Sigma_Q - 2(\Sigma_P\Sigma_Q)^{\frac{1}{2}}) \tag{2.1}$$

Dove m_p e m_q sono i vettori medi calcolati rispettivamente dalle nuvole reali e da quelle generate. Σ_P e Σ_Q rappresentano le matrici di covarianza calcolate a partire dalle nuvole reali e da quelle generate. Infine $Tr(A)$ indica la somma degli elementi sulla diagonale principale della matrice A.

2.4.2 SetVAE

Per problemi di strutturazione del dataset *ArCH*, dove ricadeva l'interesse di aumentare il volume di dati, non si è potuta utilizzare la rete sopra citata poichè richiedeva una suddivisione algoritmica delle varie parti degli oggetti. Con questa motivazione è stata impiegata la *SetVAE*. Alcune caratteristiche che hanno reso l'utilizzo di questa rete piuttosto interessante sono state il recente sviluppo(2021) e la qualità degli oggetti prodotti: allo stato dell'arte questa rete presenta una produzione di oggetti di qualità superiore, in quanto le nuvole generate hanno una maggiore densità di punti. La rete è capace di estrarre delle relazioni interessanti sul set considerato in input, operando in modo non supervisionato. Inoltre un vantaggio non indifferente di questa rete è che può lavorare con un qualsiasi dataset strutturato, senza richiedere una specifica configurazione come altre reti.

Capitolo 3

Metodologia

All'interno di questo capitolo viene presentato il workflow del lavoro proposto, rappresentato in figura 3.1, per poi descrivere nel dettaglio ciò che è stato fatto all'interno di ogni fase. Perciò si inizia illustrando i dataset utilizzati, per poi mostrare un esempio del lavoro di etichettatura svolto manualmente sul dataset *ArCH*. Successivamente vengono introdotte le reti generative utilizzate per passare infine alla rete usata per effettuare la classificazione dei dataset standard ed aumentati.

Figura 3.1: Workflow

3.1 Dataset utilizzati

Nel campo del deep learning esistono diversi dataset disponibili per le nuvole di punti, ognuno dei quali può essere utilizzato per affrontare task differenti. Tra i più conosciuti, oltre a quelli che verranno introdotti successivamente, è possibile trovare:

- *ModelNet[16]*,

- *ScanNet[17]*
- *S3DIS[18]*

Il primo è un dataset ben strutturato, con un discreto numero di categorie di oggetti. Il secondo è un dataset RGB-D video con più di 1500 scansioni e 2.5 milioni di punti, utili soprattutto per i task di classificazione e segmentazione. Infine, *S3DIS* è un dataset con 271 nuvole di punti rappresentanti delle stanze e con 13 diverse categorie semantiche.

3.1.1 ShapeNet

.

Il primo dataset utilizzato è stato lo *ShapeNet[5] [19]*, uno dei più utilizzati all'interno del contesto delle nuvole di punti. Esistono due versioni di questo dataset che differiscono in termini di dimensione: *ShapeNetCore* e *ShapenetSem*. Il primo dei due rappresenta la versione completa del dataset, con più di 51000 modelli unici classificati in 55 classi, presentate nella tabella 3.1 che rappresentano degli oggetti comuni. Il secondo invece rappresenta una versione ridotta, con un numero di classi minore. In particolare, in questo studio viene usata la versione *Core*, ristretta però a solo 16 classi, elencate nella tabela 3.2: il motivo per cui è stato utilizzato questo particolare dataset è che viene indicato nei paper delle reti, come la *TreeGAN* e *SetVAE*.

Tabella 3.1: ShapeNetCore

ID	**Classe**	**N. oggetti**
04379243	table	8443
02958343	car	7497
03001627	chair	6778
02691156	airplane	4045
04256520	sofa	3173
04090263	rifle	2373
03636649	lamp	2318
04530566	watercraft	1939
02828884	bench	1816
03691459	loudspeaker	1618
02933112	cabinet	1572
03211117	display	1095
04401088	telephone	1052
02924116	bus	939
02808440	bathtub	857
03467517	guitar	797

03325088	faucet	744
03046257	clock	655
03991062	flowerpot	602
03593526	jar	597
02876657	bottle	498
02871439	bookshelf	466
03642806	laptop	460
03624134	knife	424
04468005	train	389
02747177	trash bin	343
03790512	motorbike	337
03948459	pistol	307
03337140	file cabinet	298
02818832	bed	254
03928116	piano	239
04330267	stove	218
03797390	mug	214
02880940	bowl	186
04554684	washer	169
04004475	printer	166
03513137	helmet	162
03761084	microwaves	152
04225987	skateboard	152
04460130	tower	133
02942699	camera	113
02801938	basket	113
02946921	can	108
03938244	pillow	96
03710193	mailbox	94
03207941	dishwasher	93
04099429	rocket	85
02773838	bag	83
02843684	birdhouse	73
03261776	earphone	73
03759954	microphone	67
04074963	remote	67
03085013	keyboard	65
02834778	bicycle	59
02954340	cap	56

Tabella 3.2: Le 16 classi di ShapeNetCore utilizzate dalle reti.

ID	**Classe**	**N. oggetti**
02691156	Airplane	2690
02773838	Bag	76
02954340	Cap	55
02958343	Car	1824
03001627	Chair	3746
03261776	Earphone	69
03467517	Guitar	787
03624134	Knife	392
03636649	Lamp	1546
03642806	Laptop	445
03790512	Motorbike	202
03790512	Mug	184
03948459	Pistol	275
04099429	Rocket	66
04225987	Skateboard	152
04379243	Table	5266

Figura 3.2: Esempio degli oggetti di diversi classi dello ShapeNet

Come si può vedere dalla figura 3.2, il dataset *ShapeNet* presenta diversi classi di oggetti comuni, come per esempio sedie, aerei, cappelli, tavoli. I modelli contenuti all'interno di queste classi sono poi suddivisi secondo le varie componenti dell'oggetto stesso. Per rendere il concetto più chiaro viene presentato l'esempio di un aereo composto da varie parti come la coda, le ali, i motori e la struttura principale.

3.1.2 ArCH

Come già accennato, una delle grandi carenze dell'ambito dei beni culturali è proprio la mancanza di un dataset etichettato con il quale possa essere effettuata la classificazione. Perciò per svolgere i task proposti è stato necessario effettuare manualmente un'etichettatura sull'unico dataset pubblico disponibile l' *ArCH*. Questo dataset è composto complessivamente da 17 scene: 15 adibite al training e 2 da utilizzare per il testing. Le nuvole che rappresentano queste scene, oltre alle 3 coordinate utilizzate per rappresentare i punti, forniscono ulteriori caratteristiche date dal colore, in formato RGB, e le normali dei punti. Attraverso il software CloudCompare e dopo aver scaricato il dataset, è stato possibile aprire le singole scene e visualizzare interamente le nuvole di punti.

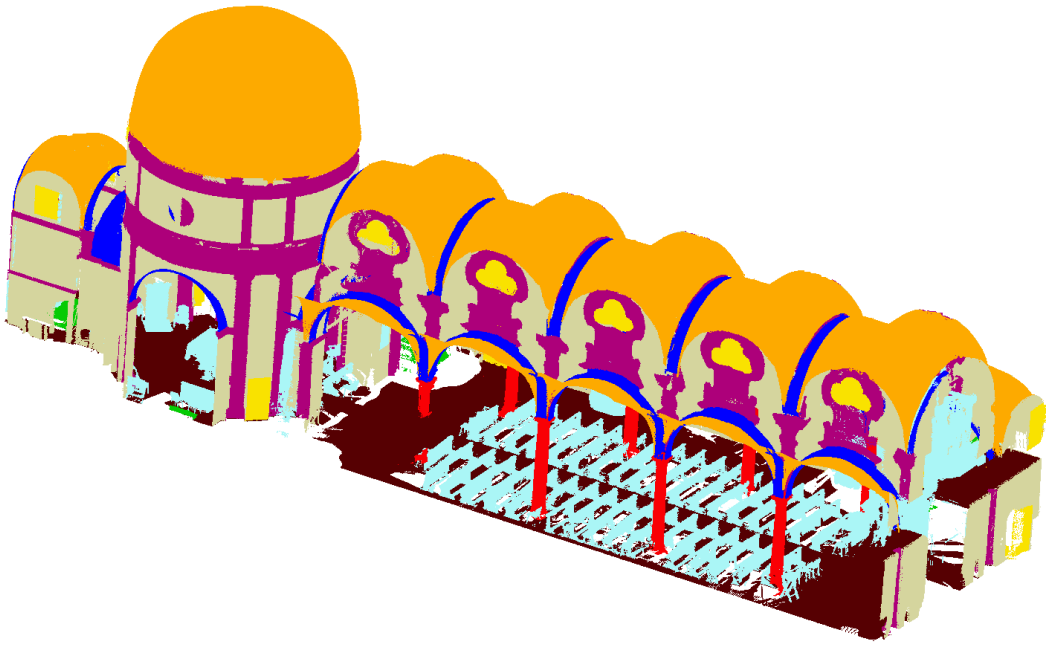

Figura 3.3: Scena 2-TR-church

Come si può vedere dalla figura 3.3 gli oggetti presenti all'interno della scena sono rappresentati con colori diversi in base alla classe di appartenenza. La totalità delle classi è composta da:

1. arch
2. column
3. moldings
4. floor
5. door-window
6. wall
7. stairs
8. vault
9. roof
10. other

Sono stati perciò individuati gli oggetti di tutte le scene e raggruppati all'interno delle relative classi.

In un primo momento sono state ipotizzate le classi su cui si sarebbe fatta data augmetation, evidenziandole all'interno di 3.3. La scelta è ricaduta su queste due

Tabella 3.3: Numero di oggetti di ogni classe del dataset ArCH

Classi	**N. Oggetti**
Arch	284
Column	224
Moldings	609
Floor	81
Doors-Windows	632
Wall	217
Stairs	65
Vault	149
Roof	53
Others	157

classi per lo scarso numero di oggetti, a cui potrebbero corrispondere delle metriche di classificazione relativamente basse.

3.2 Esempio del lavoro di etichettatura svolto sul dataset ArCH

Grazie al software CLOUD COMPARE è stato possibile ritagliare manualmente tutti i singoli oggetti delle nuvole di punti, per poi raggrupparle all'interno delle classi. Di seguito viene proposto un esempio visivo del lavoro svolto, prendendo come riferimento la scena di partenza dell'immagine 3.4. Vengono riportate, nel seguente ordine, gli oggetti della classe *columns*, figura 3.5, quelli della classe *doors-windows*, figura 3.6, quelli della classe *vault*, figura 3.7 ed infine quelli della classe *floor*, figura 3.8

Figura 3.4: Scena 3-VAL-ROOM

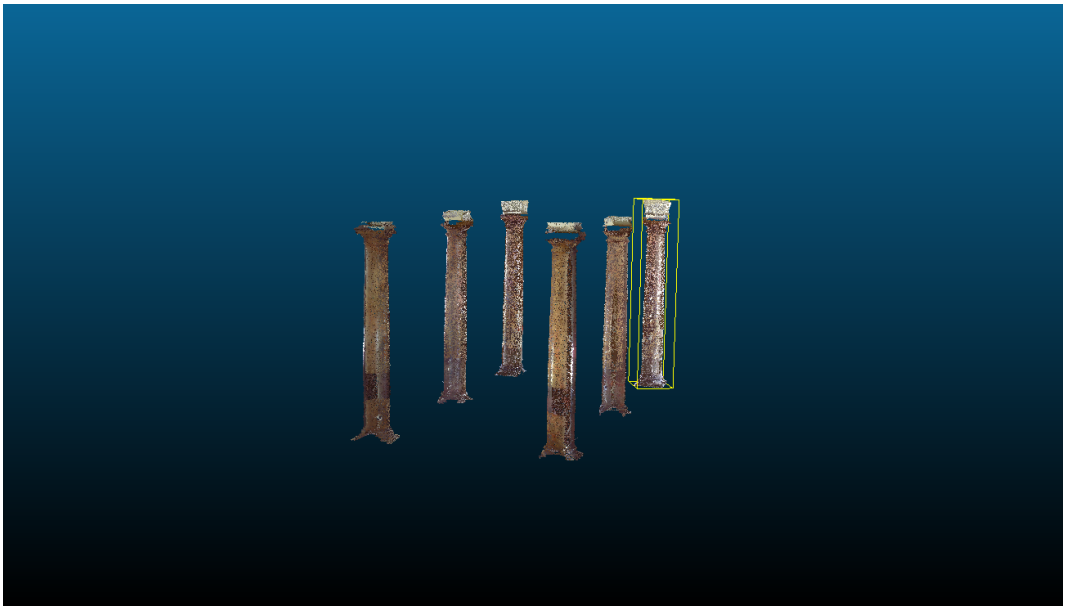

Figura 3.5: Colonne delle scena 3

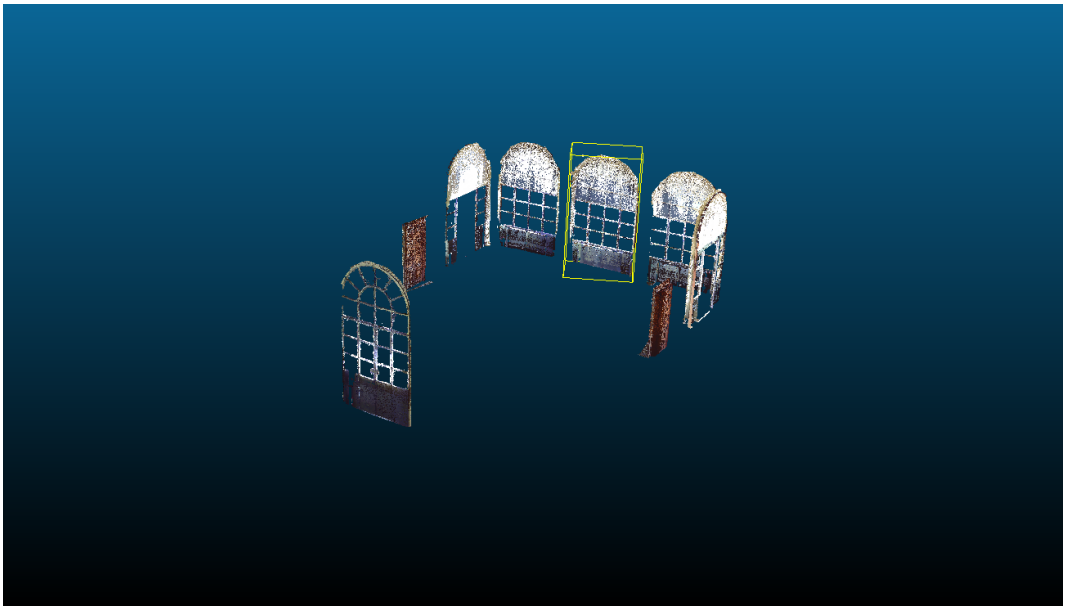

Figura 3.6: Porte finestre della scena 3

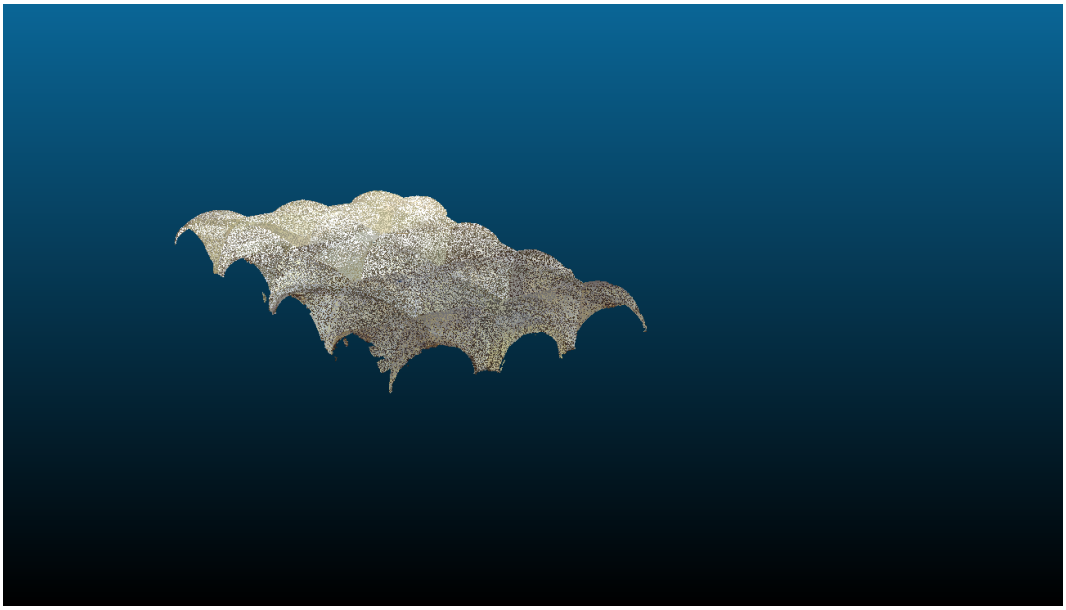

Figura 3.7: Volte della scena 3

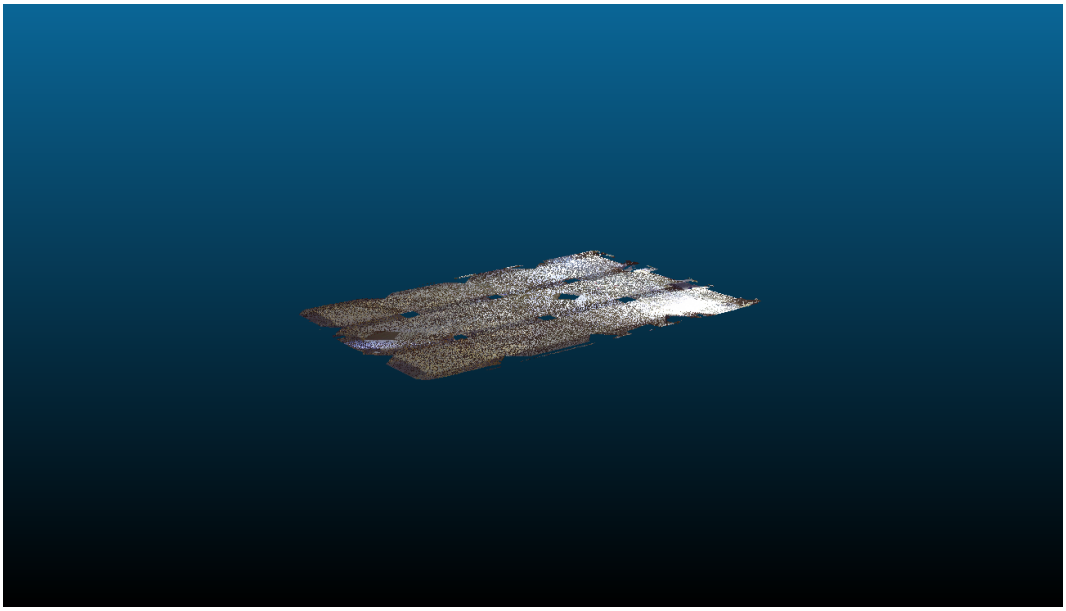

Figura 3.8: Pavimento della scena 3

3.3 Statistiche sul dataset

Dopo aver etichettato tutto il dataset sono state svolte delle statistiche su di esso: nel dettaglio sono stati calcolati il numero di punti per ogni classe di ogni scena e il numero di oggetti per ogni classe di ogni scena.

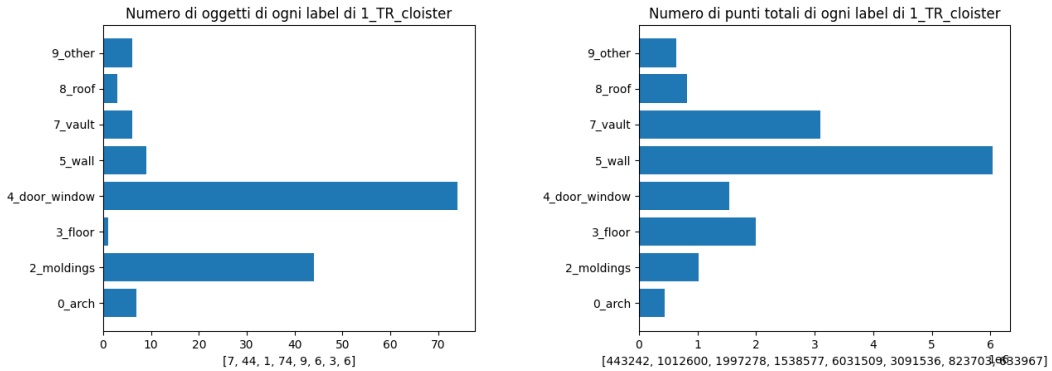

Figura 3.9: Statistiche sulla scena 1 del dataset

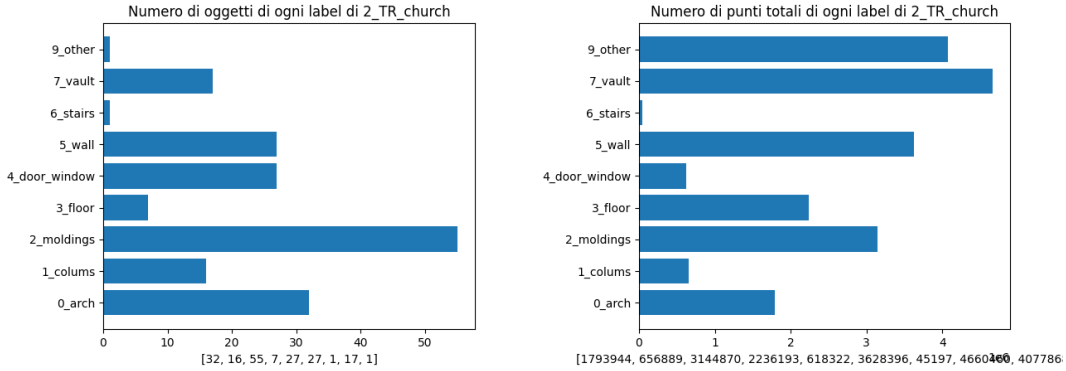

Figura 3.10: Statistiche sulla scena 2 del dataset

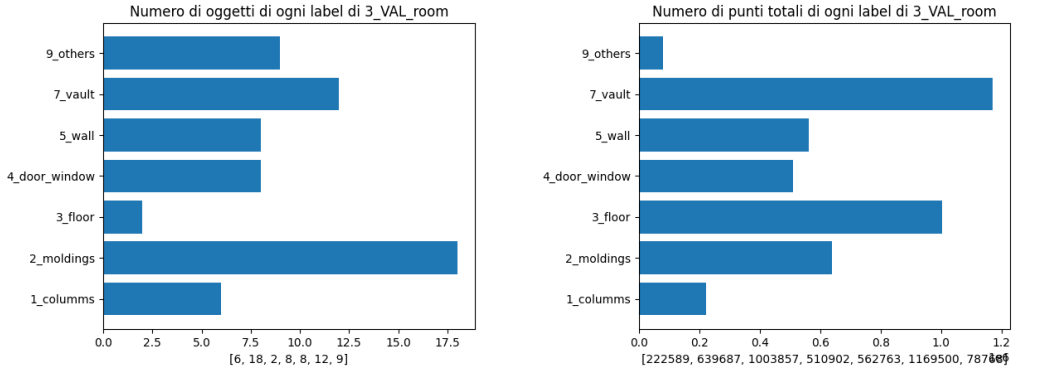

Figura 3.11: Statistiche sulla scena 3 del dataset

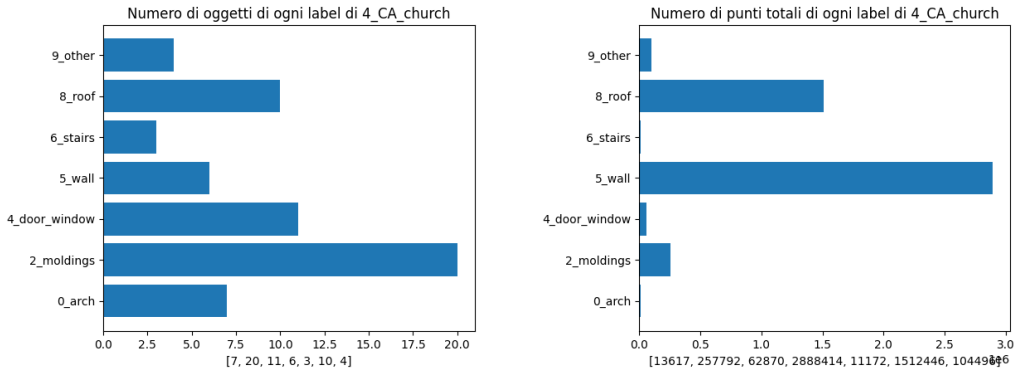

Figura 3.12: Statistiche sulla scena 4 del dataset

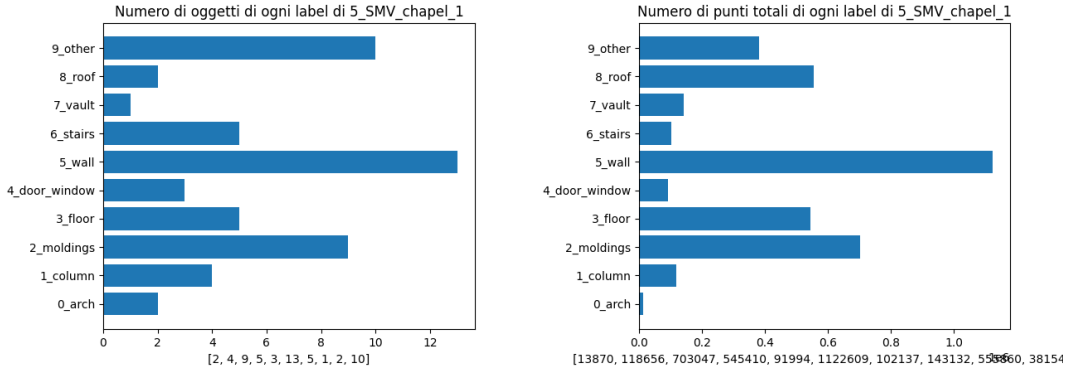

Figura 3.13: Statistiche sulla scena 5 del dataset

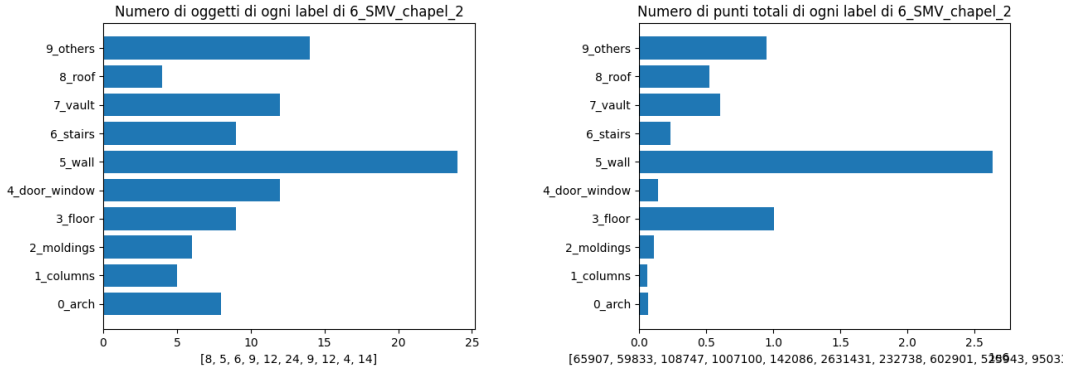

Figura 3.14: Statistiche sulla scena 6 del dataset

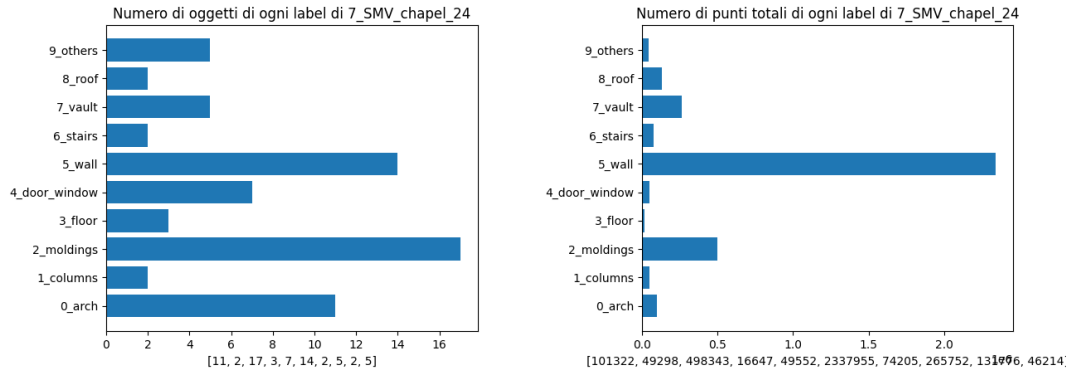

Figura 3.15: Statistiche sulla scena 7 del dataset

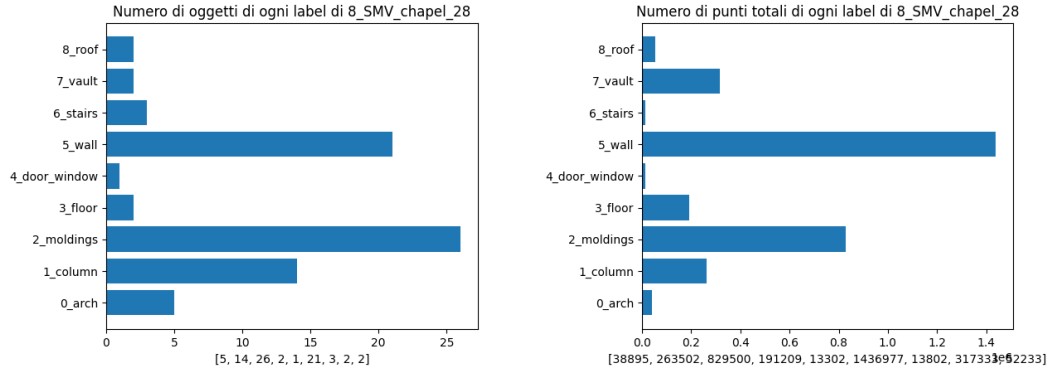

Figura 3.16: Statistiche sulla scena 8 del dataset

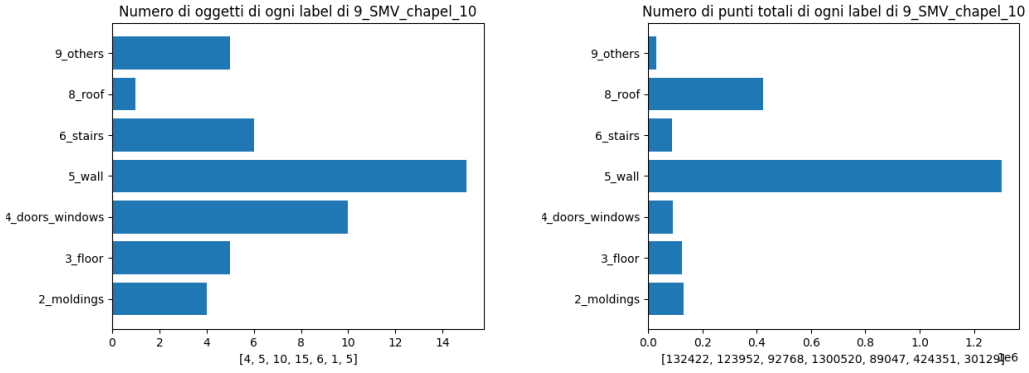

Figura 3.17: Statistiche sulla scena 9 del dataset

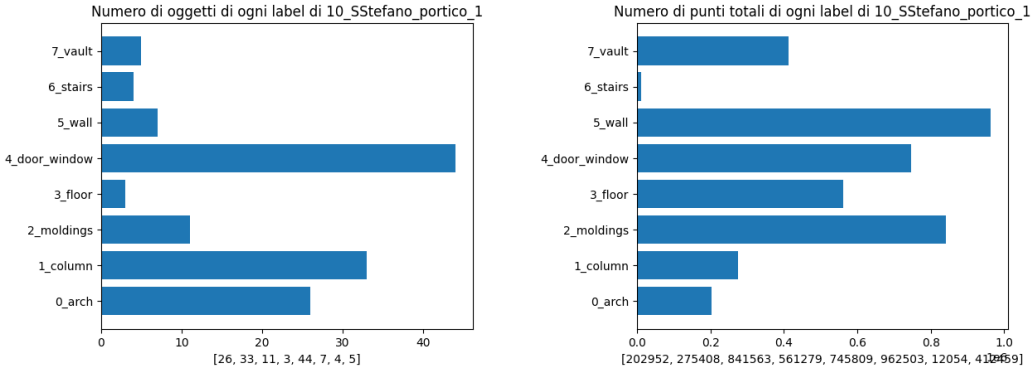

Figura 3.18: Statistiche sulla scena 10 del dataset

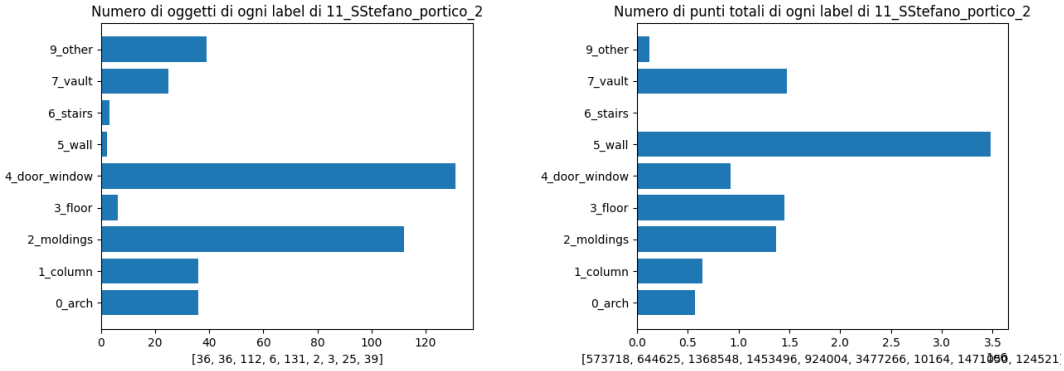

Figura 3.19: Statistiche sulla scena 11 del dataset

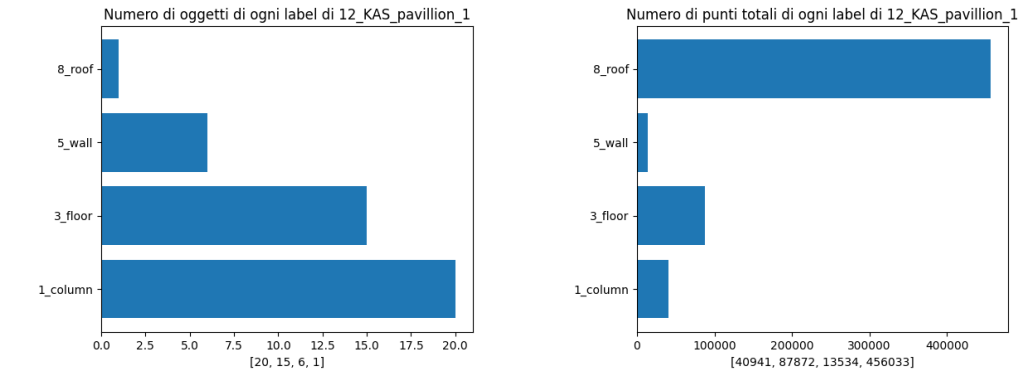

Figura 3.20: Statistiche sulla scena 12 del dataset

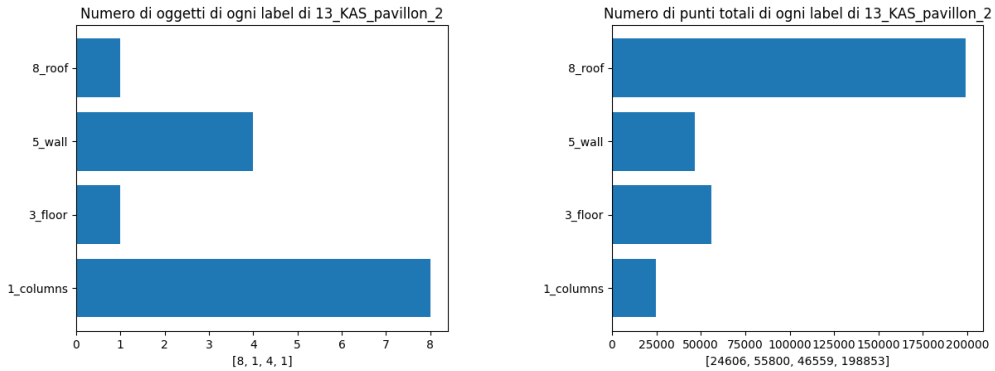

Figura 3.21: Statistiche sulla scena 13 del dataset

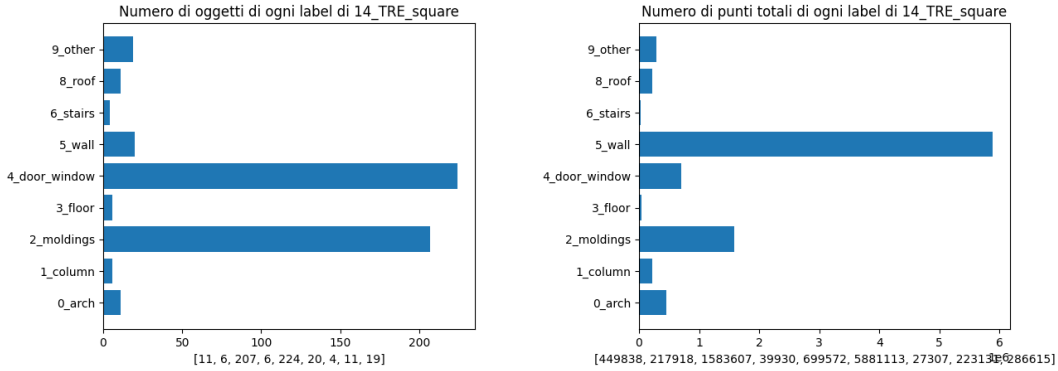

Figura 3.22: Statistiche sulla scena 14 del dataset

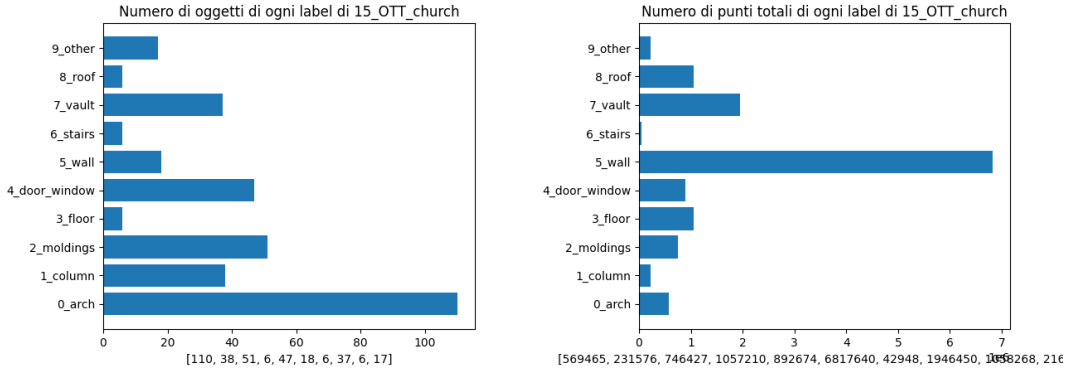

Figura 3.23: Statistiche sulla scena 15 del dataset

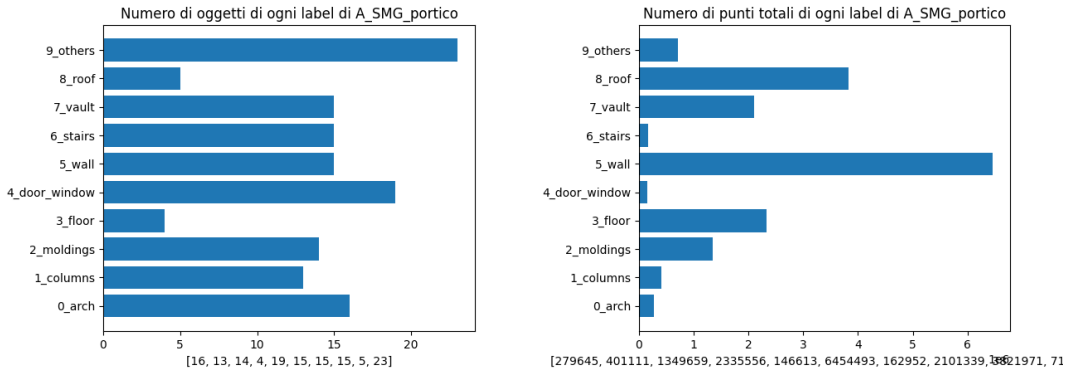

Figura 3.24: Statistiche sulla scena di test A

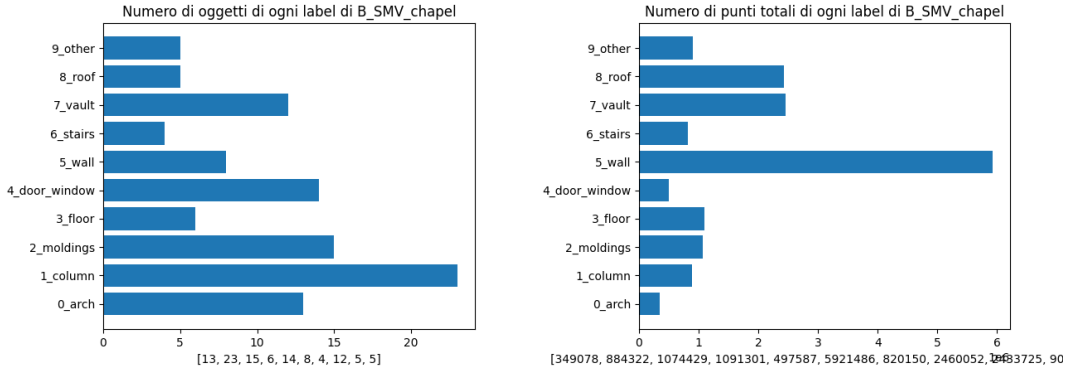

Figura 3.25: Statistiche sulla scena di test B

Come si può vedere dalla Fiura 3.17 la classe *stairs* ha un discreto numero di oggetti ma pochi punti. Questo dimostra come in realtà sia importante che gli oggetti siano composti da un numero considerevole di punti in modo tale da poterne generare dei nuovi con una qualità accettabile. Questa considerazione può essere estesa anche alle altre classi di ogni scena.

3.4 Reti generative utilizzate

3.4.1 Utilizzo della TreeGAN

Prima di utilizzare la rete con il dataset *ArCH*, è stata testata con il dataset di riferimento proposto dal paper, lo *ShapeNetCore*, per poi apportare le adeguate modifiche ed utilizzarla con gli scopi proposti. Dopo aver configurato la rete è stato eseguito un primo tentativo di training: non avendo a disposizione delle risorse consone per lavorare in tempi ragionevoli, si è utilizzato il modello pre-allenato indicato nel paper. Tuttavia, disponendo dei pesi inerenti solo alle classi *Airplane*

e *Chair*, è stato necessario allenare la rete anche sulle altre classi, così da poterne scegliere una qualsiasi su cui poi effettuare la data augmentation. Una volta scelta la classe sono stati generati degli oggetti: è importante valutare correttamente il numero di oggetti da aggiungere poichè, se in numero eccessivo rischiano di influenzare negativamente il comportamento della rete. Proporre una mole eccessiva di dati generati alla rete, e perciò un dataset significativamente diverso da quello originale, porterebbe a compiere un addestramento su oggetti che non rispecchiano poi la realtà. Per questo motivo sono stati aggiunti solo 10 nuovi oggetti della classe *rocket*, ovvero quella da aumentare. Attraverso una nuova classificazione con la *PointNet* si può notare come i valori delle metriche siano sensibilmente migliorati.

3.4.2 Utilizzo della SetVAE

Il limite però della *TreeGan* è rappresentato dalla necessità di avere un dataset strutturato: gli oggetti all'interno delle scene dovevano essere suddivisi in varie parti.. Non disponendo di un algoritmo che potesse risolvere questo problema è stata individuata un'altra rete, la *SetVAE* con cui poter portare a termine il task di data augmentation sul dataset *ArCH*. Come già accennato in precedenza, una particolarità di questa rete è rappresentata proprio dal poter utilizzare un dataset qualsiasi, senza che esso venga strutturato in un modo specifico. Per aggiungere il dataset di interesse(*ArCH)* tra quelli su cui lavorare, sono stati modificati gli script relativi allo *ShapeNet*, già presenti all'interno della repository della rete. Non disponendo di un modello già pre-allenato, nella fase di training e di testing sono state specificate le classi dove si sarebbe poi aumentato il numero di oggetti. La scelta di queste classi viene ipotizzata in un primo momento basandosi sul numero di oggetti, per poi essere confermata o meno mediante il calcolo delle metriche della classificazione. Per poter usufruire del dataset *ArCH*, si è effettuata una traslazione nel punto (0,0,0) di tutte le nuvole di punti, per poi fare uno split con le seguenti precentuali: *80% per il training*, *10% per il testing* e *10% per la validation.* Lo script utilizzato per lo split del dataset e la traslazione delle nuvole di punti è il seguente:

```
import os, glob
import numpy as np
import random
import pickle

base_path = "Path del dataset in locale"

train_set = 0.8
valid_set = 0.1
test_set = 0.1

X_train=[]
```

```
X_valid=[]
X_test=[]
Y_train=[]
Y_valid=[]
Y_test=[]

def crea_oggetto(o,n_punti=2048):
    punti = []
    with open(o, "r") as fr:
        for l in fr:
            s = l.strip().split(" ")
            x = float(s[0])
            y = float(s[1])
            z = float(s[2])
            punti.append([x, y, z])

    #Prendo i punti in maniera casuale
    if n_punti < len(punti):
        punti_scelti = random.sample(punti, n_punti)
    else:
        punti_scelti = random.choices(punti, k=n_punti)

    punti_scelti = np.array(punti_scelti)
    #Tolo i minimi, per spostare l'oggetto in (0,0,0)
    minimi = np.min(punti_scelti, axis=0)
    punti_scelti = punti_scelti - minimi
    return punti_scelti
```

A questo punto si ottiene la traslazione delle nuvole di punti. Il codice per lo split è il seguente:

```
classi = sorted(glob.glob(base_path+"*"))

for i_classe,classe in enumerate(classi):
    oggetti = glob.glob(classe+"/*")
    n_train = int(len(oggetti)*train_set)
    n_valid = int(len(oggetti)*(train_set+valid_set))
    n_test = len(oggetti)
    print("\nClasse {} con {} oggetti: {} --> train:{} - valid:{} - test: {
    #train
    print(" - train...")
    for i in range(n_train):
        o = oggetti[i]
```

```
            punti = crea_oggetto(o,n_punti=2048)
            X_train.append(punti)
            Y_train.append(i_classe)

        #valid
        print(" - valid...")
        for i in range(n_train, n_valid):
            o = oggetti[i]
            punti = crea_oggetto(o,n_punti=2048)
            X_valid.append(punti)
            Y_valid.append(i_classe)

        #test
        print(" - test...")
        for i in range(n_valid, n_test):
            o = oggetti[i]
            punti = crea_oggetto(o,n_punti=2048)
            X_test.append(punti)
            Y_test.append(i_classe)

print("\nConversione in numpy array...")
X_train = np.array(X_train)
X_valid = np.array(X_valid)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
Y_valid = np.array(Y_valid)
Y_test = np.array(Y_test)

print("Salvataggio...")
with open('split_arch.pickle', 'wb') as handle:
    pickle.dump([X_train, X_valid, X_test,
    Y_train, Y_valid, Y_test],
    handle, protocol=pickle.HIGHEST_PROTOCOL)
```

3.5 Classificazione

3.5.1 Utilizzo della PointNet

Dopo aver etichettato l'intero dataset *ArCH*, è stata studiata la rete *PointNet*, così da avere una rete con la quale poter fare la classificazione del dataset. Come nel caso delle altre reti, in un primo momento si è lavorato con il dataset proposto dal paper, lo *ShapeNetCore*, per poi passare all'*ArCH*.

Tabella 3.4: Dataset ShapeNetCore

ID	**Classe**	**N. oggetti**
02691156	Airplane	2690
02773838	Bag	76
02954340	Cap	55
02958343	Car	1824
03001627	Chair	3746
03261776	Earphone	69
03467517	Guitar	787
03624134	Knife	392
03636649	Lamp	1546
03642806	Laptop	445
03790512	Motorbike	202
03790512	Mug	184
03948459	Pistol	275
04099429	Rocket	66
04225987	Skateboard	152
04379243	Table	5266

Nella tabella 3.4 è stata evidenziata la probabile classe da aumentare per via dello scarso numero di oggetti. La conferma potrà arrivare solo una volta effettuata la classificazione, verificando anche attraverso le metriche se sia o meno la scelta più giusta. Inoltre vengono calcolate anche l'accuracy del training e della validation: in presenza di valori non eccessivamente alti si potrebbe aumentare il numero di epoche dell'allenamento per tentare di incrementarle.

3.5.2 Data augmentation

Una volta generati gli oggetti, è stato il momento di aggiungerli al dataset per effettuare una nuova classificazione e prendere nota di eventuali miglioramenti. In questo passaggio gli oggetti generati sono stati manipolati in diversi modi: al termine della generazione con la TreeGAN viene salvato un file chiamato *"vis-log"* con all'interno le nuvole. Per poter visualizzare le nuvole, questo file è stato caricato, attraverso *Visdom*, uno strumento che permette la visualizzazione live di numerosi dati, su un server remoto, così da poter leggere e salvare le coordinate dei punti in file *".txt"*.

```
import visdom
vis = visdom.Visdom()
vis.replay_log("path del file vis-log")
```

Diversamente, con l'utilizzo della *SetVAE* gli oggetti generati vengono salvati in vettori *"numpy"* e anche qui devono essere convertiti in un altro tipo di file per essere visualizzati.

3.5.3 Metriche della classificazione

A questo punto vengono utilizzate le principali metriche di valutazione dello stato dell'arte , relative al task di classificazione, quali:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{3.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

$$F1 = 2 \times \frac{Prec \times Recall}{Prec + Recall} \tag{3.4}$$

dove TP indicano i Veri Positivi, i casi positivi riconosciuti come tali, FP Falsi Positivi,i casi negativi riconosciuti come positivi, FN Falsi Negativi, i casi positivi riconosciuti come negativi, e TN i Veri Negativi, i casi negativi riconosciuti correttamente dall'algoritmo. Specificando un determinato valore di soglia gli oggetti verranno assegnati ad una determinata classe. La metrica *(3.1)* descrive in modo generale come il modello performi tra le varie classi ed è utile quando tutte le classi hanno la stessa importanza, come in questo caso. La metrica *(3.2)* misura l'accuratezza del modello nel classificare un sample come positivo. Quando la precisione è alta, si considera il modello affidabile, capace quindi di distinguere correttamente gli oggetti. La *(3.3)* misura l'abilità del modello nel determinare i samples positivi. Più è alta la *Recall* e maggiore è il numero di samples positivi trovati. Infine la *(3.4)* rappresenta una sorta di combinazione tra *recall* e *precision*. Il caso migliore che possa verificarsi è quello in cui i valori dell'accuracy e dell'F1 siano abbastanza alti, così da avere la conferma che la rete produca dei buoni risultati. In genere i dati sulle predizioni del modello vengono rappresentati mediante la *Matrice di confusione*, figura 3.26, dove nelle colonne vengono riportate le previsioni che il modello effettua e sulle righe lo stato effettivo. Questa tabella permette di vedere facilmente se sono stati commessi errori di classificazione e se le previsioni sono più o meno corrette. Si ha quindi la possibilità di comprendere le performance del modello predittivo, per poi intervenire se ce ne fosse la necessità.

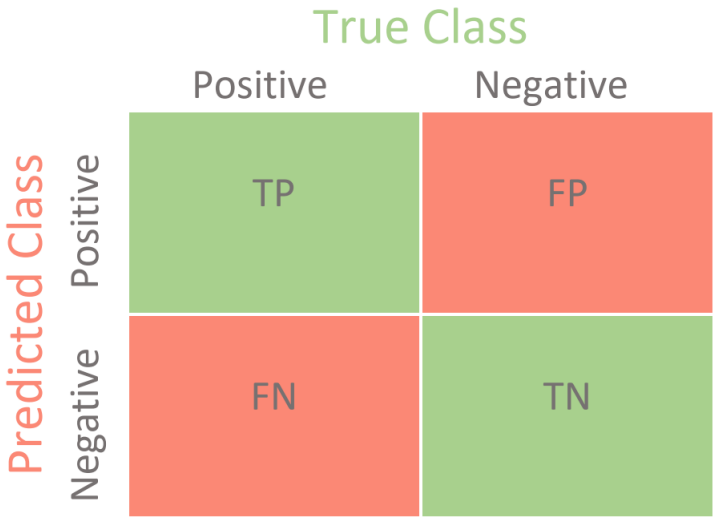

Figura 3.26: Esempio di una matrice di confusione

Con la figura 3.26 viene rappresentata la tabella per una classificazione binaria: ovviamente questa può essere estesa in caso sia necessario effettuare una classificazione multiclasse.

L'ultima metrica da citare è la *Loss* ovvero la differenza tra l'output desiderato e quello calcolato, avendo a disposizione una misura dell'errore della rete. Questa metrica viene utilizzata per sistemare i valori dei pesi, calcolati nella fase di training, andando a ritroso a partire dall'output fino a risalire al primo strato (*Backward Propagation*). Perciò il *Learning Rate* rappresenta l'entità della variazione dei valori dei pesi, che può essere più o meno estesa. Nel caso della *PointNet* la funzione di *Loss* utilizzata è la *Sparse Categorical Cross Entropy*: questo tipo di funzione di *Loss* è particolarmente utilizzata quando l'appartenenza degli oggetti ad una classe è mutuamente esclusiva, ovvero un oggetto può essere classificato con una e una sola classe.

$$L(\theta) = -\sum_{i=1}^{outputsize} y_i \cdot \log \hat{y}_i \tag{3.5}$$

Dove $\hat{y}_i$ rappresenta l'*i-esimo* valore scalare dell'output e y_i il valore atteso.

Capitolo 4

Risultati e discussioni

In questo capitolo vengono riportati i risultati ottenuti in merito ai lavori svolti ed indicati nel capitolo 3. Le versioni di *Python* utilizzate sono state diverse, in base alla rete. In alcuni casi sono state usate le versioni indicate nel paper delle reti, in altri sono state cambiate per problemi di compatibilità con le librerie e con Colab, uno strumento che permette di eseguire del codice *Python* sul browser, con un accesso gratuito alla GPU, una condivisione semplificata e senza dover fare alcun tipo di configurazione. Sono stati poi utilizzati i framework *Pytorch[20]*, una libreria con la quale si possono utilizzare i tensori,sfruttando le GPU e le CPU, nell'ambito del deep learning, *TensorFlow[21]*, che mette a disposizione librerie e risorse per l'apprendimento automatico e *Keras [22]* Per quanto riguarda la parte hardware, sono state utilizzate principalmente le risorse grafiche di colab

- Tesla T4, 15 GB Dedicati, 12 GB di RAM

e il server messo a disposizione dll'Università Politecnica delle Marche

- PU NVIDIA 2080 TI, 11 GB Dedicati, 128 GB di RAM

4.1 Risultati della TreeGAN

Nel capitolo precedente è stata anticipata la classe su cui si sarebbe svolta la data augmentation all'interno del dataset *ShapeNet*, ovvero la classe *rocket.* Dopo un allenamento di 1000 epoche sono stati utilizzati i pesi calcolati per la generazione con dei risultati piuttosto validi:

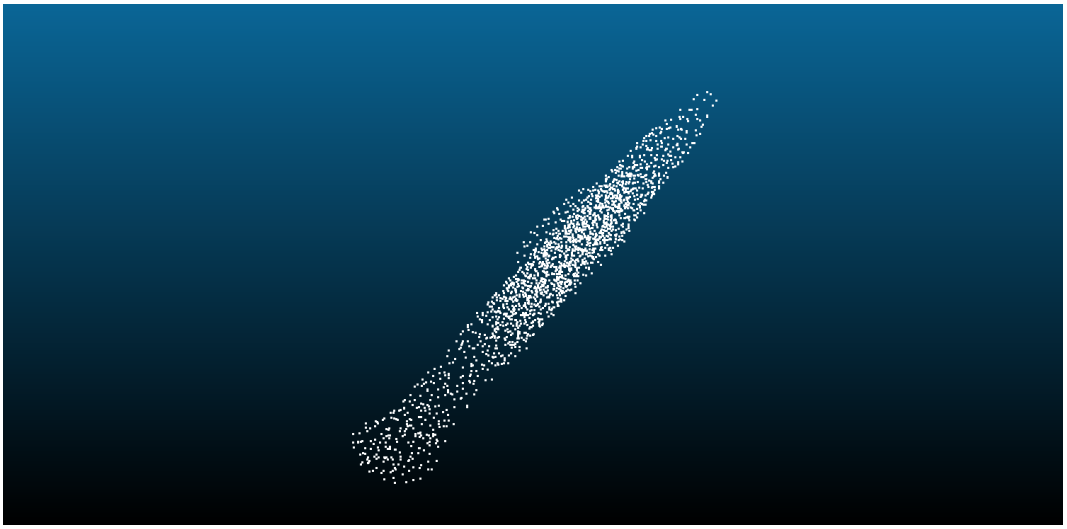

Figura 4.1: Esempio di un razzo generato-1

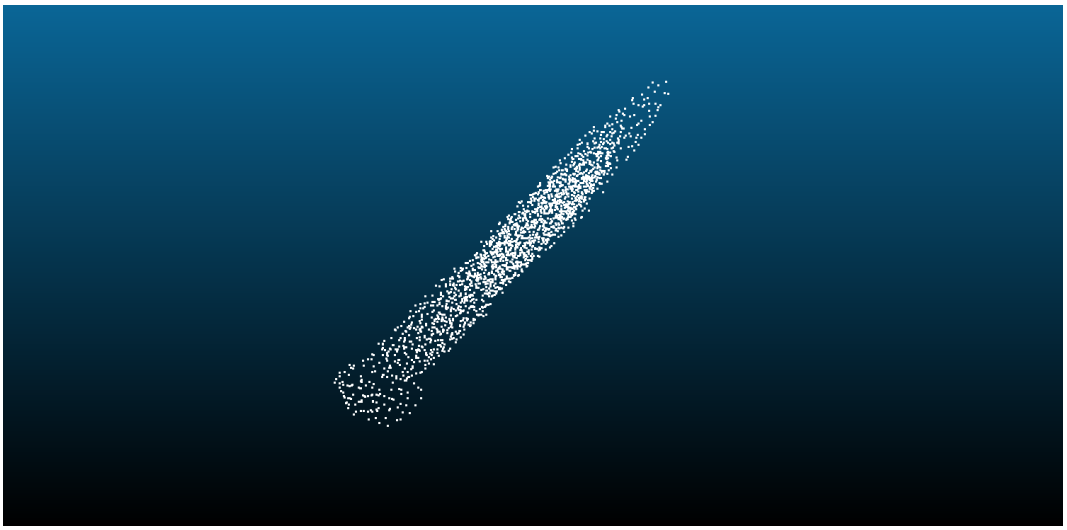

Figura 4.2: Esempio di un razzo generato-2

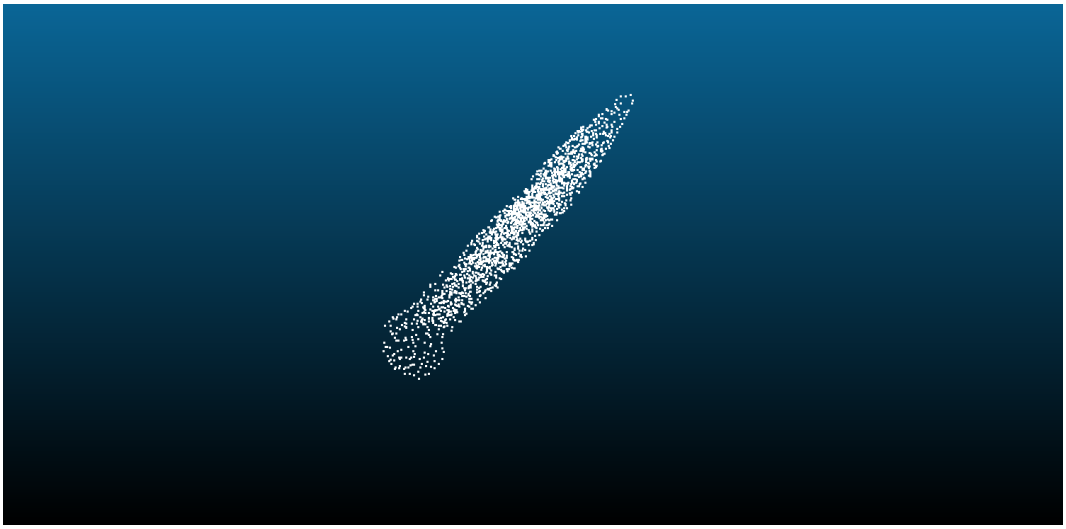

Figura 4.3: Esempio di un razzo generato-3

I risultati delle generazioni sono stati considerati piuttosto buoni come si può vedere dalle immagini 4.1, 4.2 e 4.3: se si volessero ottenere delle nuvole che rappresentino l'oggetto con maggior dettaglio si potrebbe eseguire un training con un numero di epoche maggiore di 1000.

4.2 Risultati della SetVAE

Il processo di data augmentation del dataset *ArCH* è stato quello più impegnativo e che ha richiesto più tempo: dopo aver individuato le classi da aumentare, *stairs* e *roof*, sono stati eseguiti più allenamenti su diverse epoche. Nonostante il più duraturo sia stato eseguito con 8000 epoche gli oggetti generati non sono risultati comunque validi, portando a considerare altre classi per la generazione degli oggetti. Pertanto sono state scelte le classi *arch* e *door-window*, dove per ognuna delle classi sono stati generati 26 nuovi oggetti. Quelli effettivamente aggiunti al dataset però sono stati 10 per la classe *arch* e 16 per la classe *door-window*. Vengono proposti i risultati degli allenamenti per la classe *arch* su diverse epoche così da proporre un confronto visivo degli oggetti generati. Il primo allenamento è stato portato fino a 6000 epoche con una loss che subiva un'oscillazione con valori compresi tra 30 e 40. Gli oggetti prodotti sono stati 4.4, 4.5.

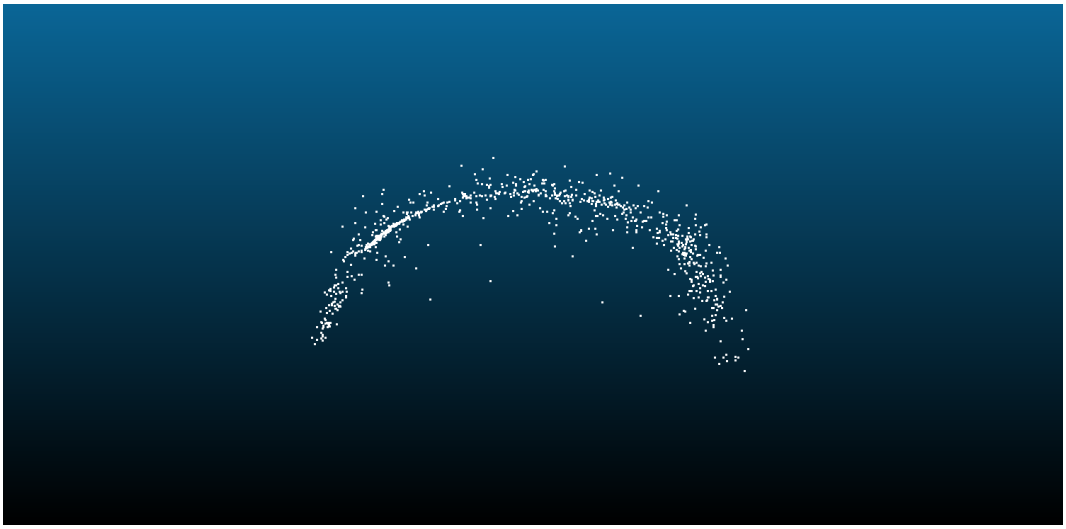

Figura 4.4: Arco generato con 6000 epoche

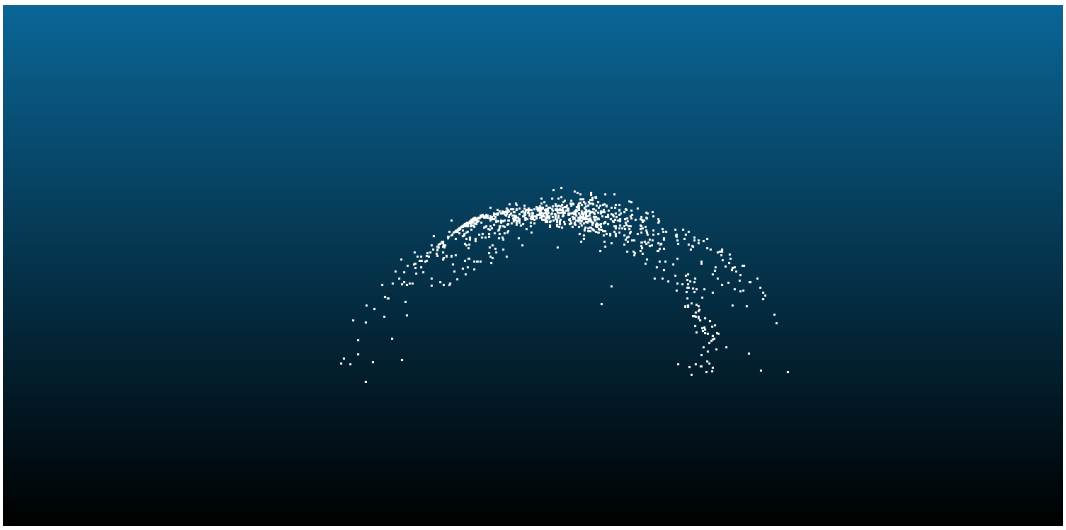

Figura 4.5: Arco generato con 6000 epoche

Come secondo allenamento le epoche sono state aumentate fin a 8000, dove l'oscillazione della loss si è attenuata con valori tra 14 e 20, ottenendo dei leggeri miglioramenti, come si può notare dalle figure 4.6, 4.7.

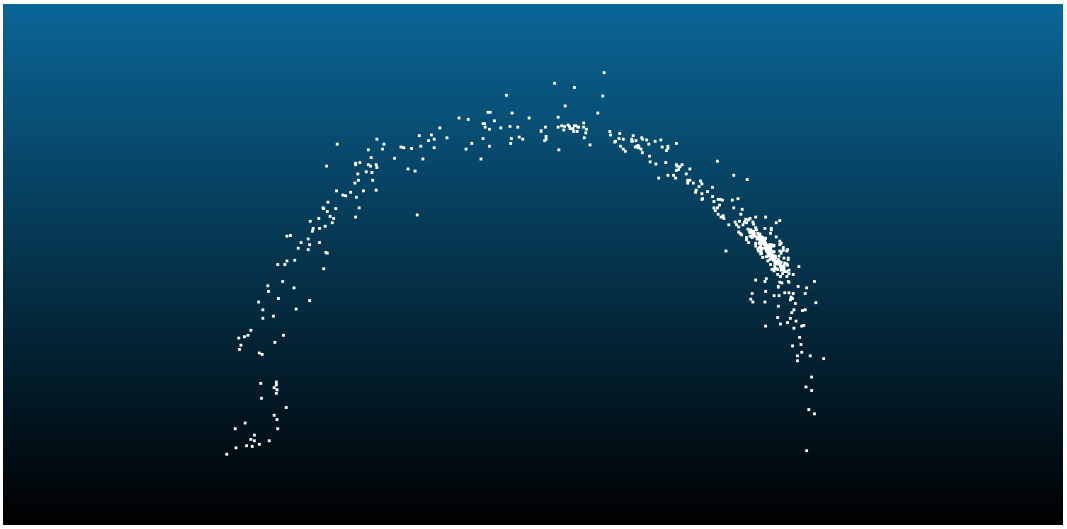

Figura 4.6: Arco generato con 8000 epoche

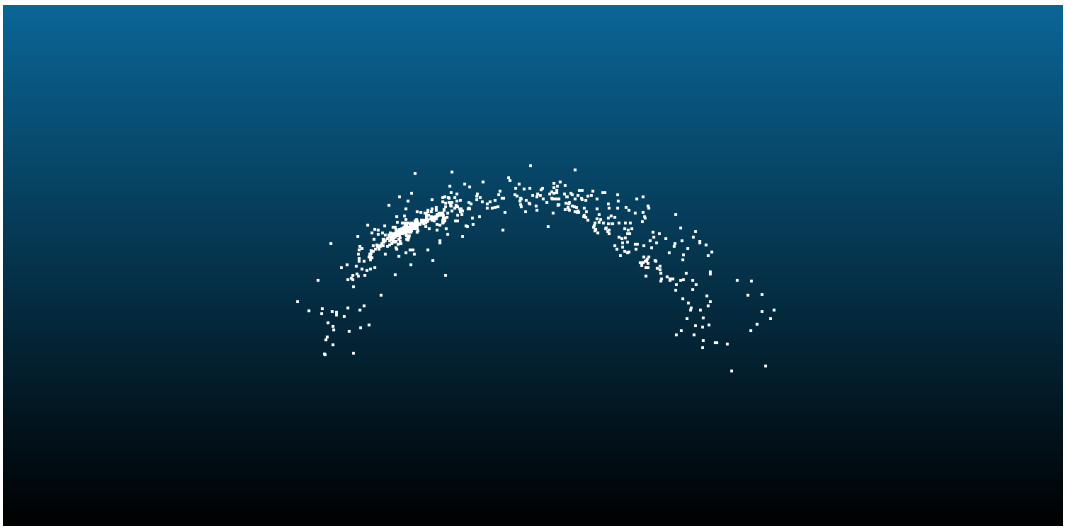

Figura 4.7: Arco generato con 8000 epoche

L'ultimo allenamento è stato di 11000 epoche, in cui l'oscillazione della loss si è ridotta fino a dei valori vicini al 12. I risultati ottenuti sono stati come quelli nelle figure 4.8 e 4.9

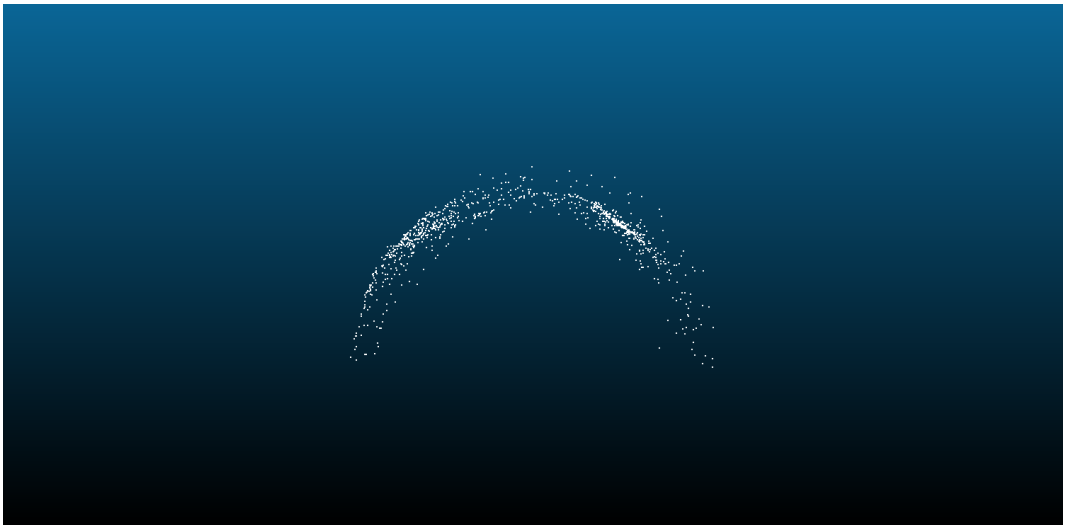

Figura 4.8: Arco generato con 11000 epoche

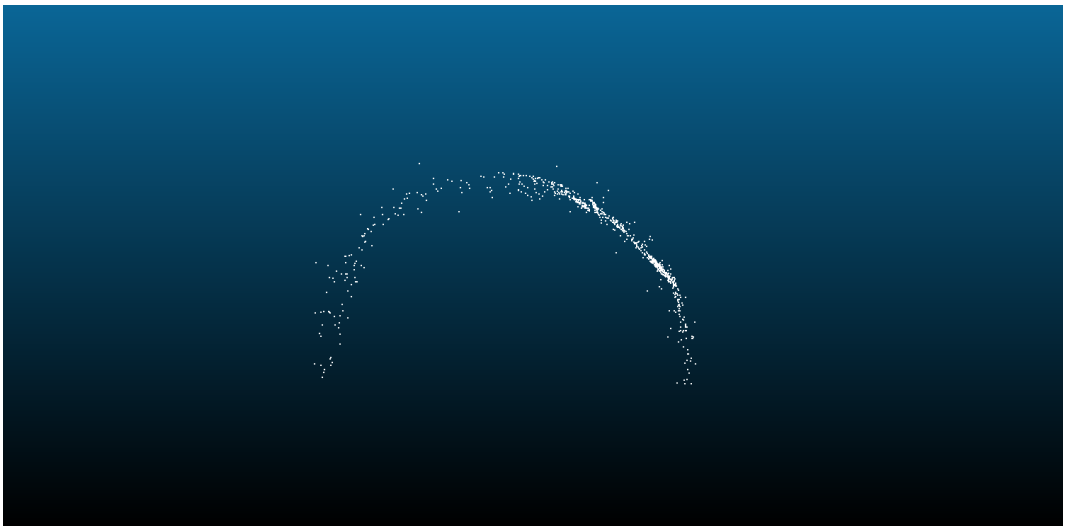

Figura 4.9: Arco generato con 11000 epoche

Non sono stati effettuati allenamenti successivi per una questione di tempistiche, visto che quello da 11000 epoche aveva impiegato circa 11 ore. Inoltre non c'è l'assoluta certezza che con un numero maggiore di epoche la loss si abbassi ulteriormente.

Per quanto riguarda invece la generazione degli oggetti della classe *door-window*, è stato effettuato un unico allenamento di 4000 epoche: in media, per completare 1000 epoche, ha richiesto il doppio del tempo rispetto a quello degli archi.

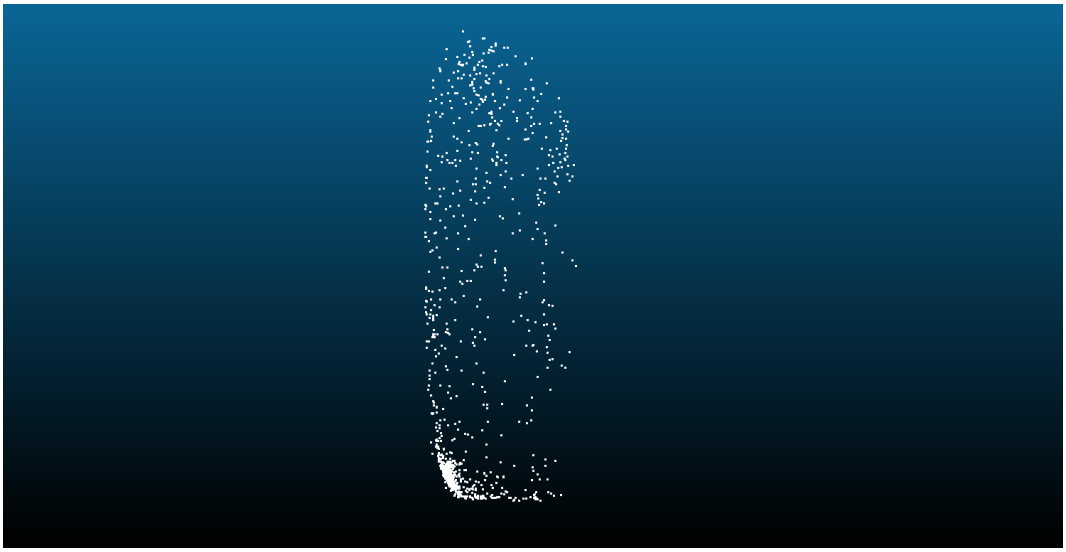

Figura 4.10: Finestra generata con 400 epoche-1

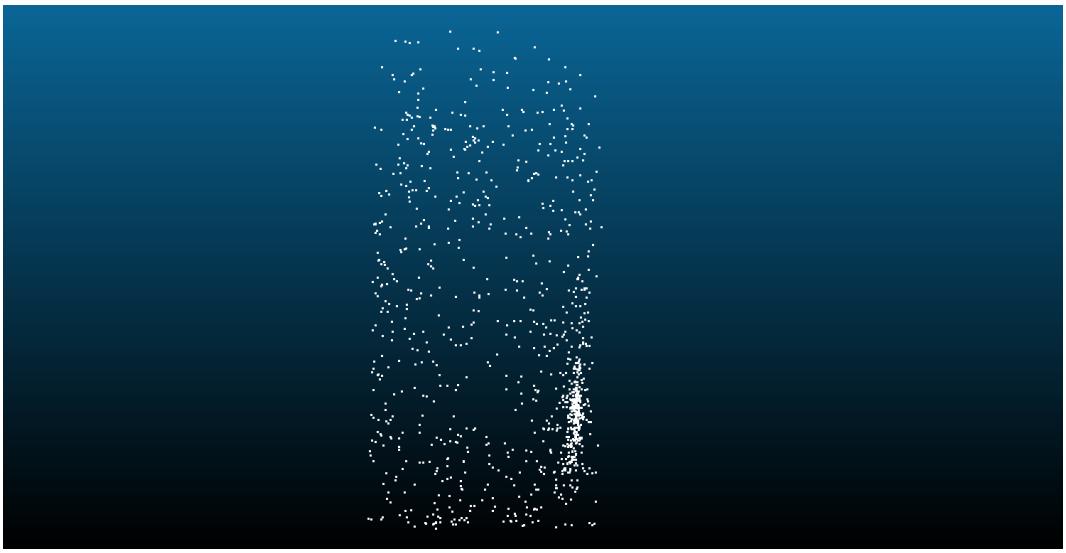

Figura 4.11: Finestra generata con 400 epoche-2

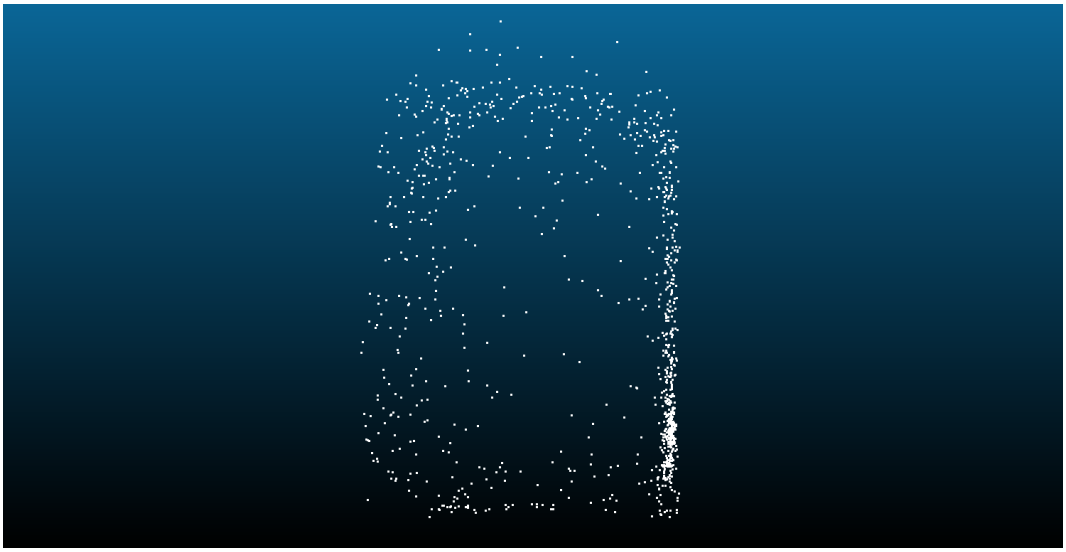

Figura 4.12: Finestra generate con 4000 epoche-3

Come si può vedere dalle figure 4.10, 4.11, 4.12 questi oggetti non sono definiti eccessivamente bene: la loss al termine delle 4000 epoche oscillava intorno ai 40, un valore piuttosto alto che compromette la corretta generazione.

Una precisazione doverosa è che queste nuvole sono state generate con 1024 punti poichè la rete, nel momento del training su Colab, non aveva abbastanza memoria per permettere un allenamento con 2048 punti. Questo è un altro fattore da considerare perchè, con molto probabilità, con il doppio dei punti a disposizione si sarebbero potuti ottenere degli oggetti più verosimili.

4.3 Classificazione tramite la PointNet

Utilizzando la *PointNet* sono state effettuate le classificazioni dei due dataset su cui si è svolto il lavoro, sia prima che dopo averli aumentati con gli oggetti generati.

4.3.1 Classificazione ShapeNet

La prima classificazione è stata fatta sul dataset proposto dal paper per vedere il valore delle metriche e avere un'idea di quali classi aumentare. I risultati prodotti sono i seguenti, tabella 4.1:

Tabella 4.1: Classificazione dello ShapeNetCore

	Precision	**Recall**	**F1-score**	**Support**
Airplane	0.9881	0.9765	0.9823	341
Bag	1.0000	0.9286	0.9630	14
Cap	0.8333	0.4545	0.5882	11
Car	0.9503	0.9684	0.9592	158
Chair	0.9886	0.9872	0.9879	704
Earphone	1.0000	0.7857	0.8800	14
Guitar	0.9815	1.0000	0.9907	159
Knife	0.9620	0.9500	0.9560	80
Lamp	0.9640	0.9371	0.9504	286
Laptop	0.9880	0.9880	0.9880	83
Motorbike	0.8644	1.0000	0.9273	51
Mug	0.9231	0.9474	0.9351	38
Pistol	0.9762	0.9318	0.9535	44
Rocket	0.5882	0.8333	0.6897	12
Skateboard	0.9630	0.8387	0.8966	31
Table	0.9802	0.9906	0.9853	848

	Precision	**Recall**	**F1-score**	**Support**
accuracy	/	/	0.9739	2874
macro avg	0.9344	0.9074	0.9146	2874
weighted avg	0.9747	0.9739	0.9737	2874

Il valore piuttosto basso della metrica *f1-score* della classe *rocket* è stato determinante per selezionarla come soggetto del data augmentation. Sono stati poi graficati gli andamenti delle accuracy sia del training che del validation, figure 4.13 e 4.14, per monitorare il comportamento della rete.

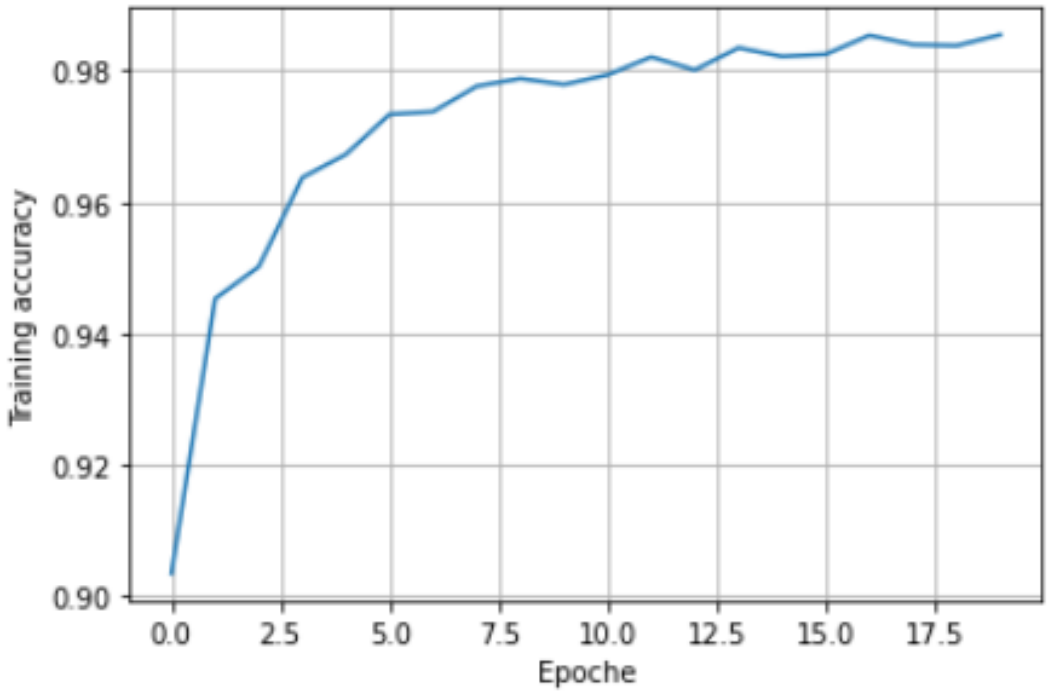

Figura 4.13: Andamento dell'accuracy di training prima di aumentare il dataset

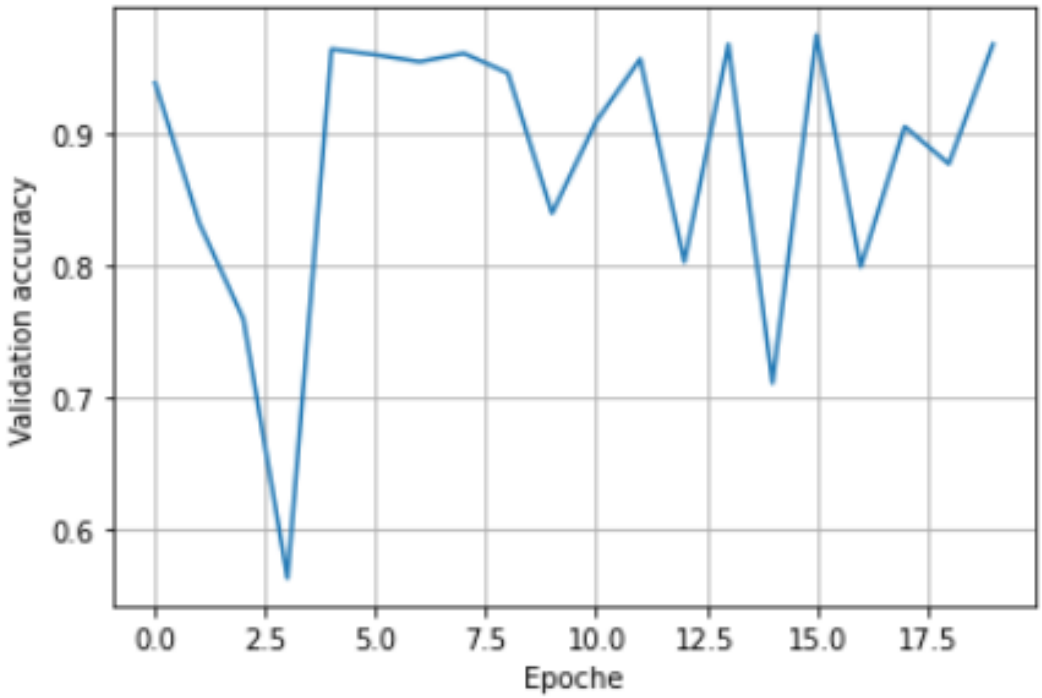

Figura 4.14: Andamento dell'accuracy di validation prima di aumentare il dataset

Dopo aver generato e aggiunto i 10 razzi al dataset, è stato il momento di effettuare la classificazione con il dataset aumentato.

Tabella 4.2: Classificazione ShapeNetCore Aumentato

	precision	**recall**	**f1-score**	**support**
Airplane	0.9978	0.9795	0.9882	341
Bag	1.0000	0.8571	0.9231	14
Cap	0.8333	0.4545	0.5882	11
Car	0.9935	0.9620	0.9775	158
Chair	0.9928	0.9759	0.9842	704
Earphone	0.8667	0.9286	0.8966	14
Guitar	0.9873	0.9811	0.9842	159
Knife	0.9737	0.9250	0.9487	80
Lamp	0.9510	0.9510	0.9510	286
Laptop	0.9880	0.9880	0.9880	83
Motorbike	1.000	0.9608	0.9800	51
Mug	0.9487	0.9737	0.9610	38
Pistol	0.9773	0.9773	0.9773	44
Rocket	0.7333	0.9167	0.8148	12
Skateboard	0.9667	0.9355	0.9508	31
Table	0.9614	0.9988	0.9798	848

	precision	**recall**	**f1-score**	**support**
accuracy	/	/	0.9753	2874
macro avg	0.9482	0.9228	0.9308	2874
weighted avg	0.9757	0.9753	0.9750	2874

Il valore del *f1-score* è migliorato, come si può vedere dalla tabella (4.2, ottenendo il risultato sperato. Come fatto in precedenza sono stati rappresentati gli andamenti delle accuracy, figure 4.15, 4.16 e poi confrontati nello stesso grafico.

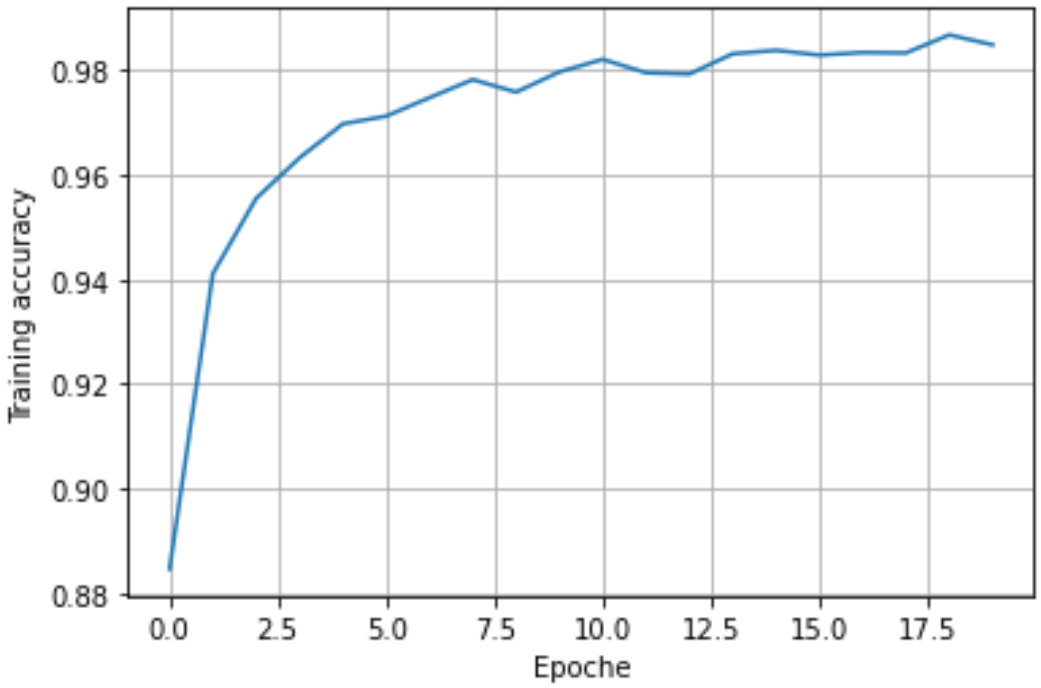

Figura 4.15: Andamento dell'accuracy di training dopo aver aumentato il dataset

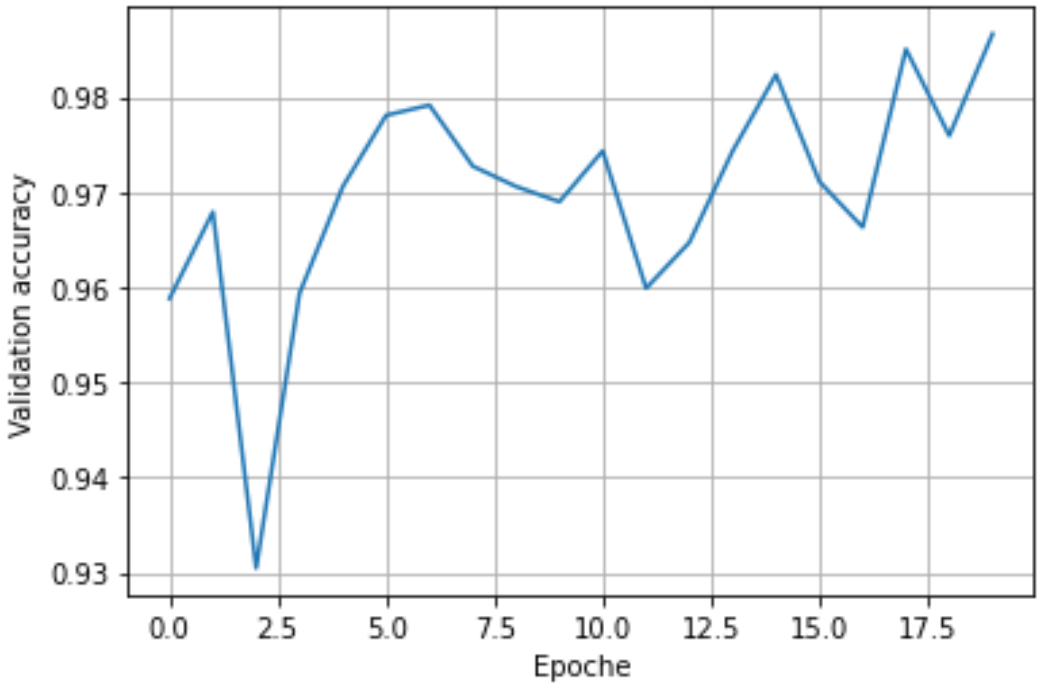

Figura 4.16: Andamento dell'accuracy di validation dopo aver aumentato il dataset

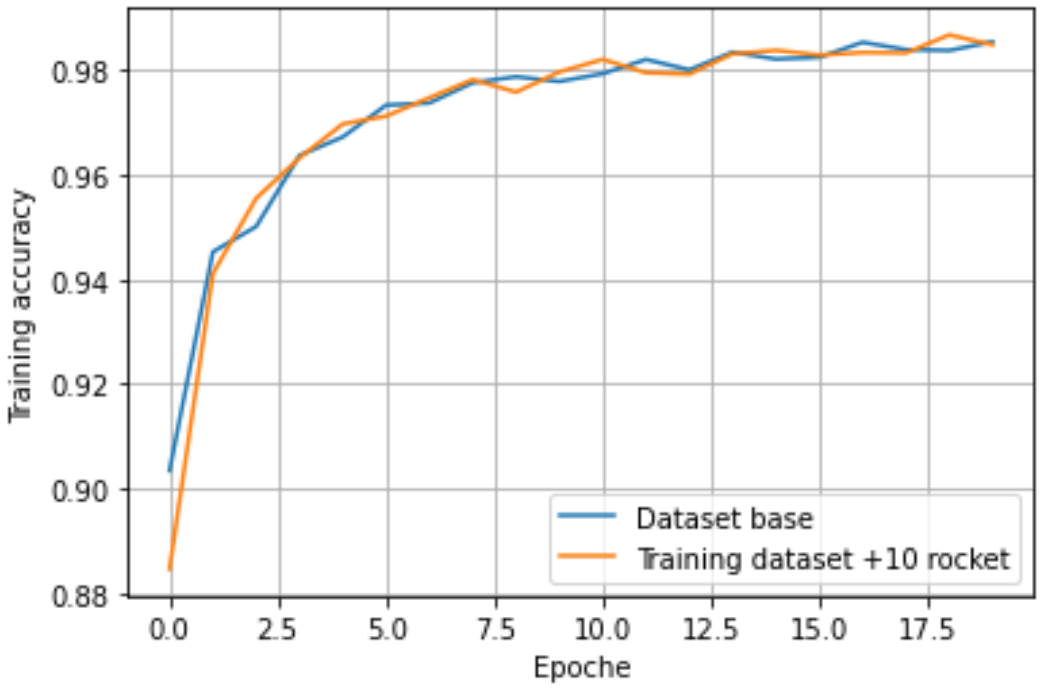

Figura 4.17: Confronto tra gli andamenti dell'accuracy di training pre e post data augmentation

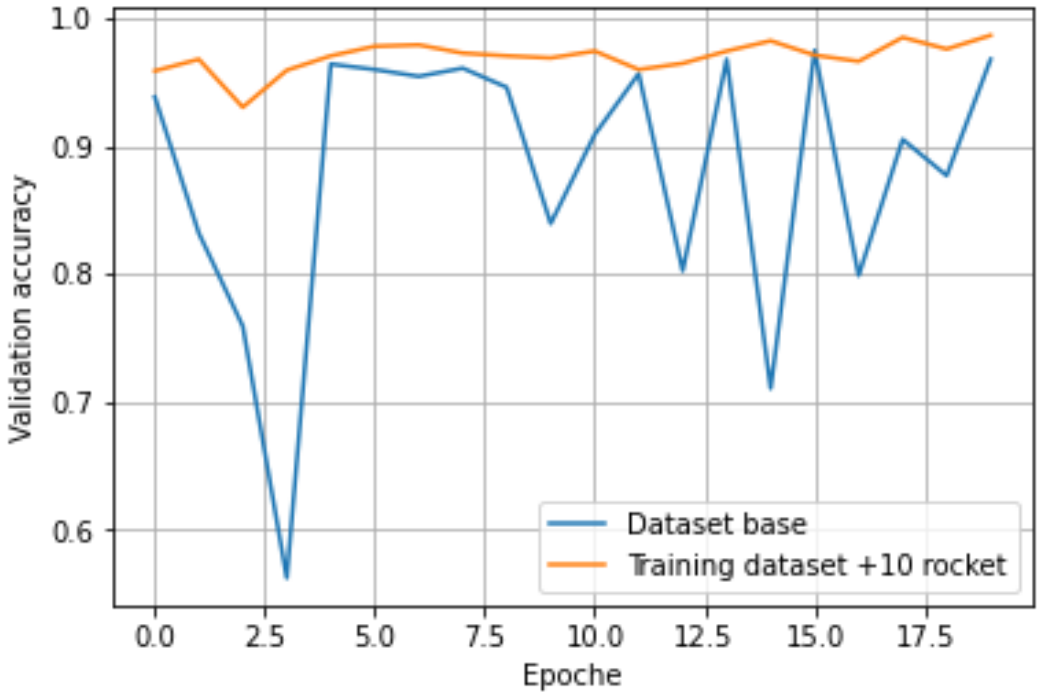

Figura 4.18: Confronto tra gli andamenti dell'accuracy di validation pre e post data augmentation

Come si può vedere dalle figure 4.17 e 4.18 con l'aggiunta di soli 10 nuovi oggetti l'andamento dell'accuracy della validation è risultato essere molto più lineare mostrando meno variazioni rispetto all'andamento del dataset standard. Confrontando le metriche e osservando come ci sia stato un significativo miglioramento del comportamento della rete dopo l'aggiunta di 10 nuovi oggetti, si è ritenuto che questo task fosse stato completato con successo.

4.4 Classificazione ArCH

Per utilizzare questo dataset con la *PointNet* sono state necessarie delle modifiche sia per quanto riguarda la struttura con cui era stato salvato, sia per quanto riguarda la rappresentazione stessa delle nuvole. Inizialmente, nella fase di etichettatura, il dataset era stato salvato con una struttura del tipo scena $\mapsto$ classi $\mapsto$ oggetti: la *PointNet* richiedeva un salvataggio del tipo classi $\mapsto$ oggetti e la normalizzazione delle nuvole di punti.

Tabella 4.3: Classificazione ArCH normalizzato

	precision	**recall**	**f1-score**	**support**
0-arch	0.000	0.000	0.000	57
1-columns	0.000	0.000	0.000	45
2-moldings	0.2578	0.7459	0.3832	122
3-floor	0.000	0.000	0.000	17
4-doors-windows	0.2277	0.1811	0.2018	127
5-wall	0.1905	0.1818	0.1860	44
6-stairs	0.000	0.000	0.000	13
7-vault	0.000	0.000	0.000	30
8-roof	0.000	0.000	0.000	11
9-others	0.000	0.000	0.000	32

	precision	**recall**	**f1-score**	**support**
accuracy	/	/	0.2450	498
macro avg	0.0676	0.1109	0.0771	498
weighted avg	0.1381	0.2450	0.1618	498

Come si può vedere dalla tabella 4.3 i risultati sono piuttosto bassi: pertanto è stata svolta una seconda classificazione sul dataset senza normalizzarlo per vedere quali fossero i risultati e cercare di capire se ci fossero stati errori nel processo di normalizzazione.

Tabella 4.4: Classificazione ArCH non normalizzato

	precision	**recall**	**f1-score**	**support**
0-arch	0.2245	0.3860	0.2839	57
1-columns	0.000	0.000	0.000	45
2-moldings	0.2571	0.1475	0.1875	122
3-floor	0.0768	0.0588	0.0667	17
4-doors-windows	0.3598	0.6772	0.4699	127
5-wall	0.1053	0.0455	0.0635	44
6-stairs	0.000	0.000	0.000	13
7-vault	0.000	0.000	0.000	30
8-roof	0.000	0.000	0.000	11
9-others	0.0702	0.1250	0.0899	32

	precision	**recall**	**f1-score**	**support**
accuracy	/	/	0.2671	498
macro avg	0.1094	0.1440	0.1161	498
weighted avg	0.1969	0.2671	0.2119	498

Non essendo ancora convincenti i risultati ottenuti, visibili nella tabella 4.4, si è deciso di cambiare il tipo split effettuato, suddividendo quindi il dataset in *80%* training, *10%* testing e *10%* validation, da uno split di partenza del tipo *60%*, *20%*, *20%*. Inoltre, piuttosto che normalizzare rispetto ai 3 assi le singole nuvole, sono state traslate nell'origine, considerando anche il fatto che gli oggetti all'interno della stessa classe potessero essere molto diversi tra loro e quindi con la normalizzazione ci sarebbero state perdite di informazioni. Dopo aver portato a termine un allenamento di 20 epoche i risultati ottenuti sono stati decisamente migliori, come si può vedere nella tabella 4.5.

Tabella 4.5: Classificazione ArCH

	precision	**recall**	**f1-score**	**support**
0-arch	0.6176	0.7241	0.6667	29
1-columns	0.7037	0.8261	0.7600	23
2-moldings	0.5593	0.5410	0.5500	61
3-floor	0.6364	0.7778	0.7000	9
4-doors-windows	0.7794	0.8281	0.8030	64
5-wall	0.5000	0.5000	0.5000	22
6-stairs	0.1667	0.1429	0.1538	7
7-vault	0.5714	0.8000	0.6667	15
8-roof	0.000	0.000	0.000	5
9-others	0.5000	0.1250	0.2000	16

	precision	**recall**	**f1-score**	**support**
accuracy	/	/	0.6310	252
macro avg	0.5035	0.5265	0.5000	252
weighted avg	0.6054	0.6310	0.6085	252

Nella speranza di migliorare ancora di più i risultati è stato fatto un training di 500 epoche, ponendo il *Learning Rate = 0.0001*. Questo parametro indica l'ampiezza del passo che deve compiere il gradiente all'interno dell'algoritmo di ottimizzazione. Inoltre il valore dell'*accuracy* del training ha raggiunto un valore notevole: *0.9721*. I risultati sono illustrati nella tabella 4.6

Tabella 4.6: Classificazione ArCH con un training di 500 epoche

	precision	**recall**	**f1-score**	**support**
0-arch	0.8571	0.8276	0.8421	29
1-columns	0.7308	0.8261	0.7755	23
2-moldings	0.7544	0.7049	0.7288	61
3-floor	0.6364	0.07778	0.7000	9
4-doors-windows	0.8235	0.8750	0.8485	64
5-wall	0.4667	0.3182	0.3784	22
6-stairs	0.2857	0.2857	0.2857	7
7-vault	0.8333	1.0000	0.9091	15
8-roof	0.7500	0.5000	0.6000	6
9-others	0.4444	0.5000	0.4706	16

	precision	**recall**	**f1-score**	**support**
accuracy	/	/	0.7302	252
macro avg	0.6582	0.6615	0.6539	252
weighted avg	0.7242	0.7302	0.7238	252

A questo punto, dopo aver generato ed aggiunto i 26 oggetti delle due classi indicate in precedenza, è stato il momento di effettuare la classificazione sul dataset aumentato.

Tabella 4.7: Classificazione ArCH aumentato

	precision	**recall**	**f1-score**	**support**
0-arch	0.8696	0.6667	0.0.7547	30
1-columns	0.6667	0.9565	0.7857	23
2-moldings	0.5690	0.5410	0.5546	61
3-floor	0.5714	0.4444	0.5000	9
4-doors-windows	0.8000	0.7385	0.7680	65
5-wall	0.3333	0.5909	0.4262	22
6-stairs	0.3333	0.1429	0.2000	7
7-vault	0.8125	0.8667	0.8387	15
8-roof	0.3333	0.1667	0.2222	6
9-others	0.0833	0.0625	0.0714	16

	precision	**recall**	**f1-score**	**support**
accuracy	/	/	0.6142	254
macro avg	0.5362	0.5177	0.5112	254
weighted avg	0.6238	0.6142	0.6094	254

Purtroppo non sono stati ottenuti i risultati sperati poichè invece di migliorare, il valore dell'*f1-score*, come si può vedere nella tabella 4.7 è peggiorato. Anche in questo caso, però, il valore dell'accuracy del training era arrivato a un buon valore: 0.9781. In realtà i risultati ottenuti rispecchiano la non eccellente qualità delle nuvole generate: con oggetti molto più definiti da quelli qui ottenuti si potrebbero ottenere dei risultati di classificazione migliori. Inoltre il limite riscontrato durante l'utilizzo della *SetVAE*, in cui non è stato possibile lavorare con nuvole di 2048 punti, ma da 1024, quasi sicuramente ha contribuito ad una generazione degli oggetti poco definita. Un dato interessante però, che ci si aspettava, emerso anche dalle statistiche delle scene del dataset *ArCH* discusse nella sezione 3.3, ha mostrato come le classi con un discreto numero di oggetti ma con un basso numero di punti abbiano dei valori delle metriche di classificazione piuttosto bassi.

Capitolo 5

Conclusioni e lavori futuri

5.1 Discussione

I risultati ottenuti hanno evidenziato il problema principale dell'ambito dei beni culturali, quale la mancanza di dati su cui lavorare. Infatti attraverso i risultati della classificazione e delle generazione sono emersi i limiti dovuti al numero ristretto di oggetti delle classi, con il quale non si sono potuti produrre risultati di alta qualità. Tuttavia, essere riusciti a far funzionare queste reti con il dataset *ArCH* rappresenta un buon punto di partenza per lavori futuri, poichè si dispone già di qualcosa funzionante a cui dovranno essere aggiunti ulteriori dati. Per quanto riguarda invece i risultati delle reti ottenuti con lo *ShapeNet*, sono piuttosto buoni, visti anche i numerosi studi svolti su di esso, mostrando nuovamente come l'ampia disponibilità di dati sia un requisito fondamentale per ottenere dei risultati importanti.

5.2 Sviluppi futuri

Il primo e forse il più importante sviluppo che si dovrebbe fare è l'aumento della mole di dati su cui lavorare, con l'idea di offrire un dataset più ampio, o migliorare quello già presente con nuove scene. Questo potrebbe garantire delle metriche di classificazione più alte e una generazione degli oggetti più precisa: la generazione, soprattutto in quest'ambito, rappresenta il task più difficile e più importante. Sarebbe utile anche verificare che l'etichettatura del dataset *ArCH* non contenga errori: in uno dei tentativi di data augmentation è emerso come alcuni oggetti fossero classificati erroneamente. Se ce ne fosse l'opportunità, potrebbe essere interessante utilizzare tutti i 2048 per generare le nuvole di punti, confrontandoli con quelli qui presenti, poichè, per limitazioni tecniche le nuvole generate con la *SetVAE* sono composte da 1024 punti. Inoltre, ulteriori sviluppi possono essere i task di *object detection* e *segmentazione semantica*, completando così la lista dei task comunemente effettuati nell'ambito del deep learning.

Bibliografia

[1] Shi Dong, Ping Wang, and Khushnood Abbas. A survey on deep learning and its applications. *Computer Science Review*, 40:100379, 2021.

[2] Matrone, F., Lingua, A., Pierdicca, R., Malinverni, E. S., Paolanti, M., Grilli, E., Remondino, F., Murtiyoso, A. A BENCHMARK FOR LARGE-SCALE HERITAGE POINT CLOUD SEMANTIC SEGMENTATION. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, 2020.

[3] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[4] Jinwoo Kim, Jaehoon Yoo, Juho Lee, and Seunghoon Hong. Setvae: Learning hierarchical composition for generative modeling of set-structured data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15059–15068, 2021.

[5] 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, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[6] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[7] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387, 2014.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[9] Sebastian Raschka and Vahid Mirjalili. Python machine learning: Machine learning and deep learning with python. *Scikit-Learn, and TensorFlow. Second edition ed*, 2017.

[10] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[11] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[12] Anh Viet Phan, Minh Le Nguyen, Yen Lam Hoang Nguyen, and Lam Thu Bui. Dgcnn: A convolutional neural network over large-scale labeled graphs. *Neural Networks*, 108:533–543, 2018.

[13] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[14] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3859–3868, 2019.

[15] Sameera Ramasinghe, Salman Khan, Nick Barnes, and Stephen Gould. Spectral-gans for high-resolution 3d point-cloud generation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8169–8176. IEEE, 2020.

[16] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[17] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[18] Iro Armeni, O. Sener, A. Zamir, Helen Jiang, I. Brilakis, Martin Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1534–1543, 2016.

[19] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):1–12, 2016.

[20] Sito PyTorch. `https://pytorch.org/`.

[21] Sito Tensorflow. `https://www.tensorflow.org/`.

[22] Sito Keras. `https://keras.io/`.