



UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA TRIENNALE IN INFORMATICA

Raciti Riccardo

iPlan

RELAZIONE MACHINE LEARNING

Prof. Giovanni Maria Farinella
Prof. Francesco Ragusa

Anno Accademico 2023 - 2024

Abstract

Il presente lavoro esamina e sperimenta il paper iPLAN, un progetto sviluppato da ricercatori cinesi con l'obiettivo di creare un'architettura per l'identificazione delle tipologie di stanze in una planimetria, la segmentazione della planimetria e l'inserimento di porte nei muri che suddivideranno lo spazio. Un elemento chiave del progetto è la possibilità offerta ai progettisti di intervenire in qualsiasi fase della generazione del design, garantendo così un'architettura estremamente versatile e di massimo supporto per i professionisti del settore. I risultati ottenuti sono confrontati con altre architetture esistenti, come Rplan o Graph2Plan. L'architettura proposta offre tre versioni di generazione che differiscono per il numero di input utilizzati. Saranno analizzati gli input e le differenze nelle prestazioni tra le diverse versioni, con un confronto anche rispetto alle architetture precedentemente menzionate.

Indice

1 Introduzione	4
2 Lavori Correlati	7
2.1 Sintesi della scena interna	7
2.1.1 Fast and Flexible Indoor Scene Synthesis	8
2.1.2 Deep Convolutional Priors for Indoor Scene Synthesis	10
2.1.3 PlanIT: Planning and Instantiating Indoor Scenes with Relation Graph and Spatial Prior Networks	11
2.2 Generazione della planimetria	12
2.3 Composizione dell'immagine	12
2.3.1 ArchiGAN: Artificial Intelligence x Architecture	13
3 Metodologia	14
3.1 Formulazione del problema	14
3.1.1 Connessione ad altri paper	14
3.2 Numero e tipo di stanze	14
3.3 Individuazione delle stanze	14
3.3.1 Atrous Spatial Pyramid Pooling (ASPP)	14
3.4 Predizione del partizionamento della stanza	15
4 Esperimenti	18
4.1 Dataset	18
4.2 Metriche	18
4.3 Baselines	19
4.4 Valutazioni Quantitative	19
4.5 Valutazione Qualitativa	22
4.6 Interazioni con L'Utente	23
4.7 Generalizzabilità	24
5 Specifiche	26
5.1 Architettura di BCVAE	26

5.2	Postelaborazione per predizione del partizionamento della stanza	27
5.3	Ulteriori confronti qualitativi	28
5.4	Ulteriori valutazioni di generalizzazione	30
5.5	Dettagli sull'implementazione	30
5.6	Limitazioni	30
6	Sperimentazioni con Validation Set	32
6.1	Room Location	33
6.1.1	Living	34
6.1.2	Location	39
6.2	Room Partition	46
6.3	Room Type	48
7	Prove Sui Modelli	52
7.1	RoomType	52
7.1.1	Valutazione Risultati	55
7.2	Locating Room	56
7.2.1	Valutazione Risultati	58
7.2.2	Ulteriori funzioni	60
7.3	Room Partition	61
7.3.1	Modifiche	61
7.3.1.1	Parametri Input	62
7.3.1.2	Istanziazione Modello	63
7.3.1.3	Caricamento File Pickle	64
7.3.1.4	Modifica GetList	65
7.3.1.5	Generazione Dataset per il modello	67
7.3.1.6	Modifica update rBoxes	68
7.3.2	Valutazione Risultati	69
7.3.3	Ulteriori funzioni	73
8	Esecuzione Intera Pipeline	74
8.0.0.1	Valutazione Boxes	78
8.0.1	Ulteriori Sperimentazioni	83
Conclusione		86
Bibliografia		88

Capitolo 1

Introduzione

iPLAN si distingue come un modello in grado di generare automaticamente layout, ma va oltre, permettendo una significativa interazione con i progettisti durante l'intero processo. Questa caratteristica consente un'associazione sinergica tra l'intelligenza artificiale e gli esseri umani per plasmare gradualmente un concetto abbozzato fino a ottenere il progetto finale.

Il termine "layout" è qui utilizzato per indicare la pianta di una planimetria. Nella progettazione tradizionale, il processo segue un iter di regolazione e finalizzazione iterativa, passando dai dettagli grezzi a quelli fini e dai concetti globali a quelli locali. Questa metodologia impone routine complesse e ripetitive per i progettisti.

Recentemente, è emerso che la generazione automatica di immagini progettuali, con un coinvolgimento umano minimo, è possibile grazie all'apprendimento dei dati. Questa nuova direzione di ricerca fonde il deep learning con il design, aprendo una via innovativa per la progettazione assistita dall'IA.

Sebbene la generazione completamente automatica sia cruciale, il processo di design è intrinsecamente procedurale, coinvolgendo iterazioni tra routine ripetitive e pensiero creativo nelle fasi intermedie. Un sistema ideale di supporto AI dovrebbe automatizzare le parti routine, consentendo al progettista di preservare la propria creatività. Questo implica che il sistema sia in grado di interagire attivamente con il progettista, accettando la guida umana e suggerendo soluzioni in modo proattivo, completando un ciclo di feedback continuo.

Dataset

I datset utilizzati per questo modello sono RPLAN[1] e LIFULL[2].

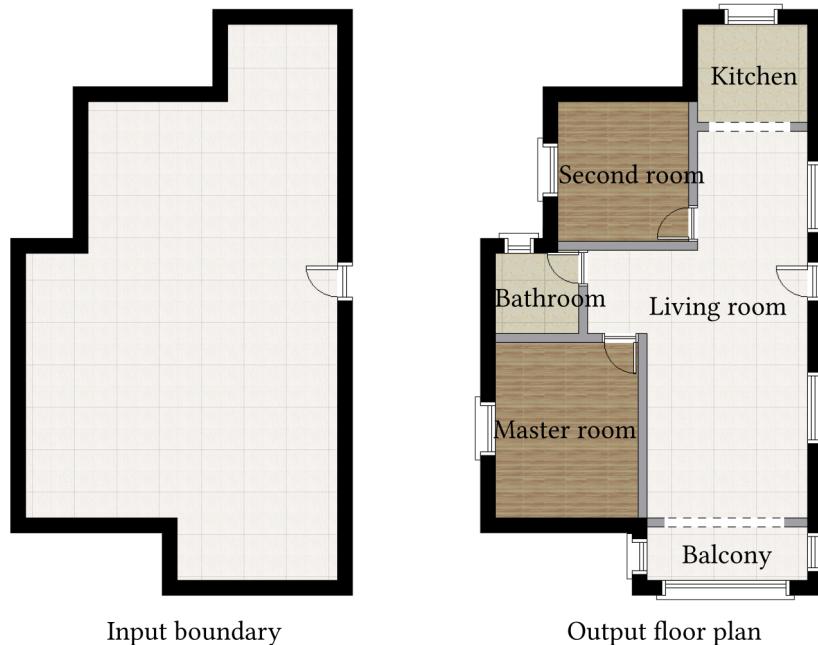


Figura 1.1: Esempio dataset.

RPLAN e LIFULL solitamente includono solo gli snack finali e non i risultati intermedi, i quali sono chiamati nell'immagine soprastante *Input boundary* e *Output floor plan*.

Per portare un esempio più preciso e soffermandoci su RPLAN, il seguente dataset contiene 80787 layout finali e la loro forma è del seguente tipo.

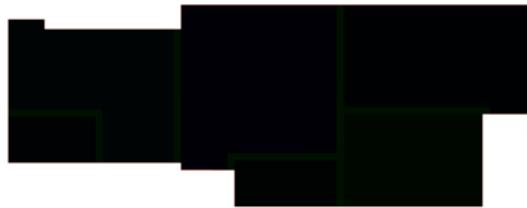


Figura 1.2: Esempio dataset RPLAN.

Una possibile strategia per affrontare questa sfida consiste nel praticare il *reverse-engineer* delle fasi intermedie dai progetti finali. Tuttavia, questa soluzione introduce a sua volta una complessità aggiuntiva: l'ordine delle fasi può variare notevolmente anche per lo stesso progetto finale, a seconda del compito o dell'obiettivo specifico. Inoltre, l'incertezza sull'ordine è influenzata dalle forti preferenze e stili personali dei designer.

Per colmare la lacuna di dati relativi alle fasi procedurali di progettazione, gli autori del paper hanno adottato un approccio di decodifica del progetto finale, identificando il processo da uno stato a un altro. Questa metodologia si basa su principi ampiamente accettati dai professionisti del settore, e nella relazione, quando ci si riferirà ai professionisti, si farà riferimento a figure quali architetti di tipo citato in [3].

Quest'approccio ha permesso di sviluppare catene di Markov in grado di catturare l'intera procedura di progettazione. Dato che esistono molteplici modi per decodificare gli indicatori finali (vale a dire, variazioni nell'ordine delle fasi), il modello è progettato con la flessibilità di accettare input con ordini arbitrari. Questa caratteristica consente al modello di apprendere le variazioni di stile implicite nei dati, contribuendo così a superare le sfide legate alla natura mutevole e soggettiva del processo di progettazione.

Capitolo 2

Lavori Correlati

La generazione del layout è stata un'area di ricerca attiva nel computer vision, vi sono differenti casi studio:

- *Sintesi della scena interna.*
- *Generazione della planimetria.*
- *Composizione dell'immagine.*

Gli approcci esistenti possono essere generalmente raggruppati in due categorie: **metodi artigianali**, o fatti a mano, *basati su regole* e **metodi basati sui dati**. Verranno esaminati solamente quelli della seconda categoria in quanto correllati al modello.

2.1 Sintesi della scena interna

Comporta tipicamente il posizionamento di modelli di mobili da un database esistente in una determinata stanza. Le reti neurali convoluzionali possono essere addestrate a inserire iterativamente un oggetto alla volta in una stanza per la generazione di scene interne [4, 5].

È anche possibile impiegare una semantica di scena di alto livello, ad esempio, grafici di scena apriori per una generazione più controllata [6].

La più grande differenza tra la sintesi della scena interna e la generazione della planimetria sono i loro *requisiti sul partizionamento dello spazio*. Mentre la sintesi della scena interna posiziona oggetti in una stanza in cui la stanza stessa non ha bisogno di essere divisa, la planimetria richiede normalmente una divisione esplicita dello spazio per diverse funzionalità.

2.1.1 Fast and Flexible Indoor Scene Synthesis

Questo modello come descritto precedentemente [4] è un modello basato sulle Convolutional Neuronal Network CNN, il quale scopo consiste nell'aggiungere in un layout, vuoto o parzialmente riempito, degli oggetti uno alla volta. Di seguito viene riportata la pipeline del modello:

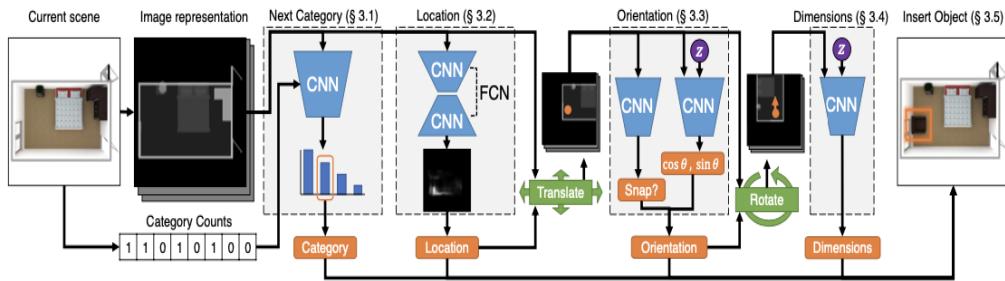


Figura 2.1: Pipeline fast and flexible indoor scene synthesis via deep convolutional generative models.

Questo modello per scegliere che oggetto andare ad inserire all'interno della stanza calcola delle probabilità in base allo spazio disponibile nella stanza e dando priorità agli oggetti più grandi, questo si traduce che gli oggetti più grandi qual'ora entrassero nel layout verranno considerati con una probabilità più alta.

Ecco un esempio di probabilità generata dal modello.

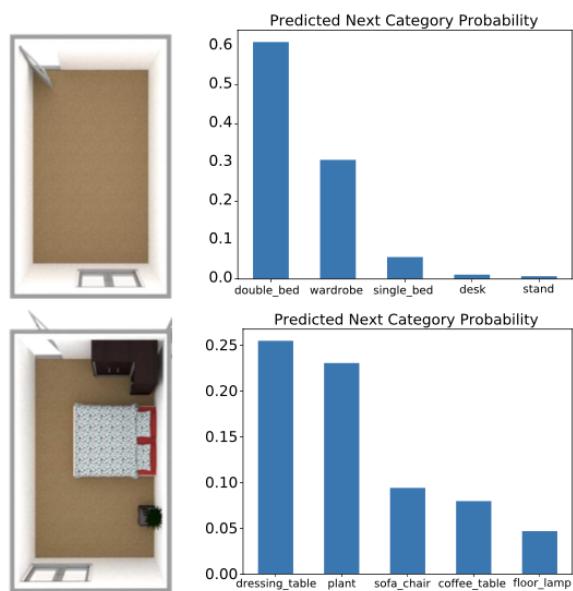


Figura 2.2: Esempio probabilità.

2.1.2 Deep Convolutional Priors for Indoor Scene Synthesis

Un’altro esempio di modello generativo basato su CNN [5] è il seguente.

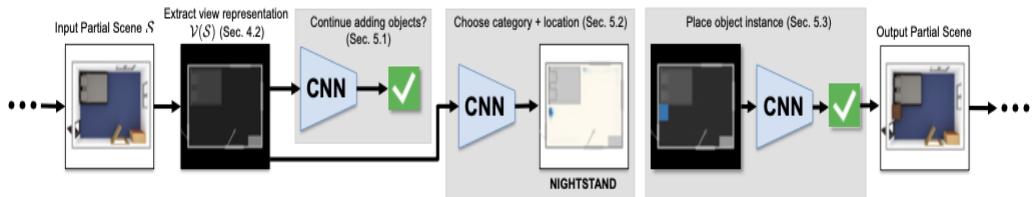


Figura 2.3: Pipeline Deep convolutional priors for indoor scene synthesis.

In questo modello le probabilità vengono generate attraverso la probabilità di Bernoulli. La probabilità di ottenere un successo in un esperimento di Bernoulli è spesso denotata come p , e può essere espressa come:

$$P(X = 1) = p$$

Dove $P(X = 1)$ rappresenta la probabilità di successo, e p è la probabilità stessa. Inoltre, la probabilità di insuccesso può essere calcolata come:

$$P(X = 0) = 1 - p$$

2.1.3 PlanIT: Planning and Instantiating Indoor Scenes with Relation Graph and Spatial Prior Networks

Nel seguente modello [6] si comincia a vedere già qualcosa che ha a che fare con l'idea applicato per il paper iPlan.

Questo modello prende in input una stanza, con arredamenti o non, poi ne calcola un **grafo di relazione** mettendo in relazione la stanza, intesa come muri e spazi disponibili, e calcola tramite il grafo la possibilità di inserire quell'item nella stanza o meno.

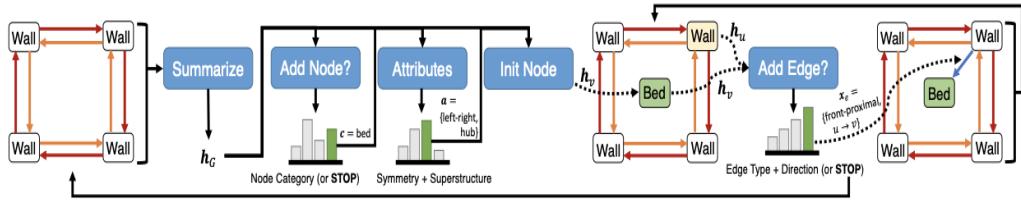


Figura 2.4: Grafo di relazione

In questo contesto la pipeline dell'architettura è un pò differente.

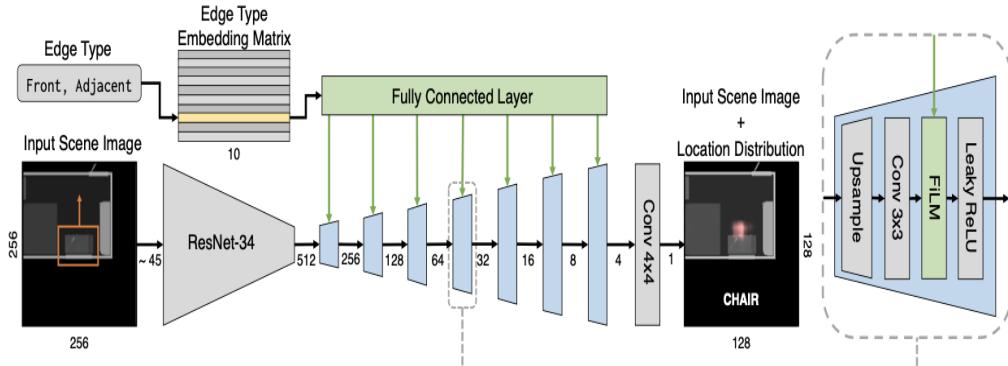


Figura 2.5: Pipeline

Come si può notare si prende il layout in input e si classificano gli oggetti tramite il classificatore CNN **ResNet-34**, per poi scegliere il miglior item che si abbina all'ambiente.

2.2 Generazione della planimetria

Un ambito correlato è rappresentato dalla composizione d’immagini dai grafici di scena, in cui l’obiettivo è estrarre la scena da un grafico di layout che specifica posizioni e caratteristiche degli oggetti.

In questo contesto, la generazione può essere realizzata tramite Generative Adversarial Networks (GAN) basate sulla convoluzione del grafo, come illustrato da Johnson et al. [7]. Ulteriori avanzamenti possono essere ottenuti separando il layout dall’aspetto degli oggetti, un approccio esplorato da Ashual et al. [8].

Per ottenere una maggiore controllabilità, Li et al. [9] propongono la sintesi di immagini da un grafico di scena e i relativi ritagli di immagini. A differenza della generazione di planimetrie, la sfida nella composizione d’immagini risiede nel comporre oggetti diversi all’interno di un’immagine, piuttosto che partizionare lo spazio. Questo sottolinea la diversità di approcci richiesta per affrontare le sfide specifiche di ciascun contesto.

2.3 Composizione dell’immagine

La generazione di planimetrie può essere concepita come un problema di sintesi delle immagini, rientrando così in un’area di ricerca attiva nel campo della visione artificiale. Grazie alla diffusione dell’apprendimento profondo, gli approcci più promettenti sono emersi attraverso i Generative Adversarial Networks (GAN) [10, 11, 12, 13, 14, 15, 16].

I GAN basati su immagini hanno dimostrato una notevole efficacia nella generazione di planimetrie [17, 18, 19, 20, 21, 22, 23]. In aggiunta, i GAN basati su grafici possono generare planimetrie considerando esclusivamente i vincoli spaziali, come le connessioni tra le stanze e i tipi di stanze, espressi sotto forma di grafo [24, 25].

Tuttavia, è rilevante notare che tutti questi metodi seguono un approccio end-to-end, limitando quindi l’interattività del progettista. Questo vincolo sottolinea l’importanza di sviluppare approcci che possano offrire una maggiore interazione durante il processo di generazione delle planimetrie.

2.3.1 ArchiGAN: Artificial Intelligence x Architecture

Nel seguente paper [17] non è presente la pipeline del modello, però ci permette di vedere i risultati ottenibili dai modelli generativi. Nel primo esempio mostrato possiamo vedere come da uno sketch vuoto il modello crea un layout e lo partiziona dividendolo in stanze , **stesso obiettivo di iPlan**, per poi aggiungere l'arredamento.

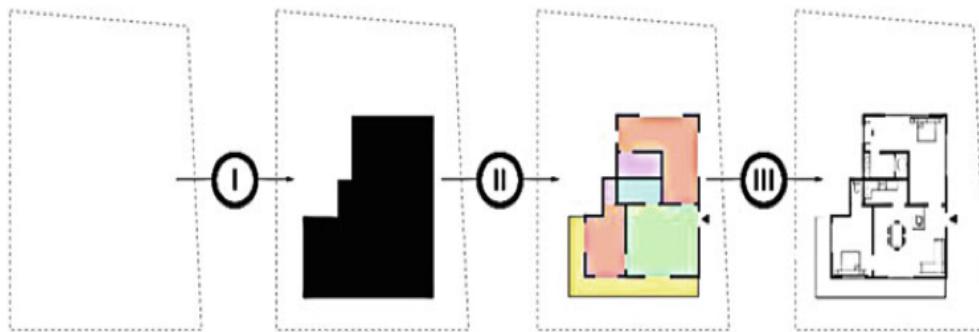


Figura 2.6: Esempio input output.

In questi altri esempi invece ci vengono mostrati gli input forniti al modello, la generazione del modello e il ground Truth.

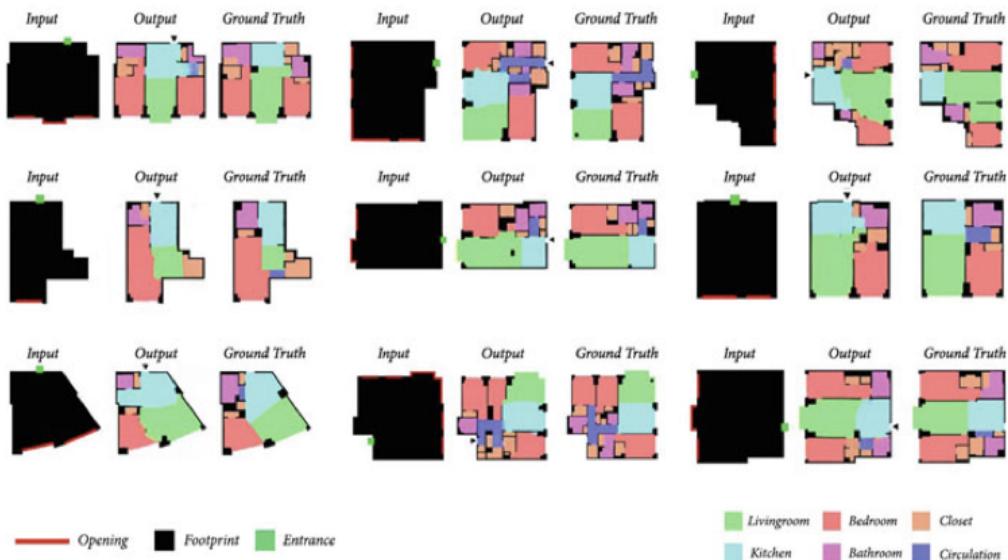


Figura 2.7: Esempio input output.

Capitolo 3

Metodologia

In questo capitolo, esploreremo come il modello **iPlan** affronta il compito di generare layout architettonici. Il modello riceve come input il contorno dello spazio e suddivide la procedura di progettazione in tre fasi fondamentali: l'identificazione dei tipi di stanze, la localizzazione delle stanze e la definizione finale delle partizioni delle stanze.

Il flusso di lavoro di iPlan è progettato per emulare il processo seguito dai progettisti umani, in grado di accogliere gli input dei progettisti in qualsiasi fase del processo di progettazione. Questo approccio mira a massimizzare la flessibilità e l'interattività nel processo creativo.

La rappresentazione del flusso di lavoro assume la forma di una distribuzione di probabilità congiunta che incorpora tutti i fattori menzionati. Tale distribuzione viene successivamente scomposta in fasi e formulata come una catena di Markov. Ogni distribuzione così scomposta rappresenta un punto di ingresso flessibile per l'input dell'utente aziendale o può essere utilizzata per la generazione automatica, offrendo un ampio spettro di possibilità nella gestione del processo di progettazione.

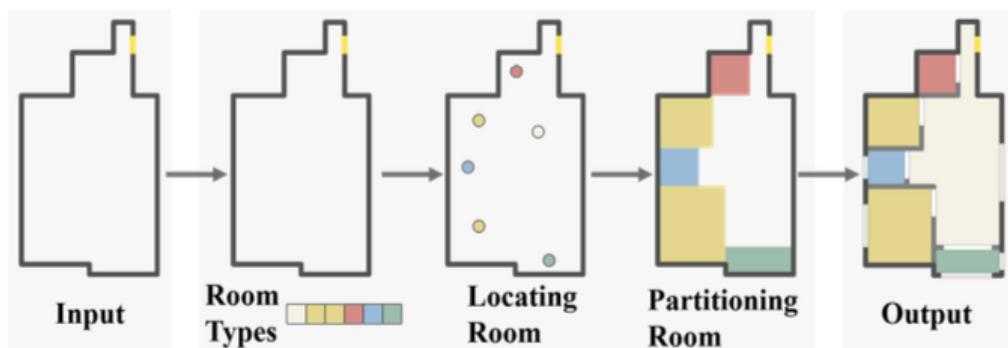


Figura 3.1: Esempio input e output.

3.1 Formulazione del problema

Un dataset di \mathbf{H} layout è denotato con $\mathbf{D} = \{\mathbf{D}_i\}_{i=1}^H$ con l'*i-esimo* layout $\mathbf{D}_i = (\mathbf{B}_i, \mathbf{R}_i, \mathbf{T}_i, N_i, \mathbf{C}_i)$. $\mathbf{B}_i \in \mathbb{R}^{128 \times 128}$ è la soglia, N_i denota il numero totale di stanze. Si utilizza j come indice per la specifica stanza. $\mathbf{R}_i = \{\mathbf{r}_{i,j}\}_{j=1}^{N_i}$ indica la regione della stanza con $\mathbf{r}_{i,j} \in \mathbb{R}^4$ che rappresenta l'angolo in alto a sinistra e l'angolo in basso a destra della bounding box della stanza. $\mathbf{T}_i = \{t_{i,j}\}_{j=1}^{N_i}$ è un insieme di tipologie di stanze dove $t_{i,j} \in \mathbb{N}^+$. $\mathbf{C}_i = \{\mathbf{c}_{i,j}\}_{j=1}^{N_i}$ è un insieme dei centri delle stanze $\mathbf{c}_{i,j} \in \mathbb{R}^2$.

Dato \mathbf{D} , il modello mira a progettare un **un modello generativo** per la distribuzione $\mathcal{P}(\mathbf{D}) = \prod_{i=1}^H \mathcal{P}(\mathbf{D}_i)$.

Lo scopo di questo modello è quello di formulare una planimetria come in un generatore di immagini, ma con il focus di integrare l'interazione umana nel processo di generazione, attraverso la corretta decomposizione di $\mathcal{P}(\mathbf{D})$. Da un punto di vista matematico, ci sono molti modi per decomporre $\mathcal{P}(\mathbf{D})$. Al fine di ridurre gli input umani attraverso vari livelli di dettaglio, si sono affidati ai principi di progettazione in modo da imitare il flusso di lavoro di progettazione umana e dividere naturalmente una procedura di progettazione della planimetria in diverse fasi.

Si possono discriminare tre fasi:

- Il numero di stanze desiderate e il loro tipo sono determinati.
- Vengono "più o meno" stimate le posizioni delle stanze e le loro funzionalità.
- Il partizionamento delle stanze conduce alla finalizzazione del progetto.

Questo diagramma serve come forte bias induttivo per il modello, ne segue la decomposizione di $P(\mathbf{D})$ in:

$$\begin{aligned} \mathcal{P}(\mathbf{D}) &= \prod_{i=1}^H \mathcal{P}(\mathbf{D}_i) = \prod_{i=1}^H \mathcal{P}(\mathbf{R}_i, \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i) \\ &= \prod_{i=1}^H \mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i) \mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i) \\ &\quad \mathcal{P}(\mathbf{T}_i, N_i | \mathbf{B}_i) \mathcal{P}(\mathbf{B}_i) \end{aligned} \quad (1)$$

Dove $\mathcal{P}(\mathbf{B}_i)$ tiene conto del confine conosciuto a priori; $\mathcal{P}(\mathbf{T}_i, N_i | \mathbf{B}_i)$ è per l'inferenza sul numero delle stanze e sul loro tipo;

$\mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i)$ e $\mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i)$ corrispondono rispettivamente alla progettazione parziale o grossolana e a quella finale, dove il primo stima le

posizioni della stanza mentre il secondo prevede le partizioni esatte.

Un esempio dell'equazione (1) è fornita nell'immagine 3.1, dove il secondo, il terzo e il quarto blocco corrispondono a $\mathcal{P}(\mathbf{T}_i, N_i | \mathbf{B}_i)$, $\mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i)$ e $\mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i)$ rispettivamente.

Ispirandosi alle scene interne sintetizzate dove gli oggetti sono posizionati interattivamente [4, 5], assumono che le stanze siano progettate una ad una. Formalmente modellano le probabilità $\mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i)$ e $\mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i)$ come catene di Markov, in modo tale che i progetti siano condotti in modo graduale e le decisioni iniziali influenzino quelle successive, il che consente a un progettista di concentrarsi su una stanza alla volta e dare una guida in qualsiasi fase:

$$\mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i) = \prod_{j=1}^{N_i} \mathcal{P}(\mathbf{c}_{i,j} | \mathbf{c}_{i,<j}, t_{i,j}, N_i, \mathbf{B}_i) \quad (2)$$

$$\mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i) = \prod_{j=1}^{N_i} \mathcal{P}(\mathbf{r}_{i,j} | \mathbf{r}_{i,<j}, \mathbf{c}_{i,j}, t_{i,j}, N_i, \mathbf{B}_i) \quad (3)$$

Dove $\mathbf{r}_{i,<j} = \{\mathbf{r}_{i,1}, \dots, \mathbf{r}_{i,j-1}\}$ e $\mathbf{c}_{i,<j} = \{\mathbf{c}_{i,1}, \dots, \mathbf{c}_{i,j-1}\}$ denotano l'insieme di partizioni e centri della stanza allocati per la j-esima stanza, rispettivamente.

3.1.1 Connessione ad altri paper

Le equazioni (1)-(3) sono una generalizzazione di metodi esistenti e più raffinati. Graph2Plan [26] determina simultaneamente \mathbf{C}_i , \mathbf{T}_i e da N_i fornisce \mathbf{B}_i , per poi predire $\mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i)$. RPLAN[1] stima $\mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i)$ tramite predizione consecutive di $\mathcal{P}(\mathbf{c}_{i,j}, t_{i,j} | \mathbf{c}_{i,<j}, t_{i,<j}, \mathbf{B}_i)$ e poi indirettamente stima l'area della stanza \mathbf{R}_i dalla locazione dei muri.

In contrapposizione a questo nel modello si decomponete ulteriormente $\mathcal{P}(\mathbf{C}_i, \mathbf{T}_i, N_i | \mathbf{B}_i)$ in $\mathcal{P}(\mathbf{T}_i, N_i | \mathbf{B}_i)$ e $\mathcal{P}(\mathbf{C}_i | \mathbf{T}_i, N_i, \mathbf{B}_i)$, per poi decomporre l'ultimo utilizzando l'equazione (2). Inoltre $\mathcal{P}(\mathbf{R}_i | \mathbf{C}_i, \mathbf{T}_i, N_i, \mathbf{B}_i)$ è decomposto in multipli step dall'equazione (3).

Le seguenti decomposizioni portano un modello generativo procedurale molto preciso che consente interazioni con l'utente a passi arbitrari. Ciò consente interazioni uomo-AI più flessibili e più vicine.

Per brevità, d'ora in poi verranno omessi i pedici i di $\mathbf{T}_i, N_i, \mathbf{B}_i$ e \mathbf{C}_i nelle sezioni successive.

3.2 Numero e tipo di stanze

\mathbf{T} e N sono date tipicamente in anticipo. In ogni caso si produce lo specifico \mathbf{B} , il quale però non è unico, per esempio in una stanza da letto ci potrebbero essere due letti singoli o un letto matrimoniale. In termini matematici esistono differenti distribuzioni per i possibili progetti.

Il modello in esame impara le seguenti distribuzioni $\mathcal{P}(\mathbf{T}, N|\mathbf{B})$ da dei reali progetti ideati da dei designer al fine di automatizzare la generazione. $\{\mathbf{T}, N\}$ può essere sostituita da una variabile random $\mathbf{Q} = \{q_k\}_{k=1}^K$, dove K indica il numero dei tipi di stanze in \mathbf{D} e q_k corrisponde al numero delle stanze al di sotto del k -esimo tipo. Quindi si modellerà $\mathcal{P}(\mathbf{Q}|\mathbf{B})$.

Il modello iPLAN propone un boundary-conditioned **Variational Autoencoder (BCVAE)** basato su **VAE** [27] dove \mathbf{B} serve come condizione. Il modello consiste in un modulo encapsulato F_{ed} , un encoder F_{en} e un decoder F_{de} . Dando in input \mathbf{B} ad F_{ed} si ottiene un vettore di embedding $\gamma \in \mathbb{R}^{128}$. Ne segue che $\{\mu, \Sigma\} = F_{en}(\mathbf{Q}, \gamma)$ dove $\mu \in \mathbb{R}^{32}$ e $\Sigma \in \mathbb{R}^{32 \times 32}$ sono la media e la covarianza della distribuzione Gaussiana. Inoltre una variabile latente z è campionata da $\mathbb{N}(\mu, \Sigma)$ usando un trucco rappresentativo [28]. Dato z e γ , si ricostruisce \mathbf{Q} da $\hat{\mathbf{Q}} = F_{ed}(z, \gamma)$. Si utilizza una standard **VAE loss** per la fase di training, tratteremo in un capitolo apposito il training.

Durante l'inferenza, per un nuovo \mathbf{B} , si campiona z da $N(\mathbf{0}, \mathbf{I})$ e si predice $\hat{\mathbf{Q}}$ da $\hat{\mathbf{Q}} = F_{de}(z, F_{ed}(\mathbf{B}))$, il quale viene poi utilizzato per recuperare $\hat{\mathbf{T}}$ e \hat{N} .

3.3 Individuazione delle stanze

La locazione delle stanze $\mathcal{P}(\mathbf{C}|\mathbf{T}, N, \mathbf{B})$ gioca un importante ruolo nel processo di progettazione. Similmente a [1] in questo modello si tratta la previsione della regione della stanza come un compito di classificazione step by step. Ad ogni step data una rappresentazione dell'immagine multicanale dello stato di progettazione corrente e del prossimo tipo di stanza desiderato t_j , il modello predice il centro della prossima stanza. L'immagine multicanale codifica il confine \mathbf{B} e tutti i centri precedentemente predetti $c_{<j}$. L'immagine consiste in $K + 4$ canali binari, tre dei quali etichette \mathbf{B} , per chiarezza: il confine inteso come delimitazione del progetto, la porta d'ingresso e i pixel dell'area interna. I K canali rappresentano i centri predetti delle stanze, con ciascun canale corrispondente a un tipo di stanza. Per ogni stanza, il centro è rappresentato da un quadrato di 9×9 pixel con valore 1. Si utilizza un canale apposito per riassumere i centri di tutte le stanze predette. Per le stanze desiderate t_j , il modello usa un vettore *one-hot*, ovvero un vettore con ogni valore posto a 0 tranne per la tipologia di stanza deside-

rata dove il valore sarà impostato a 1. La rappresentazione multicanale viene inviata a una rete **Resnet-18** [29] e il vettore one-hot, rappresentante il tipo di stanza desiderato, viene mandato a una **rete di embedding** per estrarre le caratteristiche. La rete di embedding ha tre layer fully-connected, seguiti da quattro blocchi convoluzionali, un layer di normalizzazione del Batch e un LeakyReLu layer. Le feature estratte sono concatenate e successivamente inoltrate al decoder che contiene quattro atrous spatial pyramid pooling (ASPP) [30], un blocco convoluzionale e un blocco deconvoluzionale. L'output è $\mathbf{O} \in \mathbb{R}^{(K+3) \times 128 \times 128}$, dando un vettore di probabilità di tutti i pixel. Per ogni pixel, predice $K + 3$ etichette. Come ad esempio K tipi di stanze e **EXISTING**, **FREE** e **OUTSIDE**, dove **EXISTING**, **FREE** e **OUTSIDE** indicano se un pixel appartiene a una stanza esistente, a uno spazio libero (all'interno del marco del progetto) o all'esterno, rispettivamente. Di seguito la rappresentazione dell'intero modello.

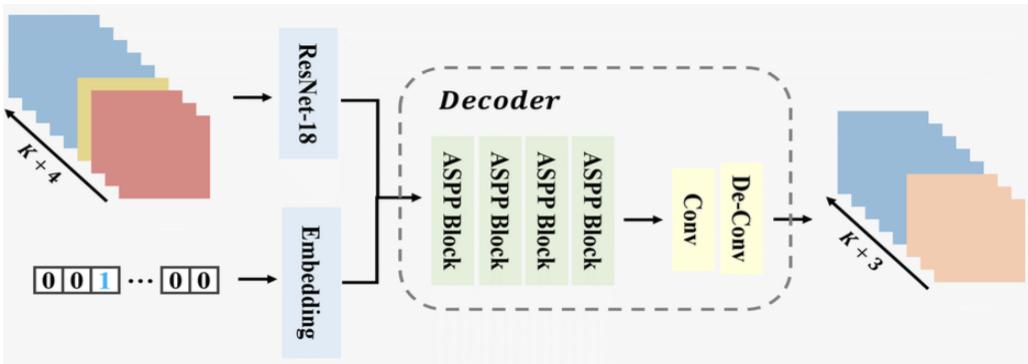


Figura 3.2: Pipeline RoomLocation.

Per la fase di training, si predice una stanza alla volta decomponendo il progetto finale in una serie di stati di progettazione con una stanza aggiunta per ogni stato. Si deve considerare che ci sono molteplici sequenze per ottenere il progetto finale e il groundtruth step-to-step non è disponibile. Per ovviare a questo problema nel modello si implementa un processo di training stocastico che impara tutte le possibili sequenze rimuovendo casualmente le stanze da un progetto.

Nel progetto si propone una funzione costo pixel-wise cross-entropy:

$$\mathcal{L} = \sum_{h=1}^{128} \sum_{w=1}^{128} -\omega_y \log \frac{\exp(O_{y,h,w})}{\sum_{k=1}^{K+3} \exp(O_{k,h,w})} \quad (4)$$

Dove y è l'indice della classe di ground-truth per il pixel situato in (h, w) . ω_y rappresenta il peso del y -esima etichetta, viene settato a 2 per le K tipologie

di stanze e 1.25 per le altre tre etichette.

Durante la fase di inferenza, per una sequenza di tipi di classi $\mathbf{T} = \{t_1, t_2, \dots, t_N\}$ prima viene predetto il centro per t_1 , ciò vuol dire $\mathcal{P}(\mathbf{c}_2 | \hat{\mathbf{c}}_1, t_2, N, \mathbf{B})$. Questa procedura viene ripetuta fino a quando non vengono determinati tutti i centri $\hat{\mathbf{C}}$. Si noti che quando gli ordini degli elementi in \mathbf{T} variano, anche i corrispondenti layout possono cambiare, fornendo così una diversificazione tra i progetti. Inoltre in qualsiasi fase intermedia, può essere incorporato l'input dall'utente, come ad esempio la sistemazione del centro della stanza, all'interno del modello prima che produca il prossimo centro di una stanza.

3.3.1 Atrous Spatial Pyramid Pooling (ASPP)

DeepLab [30] è un modello che si concentra sulla segmentazione semantica delle immagini, distinguendo e assegnando etichette semantiche ai pixel nell'immagine.

L'architettura DeepLab comprende vari elementi chiave:

Backbones preaddestrati DeepLab può utilizzare reti neurali preaddestrate, come ad esempio ResNet, VGG o MobileNet, come componenti principali della rete per l'estrazione delle feature.

Atrous (Dilated) Convolutions Un aspetto distintivo di DeepLab è l'uso di convoluzioni atrous (o dilatate), che consentono di aumentare il campo receptivo senza compromettere la risoluzione spaziale. Questo è ottenuto introducendo "holes" (buchi) tra i neuroni, consentendo la convoluzione su una finestra più ampia mantenendo la risoluzione dell'immagine originale.

ASPP (Atrous Spatial Pyramid Pooling) L'ASPP è una tecnica che permette di catturare le informazioni a diverse scale spaziali. È costituito da più convoluzioni atrous con diverse scale di dilatazione per catturare informazioni dettagliate a diverse dimensioni spaziali.

Fully Connected CRFs DeepLab incorpora l'uso di Conditional Random Fields (CRF) completamente connessi per migliorare la coerenza spaziale dell'output. Questi CRF vengono utilizzati per raffinare le previsioni pixel per pixel, considerando le relazioni spaziali tra i pixel etichettati.

Output e Processo di Inferenza L'output del modello è una mappa di segmentazione in cui ogni pixel è etichettato con l'etichetta corrispondente alla classe semantica. Durante l'inferenza, l'immagine di input viene passata

attraverso la rete e ogni pixel viene etichettato in base alla sua appartenenza alle diverse classi semantiche.

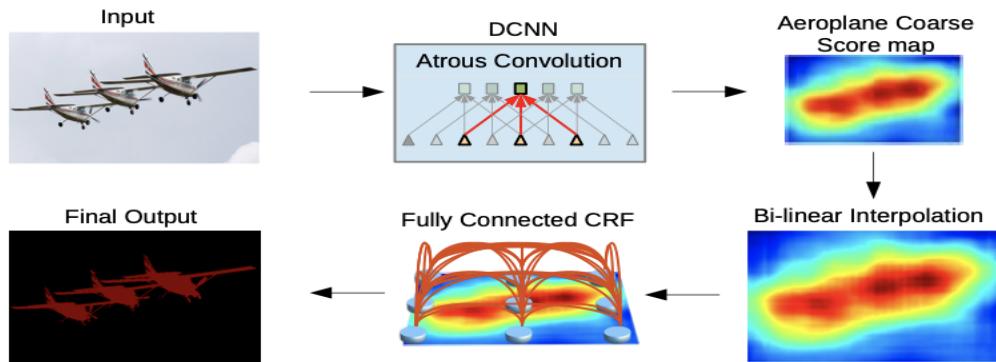


Figura 3.3: Esempio input-output ASPP.

L’architettura e il funzionamento di DeepLab integrano dunque una combinazione di tecniche avanzate di rete neurale convoluzionale, tra cui convoluzioni dilatate, pooling piramidale spaziale atrous e l’uso di CRF completamente connessi, per ottenere risultati avanzati nella segmentazione semantica delle immagini.

3.4 Predizione del partizionamento della stanza

Dopo aver predetto il centro delle stanze $\hat{\mathbf{C}}$, il modello procede con il partizionamento della stanza $\mathcal{P}(\mathbf{R}|\hat{\mathbf{C}}, \hat{\mathbf{T}}, \hat{\mathbf{N}}, \mathbf{B})$, dove bisogna considerare la grandezza della stanza e la forma. La grandezza della stanza è direttamente collegata alla specifica funzionalità della stessa, per esempio si consideri che tipicamente la camera da letto è più grande del bagno. La forma della stanza è anche soggetta alla funzionalità ma è fortemente dipendente dalla forma geometrica dei muri interni e esterni, quindi dai confini del progetto. Di solito non esiste una soluzione ottimale ma una distribuzione di soluzioni quasi ottimali che si vedono nei progetti finali.

Poiché la distribuzione può essere arbitraria, nel modello si propone una nuova Generative Adversarial Network (GAN) per modellare la previsione step-wise del partizionamento della stanza.

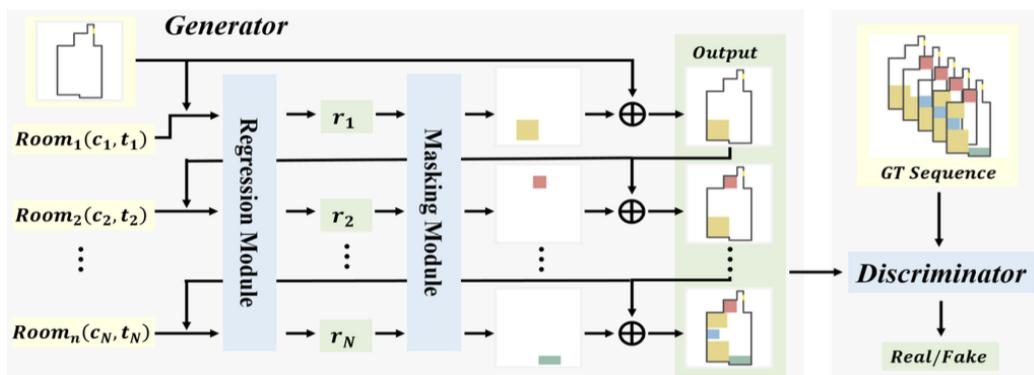


Figura 3.4: GAN.

Invece di predire direttamente la forma di ogni stanza, nel modello per prima cosa si predicono i bounding box. Ciò consente alla bounding box predetta in una fase successiva di considerare lo spazio che è già assegnato.

Generatore: Il generatore comincia con un regressore per i bounding-box \mathcal{F}_b che ritorna l'angolo in alto a sinistra e quello in basso a destra della box. \mathcal{F}_b consiste in sei unità convoluzionali e due fully connected layer, ogni unità ha un layer Convuluzionale, un Normalization-Layer e un ReLu. \mathcal{F}_b consente al progettista di interagire facilmente con il sistema. Può ad esempio trasferire una stanza, modificare una bounding box della stanza. In ogni caso, la predizione dei soli angoli della bounding box non è conveniente. Per questo motivo l'obiettivo è quello di etichettare tutti i pixel all'interno della

bounding box in modo da poter lavorare nello spazio in modo coerente. Per questo motivo è stato progettato un modello per le maschere fatto su misura \mathcal{F}_m che mappa ogni bounding box della stanza nella maschera della stanza. \mathcal{F}_m contiene sei layer convoluzionali con 3×3 kernel, utilizzando una funzione di attivazione sinosuidale.

Per una sequenza dei centroidi di una stanza e di tipi della stanza $\{(\mathbf{c}_1, t_1), (\mathbf{c}_2, t_2), \dots, (\mathbf{c}_N, t_N)\}$, si usa \mathcal{F}_b per predire le coordinate $\hat{\mathbf{r}}_1$ della bounding box relativa alla prima stanza specificata da \mathbf{c}_1 e t_1 , dove $\hat{\mathbf{r}}_1 = \mathcal{F}_b(\mathbf{S}_0, \mathbf{c}_1, t_1)$ con $\mathbf{S}_0 = \mathbf{B}$. Inoltre si ottiene la maschera della stanza utilizzando $\hat{\mathbf{M}}_1 = \mathcal{F}_m(\hat{\mathbf{r}}_1) \in \mathbb{R}^{128 \times 128}$.

L'output del generatore al primo step è computato da

$$\hat{\mathbf{S}}_1 = \mathbf{S}_0 \times (\mathbf{1} - \hat{\mathbf{M}}_1) + \hat{\mathbf{M}}_1 \times t_1 \quad (5)$$

Dove $\mathbf{1} \in \mathbb{R}^{128 \times 128}$ rappresenta una matrice con tutti gli elementi posti a 1. Durante la predizione nel secondo step, il generatore prende come input $\hat{\mathbf{S}}_1, \mathbf{c}_2, t_2$ e genera in output $\hat{\mathbf{S}}_2 = \hat{\mathbf{S}}_1 \times (\mathbf{1} - \hat{\mathbf{M}}_2) + \hat{\mathbf{M}}_2 \times t_2$ con $\hat{\mathbf{M}}_2 = \mathcal{F}_m(\hat{\mathbf{r}}_2) = \mathcal{F}_m(\mathcal{F}_b(\hat{\mathbf{S}}_1, \mathbf{c}_2, t_2))$. La stessa procedura è iterata per \hat{N} volte, cioè finchè l'intero progetto non sarà completato.

Discriminatore: il discriminatore utilizza la stessa backbone di \mathcal{F}_b , eccezione fatta per gli ultimi due layer che sono assenti. Il discriminatore è utilizzato per distinguere se una sequenza di stati di progettazione proviene dal generatore o dal ground truth. Nello specifico, una sequenza di stati di progetti predetti $\hat{\mathbf{S}} = \{\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2, \dots, \hat{\mathbf{S}}_{\hat{N}}\}$ dovrebbe essere associata come '**FALSE**', mentre una sequenza del ground truth con lo stesso ordine di tipologia di stanze, es. $\{t_1, t_2, \dots, t_{\hat{N}}\}$, dovrebbero produrre '**TRUE**'.

Per la fase di training viene utilizzata una funzione di loss dall'alto con WGAN-GP [31]

$$\begin{aligned} \mathcal{L} &= \mathbb{E}_{\hat{\mathbf{S}} \sim \mathbb{P}_g}[D(\hat{\mathbf{S}})] - \mathbb{E}_{\mathbf{S} \sim \mathbb{P}_r}[D(\mathbf{S})] \\ &\quad + \lambda_1 \mathbb{E}_{\bar{\mathbf{S}} \sim \mathbb{P}_{\bar{\mathbf{S}}}}(||\nabla_{\bar{\mathbf{S}}} D(\bar{\mathbf{S}})||_2 - 1)^2 + \lambda_2 \mathcal{L}_s \end{aligned} \quad (6)$$

Dove \mathbb{P}_g e \mathbb{P}_r rappresentano rispettivamente la distribuzione generata e quella reale. $\bar{\mathbf{S}} \sim \mathbb{P}_{\bar{\mathbf{S}}}$ denotano un'interpolazione random tra $\hat{\mathbf{S}}$ e \mathbf{S} , ed è usata una penalità per il gradiente di $\lambda_1 = 10 \cdot ||\cdot||_2$ che è la norma l_2 . L'ultimo termine è introdotto per regolarizzare esplicitamente le bounding box con $\lambda_2 = 100$:

$$\mathcal{L}_s = \sum_{j=1}^N l_j, \quad l_j = \begin{cases} 0.5(\mathbf{r}_j - \hat{\mathbf{r}}_j), & \text{se } ||\mathbf{r}_j - \hat{\mathbf{r}}_j||_1 < 1 \\ ||\mathbf{r}_j - \hat{\mathbf{r}}_j||_1 - 0.5, & \text{altrimenti} \end{cases} \quad (7)$$

Dove \mathbf{r}_j è il ground truth e $\|\cdot\|_1$ è la norma l_1 .

Nella fase di inferenza, il modello procedurale predice le aree della stanza in modo graduale, il che quindi consente l'immisione dell'input da parte dell'utente in una fase intermedia arbitraria, come la modifica dei tipi di stanza, modifica dei centri delle stanze o anche le aree della stanza già predette. Potrebbero esserci lacune tra le stanze previste. Pertanto, si impiega una semplice fase di postelaborazione come in [32], verrà affrontata successivamente.

Capitolo 4

Esperimenti

4.1 Dataset

Gli esperimenti sono stati condotti su due dataset comunemente utilizzati, RPLAN [1] e LIFULL [2]. **RPLAN** è raccolto dagli edifici residenziali del mondo reale nel mercato immobiliare asiatico, che contiene oltre 80k planimetrie e 13 tipi di stanze: *Soggiorno, Camera padronale, Cucina, Bagno, Sala da pranzo, Camera dei bambini, Studio, Seconda camera, Camera degli ospiti, Balcone, Ingresso, Ripostiglio, Mura*. Tutte le planimetrie in RPLAN sono allineate all’asse e pre-elaborate sulla stessa scala. Il dataset è stato diviso per il training, validation e test nella seguente maniera 70% – 15% – 15% [26]. **LIFULL** offre circa cinque milioni planimetrie di appartamenti del mercato immobiliare giapponese. Il dataset originale è dato sotto forma di immagini, ma un sottoinsieme è stato analizzato in formato vettoriale da [33]. Da questo dataset sono state scelte un sottoinsieme di 4 – 10 stanze. Questo specifico dataset consiste in circa cinquantaquattromila planimetrie e nove 9 tipi di stanze: *Soggiorno, Cucina, Camera da letto, Bagno, Ufficio, Balcone, Corridoio, Altro*; di questi l’85% (campionato a caso) dei dati serve come set di allenamento mentre il rimanente per i test.

4.2 Metriche

Anche se ogni planimetria contiene solo un disegno (cioè, un confine corrisponde a un disegno), il modello può prevedere la distribuzione di disegni plausibili, offrendo diversità di progettazione e scelte alternative. Per valutare la distribuzione predetta, utilizzano il *Frechet Inception Distance (FID)* [34] per calcolare la distanza tra due distribuzioni, che è stata utilizzata

anche nella generazione della planimetrie [24, 25]. Costruite con **FID**, si introducono tre metriche FID_{img} , FID_{area} , FID_{type} .

1. FID_{img} : calcolato su immagini renderizzate per valutare la differenza tra l’immagine generata e l’immagine reale.
2. FID_{area} : per valutare le differenze tra le distribuzioni delle aree della stanza. Ogni layout è rappresentato da un vettore $area_i$ $1 \times K$, con il suo k-esimo elemento $area_{i,k}$ rappresenta l’area media del k-esimo tipo di stanze dell’i-esima piantina. K indica il numero di tipi di stanza.
3. FID_{type} : per calcolare le differenze tra le distribuzioni dei numeri delle stanze rispetto ai tipi di stanze. Ogni layout è rappresentato da un vettore $type_i$ $1 \times K$, il cui elemento $type_{i,k}$ rappresenta il numero di stanze del k-esimo tipo nell’iesima piantina.

Le tre metriche non sono portate a convergenza dal metodo, poichè non sono coinvolte nella fase di training.

4.3 Baselines

Come baselines, sono stati selezionati tre metodi. Rplan [1] è un approccio basato su immagini che genera una planimetria a partire da un contorno. Viene anche utilizzata una variante di Rplan, denominata *Rplan**, che incorpora come input i tipi di stanza e i relativi centri. HouseGan++ [25] è un metodo vincolato al grafo, che tratta le stanze come nodi e richiede tipi di nodi e connettività, utilizzando un input simile a [6]. Produce quindi una maschera per tutti i nodi in base al grafo, per poi combinare le informazioni e ottenere il layout. Graph2plan [26] richiede un contorno e un grafo delle relazioni tra le stanze come input, in cui il grafo contiene informazioni sulle dimensioni, il centro, il tipo e le connessioni delle stanze. Tutte le planimetrie vengono renderizzate seguendo lo stesso metodo di [26], inclusi i dettagli relativi alla posizione di porte e finestre.

4.4 Valutazioni Quantitative

Studio di ablazione iPLAN è capace di generare completamente automaticamente progetti permettendo in qualsiasi stage l’interazione da parte dell’utente. Sulla base delle informazioni ricevute dal progettista, vengono valutate tre varianti (v1) *Our_I* prende come input il contorno, il tipo di stanza e il centro di essa; (v2) *Our_{II}* prende come input il contorno e il tipo

di stanza; (v3) Our_{III} prende in input solamente il contorno. Le varianti mostrano differenti livelli di automazione/interazione con l’utente. I dettagli dell’implementazione sono in docuemnto a parte.

Dataset	Method	FID_{img}	FID_{area}	FID_{type}
RPLAN	HouseGAN++	51.33	1.36×10^8	0.038
	Rplan	4.1	2.29×10^5	0.58
	Our_{III}	1.22	3.13×10^4	0.05
	Our_{II}	0.72	1.09×10^4	0.03
	Rplan*	0.11	8.62×10^3	6.4×10^{-3}
	Graph2Plan	0.62	8.82×10^3	2.70×10^{-4}
LIFULL	Our_I	0.16	4.89×10^2	4.44×10^{-6}
	Rplan	50.19	4.29×10^6	5.15
	Our_{III}	37.35	9.81×10^5	2.52
	Our_{II}	32.65	7.75×10^5	2.14
	Rplan*	1.43	5.62×10^5	0.064
	Graph2Plan	0.64	2.87×10^3	2.63×10^{-5}
	Our_I	0.38	2.07×10^3	3.59×10^{-6}

Figura 4.1: Metriche FID su RPLAN e LIFULL.

Nella tabella 4.1 sono riportati i risultati. iPLAN ottiene buoni risultati con tutte e tre le modalità, tra cui Our_I ottiene i migliori risultati e Our_{II} è migliore di Our_{III} . Questo avviene poichè più informazioni permettono una migliore predizione. Considerando che RPLAN e LIFULL sono significativamente diversi in termini di forme generali dei contorni e delle stanze, iPLAN può effettivamente gestire bene diverse distribuzioni di dati.

Comparazione su RPLAN iPLAN generalmente supera di granlunga tutti i metodi di base con ampi margini mostrati nella tabella 4.1. Guardando la tabella, HouseGan++ ottiene risultati peggiori, tranne FID_{type} con un contorno piccolo, rispetto altri metodi. Il motivo è che i suoi layout non sono delimitati da contorni. Prendendo in input il confine, entrambi Rplan e Our_{III} predicono il centri della stanza e le regioni consecutivamente, ma Our_{III} supera Rplan in tutte e tre le metrice. Inoltre quando sono dati sia il bordo che i tipi di stanze, Our_{II} supera i primi tre metodi. In fine vengono fornite tutte le informazioni (contorno, centro e tipo di stanza) a Rplan*, Graph2Plan e Our_I . Our_I migliora ulteriormente tutte le metriche, dove Rplan* ottiene un risultato leggermente migliore in FID_{img} . Si è osservato che nella predizione passo passo in iPLAN è meno probabile generare partizioni ambigue, mentre si osservano generazioni ambigue in Rplan*. Inoltre, l’apprendimento delle probabilità condizionali (come fa iPLAN) in opposizione a una probabilità congiunta di più fattori dai progetti finali (come fanno Rplan* e Graph2Plan) è più facile, dove la decomposizione dei progetti finali può essere vista come una strategia di aumento dei dati.

Comparazione su LIFULL Confronti simili sono stati fatti su LIFULL [4.1](#). Non viene considerata HouseGAN++ perché ha bisogno di posizioni delle porte nell'input, ma non ci sono questi dati in LIFULL. Comparando *Rplan**, *Our_I*, e Graph2Plan, i centri della stanza non sono forniti a priori come per Rplan, *Our_{II}* e *Our_{III}*, portando a un ampio divario tra questi due gruppi. È comprensibile perché LIFULL è più chellenging/eterogeno con campioni multiscala e stanze nidificate. Successivamente, *Our_{II}* ottiene risultati migliori di *Our_{III}* su tutte e tre le metriche perché viene data più conoscenza apriori (cioè i tipi di stanza), ed entrambe sono migliori di Rplan. Inoltre, quando i centri della stanza sono forniti, la previsione imprecisa del muro influisce su *Rplan**. Diversa dai risultati su RPLAN, Graph2Plan supera *Rplan** su LIFULL. Questo perché Graph2Plan prende le relazioni spaziali come input che lo rende robusto per dataset multiscala. *Our_I* è anche qui migliore di *Rplan** e Graph2Plan. Pertanto, si può concludere che, quando viene data la stessa quantità di conoscenza pregressa, iPLAN raggiunge le migliori prestazioni.

4.5 Valutazione Qualitativa

Per mostrare risultati intuitivi, i confronti qualitativi sono riportati nella Figura 4.2.

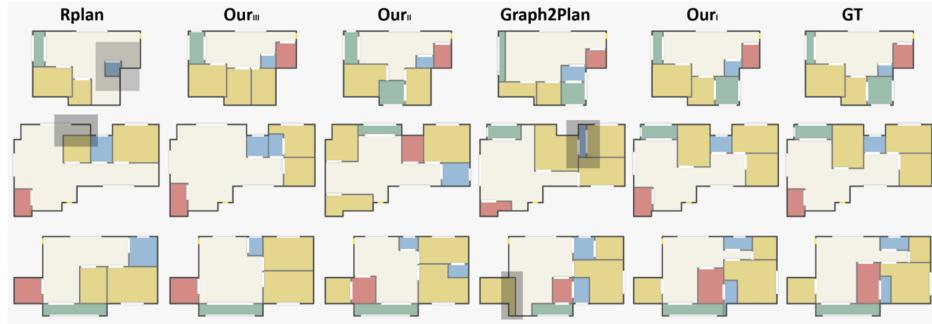


Figura 4.2: Confronti qualitativi.

Ogni colore rappresenta un tipo di stanza (le indicazioni del colore sono riportate in Figura 4.3)

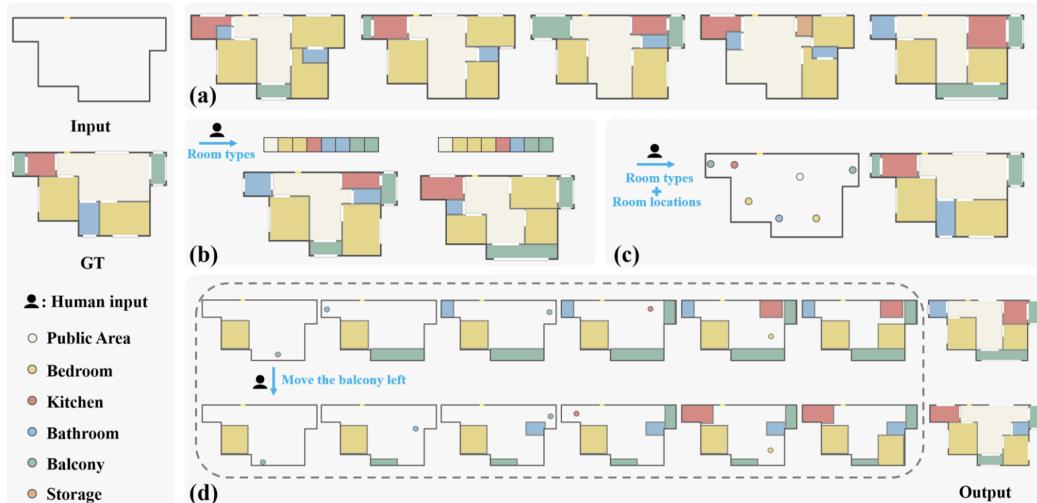


Figura 4.3: Indicazione colori.

Le caselle ombreggiate indicano aree di progettazione discutibili. In particolare, nella prima fila, Rplan progetta un bagno isolato in un angolo nella prima immagine, che è un design strano e non osservato nei dati. Nella seconda riga, la prima immagine di Rplan alloca uno spazio non edificabile (ombreggiato) all'area pubblica, che tuttavia dovrebbe essere assegnata alla camera da letto vicina. Allo stesso modo, la disposizione prodotta da Graph2Plan consiste in 3 camere da letto (seconda fila), ma il bagno è incorporato tra

loro (ombreggiato), il che significa che le persone devono passare attraverso una camera da letto per andare in bagno. Inoltre, il bagno è troppo piccolo. Per quanto riguarda l'ultima riga, l'area della stanza non è correttamente prevista da Graph2plan perché la regione (ombreggiata) nell'angolo in basso a sinistra è troppo stretta.

Confrontando Our_I , Our_{II} e Our_{III} , è evidente che Our_I è più vicino a GT a causa dei forti vincoli di input. Al contrario, quando vengono imposti meno vincoli, la diversità è significativa, come descritto in Our_{II} e Our_{III} . Ulteriori risultati e analisi sono mostrati nel materiale supplementare.

4.6 Interazioni con L'Utente

Le prestazioni di iPLAN sotto diversi livelli di interazioni umane sono mostrate nella Figura 4.3. La figura 4.3(a) è una generazione completamente automatizzata senza input umano eccezione per il contorno. iPLAN può generare diversi layout, con un numero variabile di stanze e diversi tipi/aree di stanza. La figura 4.3(b) mostra come l'utente può decidere i tipi di stanza, dove i layout finali sono diversi quando l'ordine dei tipi di stanza varia, mostrando la flessibilità di iPLAN. Inoltre fornendo a iPLAN più informazioni apriori figura 4.3(c), come: confine, tipi di stanze e centri; si ottiene un layout pianificato che è quasi lo stesso del ground truth. Viene anche valutato iPLAN introducendo un input dell'utente in un passaggio intermedio, figura 4.3(d), dove il balcone viene spostato a sinistra e di conseguenza si ottiene un layout diverso. La figura 4.3 dimostra la capacità interattiva di iPLAN, che è cruciale nei contesti in cui il designer guida il progetto.

4.7 Generalizzabilità

Risultati quantitativi Viene valutata quantitativamente la generalizzabilità di iPLAN impostando due gruppi di esperimenti su RPLAN. Nel primo gruppo, vengono selezionati 51322 layout composti da $4 \sim 7$ stanze per l’allenamento e scegliendo a caso 12.000 layout di 8 stanze per il test. Nel secondo gruppo, si considera un compito più impegnativo selezionando a caso 26574 layout contenenti $4 \sim 6$ stanze per il training e 12000 layout contenenti $7 \sim 8$ stanze per i test. È importante notare che in entrambi gli esperimenti, i layout per il test includono più stanze di quelle utilizzate per l’allenamento. I risultati quantitativi sono riportati nella tabella 4.4.

Dataset	Method	FID_{img}	FID_{area}	FID_{type}
RPLAN	Our_{II}^8	1.2	1.36×10^4	0.06
	Our_I^8	0.2	2.69×10^2	1.25×10^{-5}
	$Our_{II}^{7\sim 8}$	2.28	4.87×10^4	0.18
	$Our_I^{7\sim 8}$	0.24	6.89×10^2	1.54×10^{-5}

Figura 4.4: Risultati quantitativi.

Possiamo osservare che Our_I supera Our_{II} , che è consistente con l’intuizione fornita poiché sono fornite più conoscenze pregresse a Our_I . Inoltre, i risultati nel primo gruppo sono migliori di quelli del secondo gruppo, il che è ragionevole poiché l’impostazione di generalizzazione nel secondo gruppo è più impegnativa. Tuttavia, entrambi i gruppi raggiungono prestazioni comparabili alla tabella 4.1 e superando le linee di base.

Risultati qualitativi Un test svolto è quello di vedere se allenando iPLAN con RPLAN può essere generalizzato a tipi di contorni non visti, inteso come forme di planimetrie mai viste. Si noti che RPLAN contiene solo layout con contorni dritti allineati all'asse dei confini, quindi si considerano i confini con bordi e curve non allineati all'asse. Poiché HouseGAN++ non può progettare un layout per un confine specifico dell'edificio e Graph2Plan è limitato ai confini con bordi allineati all'asse, si confronta iPLAN solo con Rplan.

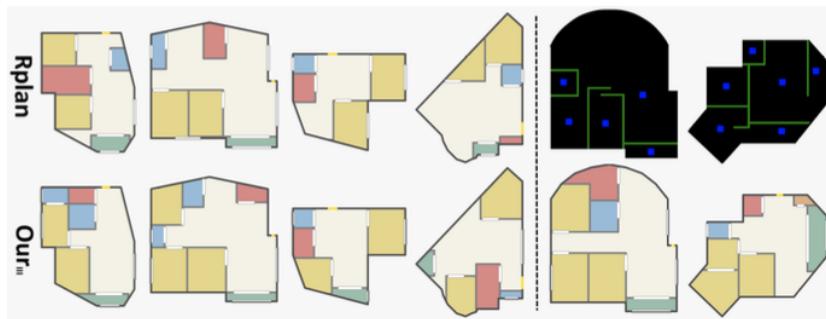


Figura 4.5: Layout con bordi non allineati all'asse. A sinistra: esempi di successo. A destra: Rplan fallisce ma iPLAN ha successo.

La figura 4.5 mostra diversi esempi di design non allineato all'asse prodotti da Rplan e iPLAN. Nella sinistra elenca quattro casi di successo su entrambi gli approcci e sulla destra mostra due esempi in cui Rplan fallisce ma iPLAN ha successo. Anche se ha successo, Rplan spesso prevede stanze isolate che dividono le aree pubbliche in forme strane e riducono la loro usabilità (i primi due esempi), mentre iPLAN utilizza meglio lo spazio. Per quanto riguarda i fallimenti di Rplan, si danno due esempi di stima del centro della stanza e di previsione del muro, dove non riesce a riempire e ripartire ragionevolmente lo spazio, un problema di cui iPLAN non soffre.

Capitolo 5

Specifiche

5.1 Architettura di BCVAE

I dettagli dell'architettura di BCVAE sono mostrati nella tabella 5.1.

Per un determinato layout, $\mathbf{Q} = \{q_k\}_{k=1}^K$ rappresenta il tipo di stanza, dove K rappresenta il numero dei tipi di stanze in \mathbf{D} e $q_k \in \mathbb{Z}$ corrispondono al numero di stanze al di sotto del k -esimo tipo.

Dopo aver fornito \mathbf{Q} a BCVAE, è implementata una riformulazione. Prima viene determinata il più grande numero di stanze per ogni k tipo attraverso l'intero set di dati e lo si denota con $q_k^* \in \mathbb{Z}$. Dopodiché, ogni $q_k \in \mathbf{Q}$ ($q_k \leq q_k^*$), viene trasformato in un vettore $q_k^* \cdot \mathbf{D}$, v_k dove i primi elementi q_k vengono impostati a 1 mentre gli elementi rimanenti a 0. Concatenando tutti i vettori trasformati, si ottiene una rappresentazione alternativa di \mathbf{Q} , $\mathbf{V} = [\mathbf{v}_1^T, \mathbf{v}_2^T, \dots, \mathbf{v}_K^T]^T$.

L'output di BCVAE viene denotato con $\hat{\mathbf{V}}$ e viene utilizzata come funzione costi l'entropy cross loss:

$$\mathcal{L}_{rec} = \sum_{j=1}^{n_c} l_j, l_i = -[v_j \log \hat{v}_j + (1 - v_j) \log (1 - \hat{v}_j)] \quad (8)$$

Dove $n_c = \sum_{i=1}^K q_k^*$ rappresenta la lunghezza di \mathbf{V} .

La perdita totale di BCVAE è:

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda D_{KL}(\mathcal{N}(\mu, \Sigma) || \mathcal{N}(\mathbf{0}, \mathbf{I})) \quad (9)$$

Dove D_{KL} rappresenta la divergenza di Kullback–Leibler (KL), $\lambda = 0.5$.

Architecture	Layer	Specification	Output Size
embedding network	<i>conv.bn.relu</i> ₁	$1 \times 16 \times 4 \times 4(s = 2, p = 1)$	$64 \times 64 \times 16$
	<i>conv.bn.relu</i> ₂	$16 \times 16 \times 4 \times 4(s = 2, p = 1)$	$32 \times 32 \times 16$
	<i>conv.bn.relu</i> ₃	$16 \times 32 \times 4 \times 4(s = 2, p = 1)$	$16 \times 16 \times 32$
	<i>conv.bn.relu</i> ₄	$32 \times 32 \times 4 \times 4(s = 2, p = 1)$	$8 \times 8 \times 32$
	<i>conv.bn.relu</i> ₅	$32 \times 16 \times 4 \times 4(s = 2, p = 1)$	$4 \times 4 \times 16$
	<i>conv.bn.relu</i> ₆	$16 \times 16 \times 4 \times 4(s = 2, p = 1)$	$2 \times 2 \times 16$
	flatten	N/A	1×64
encoder	<i>concat</i>	N/A	$1 \times (n_c + 64)$
	<i>linear.relu</i> ₁	$(n_c + 64) \times 128$	1×128
	<i>linear.relu</i> ₂	128×64	1×64
	<i>linear</i> ₃₁	64×32	1×32
decoder	<i>linear</i> ₃₂	64×32	1×32
	<i>concat</i>	N/A	1×96
	<i>linear.relu</i> ₁	96×96	1×96
	<i>linear.relu</i> ₂	96×64	1×64
decoder	<i>linear</i> ₃	$64 \times n_c$	$1 \times n_c$
	<i>sigmoid</i>	N/A	$1 \times n_c$

Figura 5.1: Le specifiche architettoniche di BCVAE. s e p denotano rispettivamente stride e padding. n_c è la dimensione del tipo di casa. I kernel di convoluzione e l'output del layer sono specificati separatamente da $(N_{in} \times N_{out} \times W \times H)$ e $(W \times H \times C)$.

5.2 Postelaborazione per predizione del partizionamento della stanza

Dopo la previsione del partizionamento della stanza, possono esistere spazi tra le stanze previste $\hat{\mathbf{R}} = \{\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2, \dots, \hat{\mathbf{r}}_N\}$. Viene utilizzata una fase di postelaborazione per garantire che l'area interna di \mathbf{B} sia completamente coperta e che le bounding box della stanza si trovino all'interno di \mathbf{B} . Si definisce un problema di ottimizzazione generico:

$$\arg \min_{\hat{\mathbf{R}}} \mathcal{L} = \arg \min_{\hat{\mathbf{R}}} \mathcal{L}_{coverage}(\hat{\mathbf{R}}, \mathbf{B}) + \mathcal{L}_{interior}(\hat{\mathbf{R}}, \mathbf{B}) \quad (10)$$

Dove $\mathcal{L}_{coverage}$ e $\mathcal{L}_{interior}$ definiscono la coerenza spaziale tra \mathbf{B} e il set di bounding box della stanza \mathbf{R}^* .

Per spiegare $\mathcal{L}_{coverage}$ e $\mathcal{L}_{interior}$ chiaramente, viene definita la funzione di distanza $d(p, \mathbf{r})$ per misurare la copertura di un punto p con una box \mathbf{r} :

$$d(p, \mathbf{r}) = \begin{cases} 0, & \text{se } p \in \Omega_{in}(\mathbf{r}) \\ \min_{q \in \Omega_{bd}(\mathbf{r})} \|p - q\|, & \text{altrimenti} \end{cases} \quad (11)$$

Dove $\Omega_{in}(\mathbf{r})$ denota l'area interna della box \mathbf{r} e $\Omega_{bd}(\mathbf{r})$ rappresenta il confine di \mathbf{r} .

La perdita di copertura può essere definita come:

$$\mathcal{L}_{coverage}(\hat{\mathbf{R}}, \mathbf{B}) = \frac{\sum_{p \in \Omega_{in}(\mathbf{B})} \min_i d^2(p, \hat{\mathbf{r}}_i)}{|\Omega_{in}(\mathbf{B})|} \quad (12)$$

Dove $|\Omega_{in}|$ è il numero di pixel all'interno dell'insieme $\Omega_{in}(\mathbf{B})$.

La perdita interna può essere denotata come segue:

$$\mathcal{L}_{interior}(\hat{\mathbf{R}}, \mathbf{B}) = \frac{\sum_i \sum_{p \in \Omega_{in}(\hat{\mathbf{r}}_i)} d^2(p, \hat{\mathbf{B}})}{\sum_i |\Omega_{in}(\hat{\mathbf{r}}_i)|} \quad (13)$$

Dove $\hat{\mathbf{B}}$ è la bounding box del confine. Si noti che $\mathbf{B} \subseteq \hat{\mathbf{B}}$.

Pertanto, nella fase di inferenza, si adeguano direttamente le stanze previste $\hat{\mathbf{R}} = \{\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2, \dots, \hat{\mathbf{r}}_N\}$ con la loss minima ottenuta dall'equazione (10).

5.3 Ulteriori confronti qualitativi

La figura 5.2 e la figura 5.3 mostrano risultati qualitativi su RPLAN e LIFULL, rispettivamente. In entrambi i set di dati, Graph2Plan e Our_I sono forniti con l'input umano completo (compresi il bordo, i tipi di stanza e le posizioni delle stanze), i loro layout generati dovrebbero essere simili al GT. Questo avviene nei risultati di Our_I , ma non sembra essere così per Graph2Plan. Le aree ombreggiate sui layout prodotti da Graph2Plan mostrano chiare differenze rispetto ai layout GT. Al contrario, i layout di Our_I sono quasi gli stessi del GT.

Inoltre, iPLAN viene confrontato con altri metodi su un set di dati più impegnativo, cioè LIFULL. Quando viene fornito solo il confine della casa, Our_{III} supera Rplan. Our_{II} corrisponde al caso in cui il confine della casa e i tipi di stanza sono noti, il che raggiunge previsioni leggermente migliori di Our_{III} man mano che vengono fornite più informazioni. Se viene fornito l'input umano completo, Our_I si comporta meglio di Our_{II} .

È importante notare che Our_I è superiore di Graph2Plan anche nel caso in cui gli venga fornito l'input umano completo.

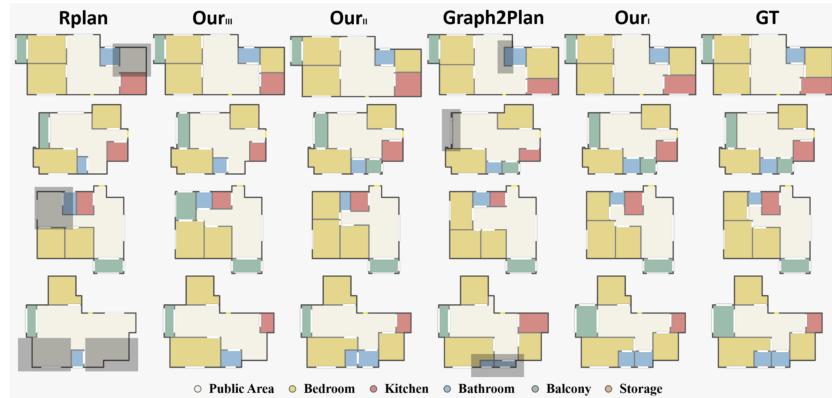


Figura 5.2: Confronti qualitativi su RPLAN. Le aree ombreggiate indicano scelte di design discutibili.

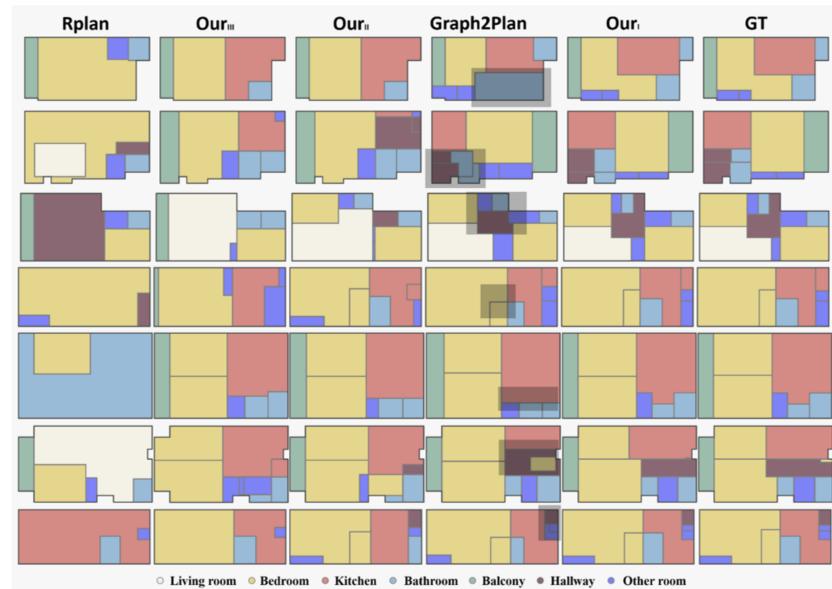


Figura 5.3: Confronti qualitativi su LIFULL. Le aree ombreggiate indicano scelte di design discutibili.

5.4 Ulteriori valutazioni di generalizzazione

Ulteriori risultati di generalizzazione su RPLAN sono presentati nella figura sottostante.



Figura 5.4: Layout con bordi non allineati all’asse. A sinistra: esempi di successo. A destra: Rplan fallisce ma il modello ha successo.

La prima e la seconda riga corrispondono rispettivamente a Rplan e Our_{III} . Our_{III} raggiunge risultati migliori. Coerentemente con le analisi nel documento principale, Rplan è incline a dividere l’area pubblica in due aree principali, causando potenziali inconvenienti per le attività familiari (le prime due colonne in figura 5.4). A volte, Rplan non riesce neanche a pianificare un bagno nel layout (la terza colonna in figura 5.4). In più, su alcuni confini, Rplan non riesce a progettare i layout (le ultime tre colonne in figura 5.4). In generale, Our_{III} supera Rplan quando il confine non è allineato all’asse.

5.5 Dettagli sull’implementazione

iPLAN è in PyTorch. Tutti i modelli sono addestrati e testati su una NVIDIA GeForce RTX 2080 Ti. Ci vogliono circa due ore per addestrare BCVAE, due giorni per ottimizzare la rete di localizzazione della stanza e un giorno per addestrare il modello di previsione dell’area della stanza.

5.6 Limitazioni

Mentre iPLAN è in grado di gestire alcuni limiti irregolari, non può far fronte a quelli estremamente irregolari. In questo articolo, si è proposto un nuovo modello generativo human-in-the-loop iPLAN per imparare il design professionale in modo stage-to-stage rispettando i principi di progettazione. Pur essendo in grado di generare in modo completamente automatizzato, iPLAN

consente interazioni ravvicinate con gli esseri umani accettando la guida dell'utente in ogni fase e suggerendo automaticamente possibili progetti di conseguenza. Valutazioni complete su due set di dati di riferimento RPLAN e LIFULL mostrano che iPLAN supera i metodi state-of-the-art, sia quantitativamente che qualitativamente. È importante sottolineare che iPLAN ha mostrato una forte capacità di generalizzazione per attività di progettazione e input di confine non ancora sconosciuti.

Capitolo 6

Sperimentazioni con Validation Set

In questa sezione verranno svolti dei test su [35] in modo da valutarne l'efficienza, la repository github la si può trovare al seguente [link](#).

I dati su cui verranno effettuate le prove si trovano all'interno di "Data/test", dove sono presenti due file con il formato ".mat" il formato di MatLab. Nella repository fornita dai contributors vi si posso distinguere tre cartelle che corrispondono ai modelli disussi durante la relazione:

- room_location: modello utilizzato per la determinazione della posizione delle stanze all'interno del progetto, la quale differisce in base alla posizione in cui si utilizza:
 - Living
 - Location
- room_partition: modello utilizzato per la partizione in stanze del progetto.
- room_type: modello utilizzato per la classificazione dei tipi di stanze.

Tutti i modelli creati verranno estesi da una classe chiamata **BasicModule**.

```
class BasicModule(t.nn.Module):
    def __init__(self):
        super(BasicModule, self).__init__()
        self.name = str(type(self))

    def load_model(self, path):
        self.load_state_dict(t.load(path))

    def save_model(self, epoch=0, is_best=False):
        pth_list = [pth for pth in os.listdir('checkpoints') if re.match(self.name, pth)]
        pth_list = sorted(pth_list, key=lambda x: os.path.getmtime(os.path.join('checkpoints', x)))
        if len(pth_list) >= 10 and pth_list is not None:
            to_delete = 'checkpoints/' + pth_list[0]
            if os.path.exists(to_delete):
                os.remove(to_delete)

        path = f'checkpoints/{self.name}_{epoch}.pth'
        t.save(self.state_dict(), path)

        if is_best:
            path = f'checkpoints/{self.name}_best.pth'
            t.save(self.state_dict(), path)
```

Figura 6.1: Codice di BasicModule.

Questa classe è progettata per gestire il salvataggio e il caricamento di modelli. Il metodo **load_model** accetta un percorso e carica lo stato del modello dal file specificato nel modello stesso. Mentre il metodo **save_model** salva lo stato del modello in un file specifico. Contiene una logica per controllare i checkpoint salvati che consiste nel ricercare i file di checkpoint che corrispondono al nome del modello e tiene i 10 più recenti, eliminando il più vecchio se ce ne sono già 10. Salva anche il checkpoint corrente come il migliore se il flag *is_best* è True.

6.1 Room Location

Come affrontato nel capitolo 3.3 lo scopo di questo modello è la previsione della regione della stanza, come processo step by step per ogni stanza presente nel progetto. Viene fatta la distinzione per due aree Living e Location.

6.1.1 Living

Nel contesto della Living vengono utilizzati due modelli una **ResNet** [6.6] per la classificazione e un **FC** [6.7] che consiste in un layer connesso utilizzato per apprendere e combinare le caratteristiche estratte dai layer precedenti.

Come mostrato nella figura [3.2] senza considerare la fase di embedding. Per il task si utilizzano due modelli, di seguito vi è l'istanziazione dei due modelli. Nel codice [6.1] viene instanziato il classificatore, mentre nel codice [6.2] il layer convulozionale.

Codice 6.1: RESNET

```
1 model = models.model(
2         module_name=opt.
3             module_name,
4         model_name=opt.
5             model_name,
6         input_channel=3,
7         output_channel=2,
8         pretrained= False
9     )
```

Codice 6.2: FC

```
1 input_channel = 512
2 connect = models.connect(
3     module_name=opt.
4         module_name,
5     model_name=opt.
6         model_name,
7     input_channel=
8         input_channel,
9     output_channel=2,
10    reshape=True
11 )
```

Di seguito vi sono i modelli caricati dopo aver effettuato il load del chekcpoint utilizzando la funzione della classe madre **load_model**.

```

ResNet{
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): LeakyReLU(negative_slope=0.01, inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): LeakyReLU(negative_slope=0.01, inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): LeakyReLU(negative_slope=0.01, inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): LeakyReLU(negative_slope=0.01, inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): LeakyReLU(negative_slope=0.01, inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
}

```

Figura 6.2: Caricamento Re-Net.

```

FC{
  (fc): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01, inplace=True)
    (3): Conv2d(512, 2, kernel_size=(1, 1), stride=(1, 1))
    (4): AdaptiveAvgPool2d(output_size=1)
  )
}

```

Figura 6.3: Caricamento di FC.

In questo modello l'input che viene dato al modello sono delle immagini relative alla piantina, con la posizone della porta d'ingresso come in figura 6.4, e una lista dei punti target che indicano la posizione della effettiva stanza.

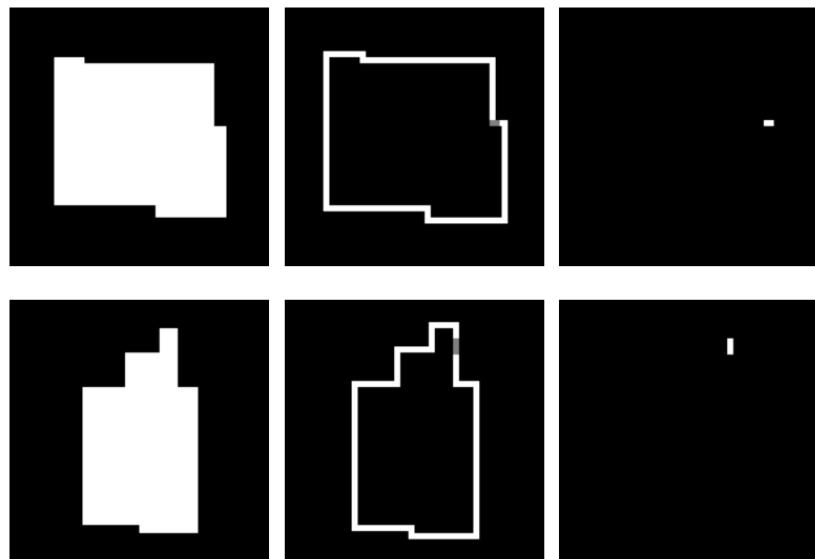


Figura 6.4: Input RESNET Room Location Living.

L'immagine verrà data come input alla resnet che ritornerà la propria classificaizone. La classificazione verrà fornita al connected layer che restituirà il punto in cui il modello posiziona la stanza.

Nella figura sottostante vengono comparate i valori reali, le x , con i valori ottenuti dal modello, le \hat{x} .

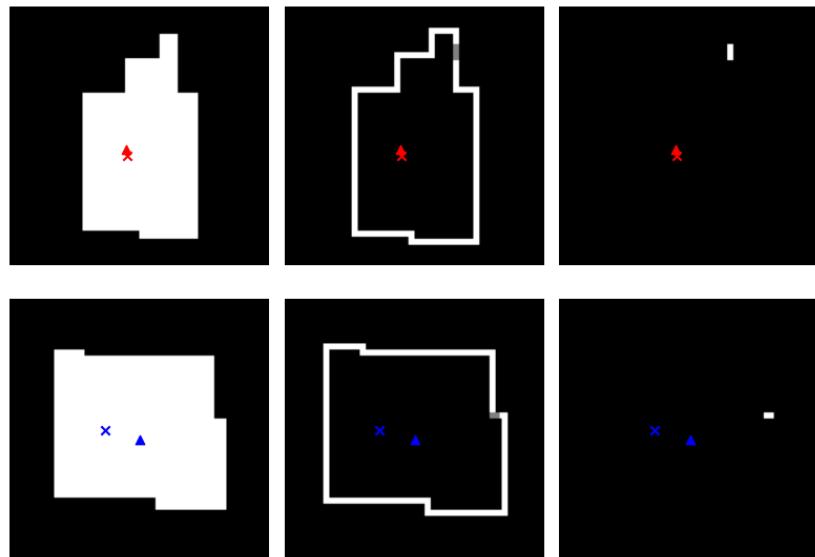


Figura 6.5: Comparazione valori target, x , e valori predetti, \hat{x}

Come si può vedere dal risultato visivo, per il primo input il modello riesce a predire quasi alla perfezione la posizione della stanza, mentre per il secondo input commette un errore abbastanza grande nell'asse delle x .

A causa dell'errore nella seconda predizione l'errore calcolato è molto alto
Distance_error : [12.5, 331.25]

```

class ResNet(BasicModule):
    def __init__(self, name, block, layers, input_channel, output_channel, pretrained=False):
        super(ResNet, self).__init__()
        self.name = name
        self.layers = layers
        self.conv1 = nn.Conv2d(input_channel, 64, 7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu1 = nn.LeakyReLU(inplace=True)
        self.layer0 = nn.MaxPool2d(3, stride=2, padding=1)

        self.layer1 = self._make_layer(block, 64, layers[0], stride=1, dilation=1)
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2, dilation=1)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2, dilation=1)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2, dilation=1)

        initialize_weights(self)

        if pretrained:
            print("load the pretrained model...")
            pretrained_model = './pretrained_model/resnet14-333feca4.pth'
            resnet = torch.load(pretrained_model)
            resnet.load_state_dict(torch.load(pretrained_model))

            self.bn1 = resnet.bn1
            self.layer1 = resnet.layer1
            self.layer2 = resnet.layer2

    def _make_layer(self, block, planes, blocks, stride=1, dilation=1):
        downsample = None
        if stride > 1 or planes != self.inplanes:
            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes*block.expansion, 1, stride=stride, bias=False),
                nn.BatchNorm2d(planes*block.expansion)
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride, dilation=dilation, downsample=downsample))
        self.inplanes = planes*block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes, dilation=dilation))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.layer0(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        return x

```

Figura 6.6: Implementazione della ResNet.

```

class FC(BasicModule):
    def __init__(self, name, input_channel, output_channel, reshape):
        super(FC, self).__init__()
        self.name = name
        self.reshape = reshape
        temp_channel = 512

        fc_type = name.split('_')[-1]
        if fc_type == 'fc1':
            self.fc = nn.Sequential(
                nn.Conv2d(input_channel, temp_channel, 3, padding=1),
                nn.BatchNorm2d(temp_channel),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel, output_channel, 1),
                nn.AdaptiveAvgPool2d(1)
            )

        initialize_weights(self)

    def forward(self, x):
        x = self.fc(x)

        if self.reshape:
            x = x.view(x.size(0), -1)

        return x

```

Figura 6.7: Implementazione di FC.

6.1.2 Location

Nel contesto della Location vengono utilizzati tre modelli una **ResNet** [6.12] per la classificazione, una **UP** [6.13] e un layer di **Embedding** [6.14].

UP è un’architettura di rete neurale convoluzionale (CNN) utilizzata principalmente per il segmentazione delle immagini, ed è caratterizzata dalla sua forma a U. Questa rete è stata introdotta da Olaf Ronneberger, Philipp Fischer e Thomas Brox nel 2015. La U-Net è ampiamente utilizzata in applicazioni di visione artificiale, in particolare per la segmentazione semantica, in cui si desidera assegnare ciascun pixel di un’immagine a una classe specifica. Come mostrato nella figura [3.2] considerando questa volta la fase di embedding.

Di seguito vengono riportate le implementazione dei tre modelli.

Codice 6.3: RESNET

```
1 model = models.model(
2         module_name=opt.
3             module_name,
4             model_name=opt.
5                 module_name,
6                 input_channel=
7                     utils.
8                         num_category+4,
9                         output_channel=
10                            utils.
11                                num_category+3,
12                                pretrained=False
13
14 )
```

Codice 6.4: RESNET

```
1 input_channel = 512
2 connect = models.connect(
3     module_name=opt.
4         module_name,
5         model_name=opt.
6             module_name,
7             input_channel=
8                 input_channel,
9                 output_channel=utils.
10                     num_category+3,
11                     reshape=False
12
13 )
```

Codice 6.5: EMBEDDING

```
1 embedding = models.embedding(
2     module_name=opt.module_name,
3     model_name=opt.model_name,
4     input_channel=13,
5     output_channel=256,
6     reshape=False
7 )
```

Una volta effettuata l'instaziazione dei modelli e il load del checkpoint l'architettura è la seguente.

```
UP:
(aspp1): Sequential(
  (0): Conv2d(768, 512, kernel_size=(3, 3), stride=(1, 1), padding=(6, 6), dilation=(6, 6))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01, inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): LeakyReLU(negative_slope=0.01, inplace=True)
  (6): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
)
(aspp2): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(12, 12), dilation=(12, 12))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01, inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): LeakyReLU(negative_slope=0.01, inplace=True)
  (6): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
)
(aspp3): Sequential(
  (0): Conv2d(768, 512, kernel_size=(3, 3), stride=(1, 1), padding=(18, 18), dilation=(18, 18))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01, inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): LeakyReLU(negative_slope=0.01, inplace=True)
  (6): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
)
(aspp4): Sequential(
  (0): Conv2d(768, 512, kernel_size=(3, 3), stride=(1, 1), padding=(24, 24), dilation=(24, 24))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01, inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): LeakyReLU(negative_slope=0.01, inplace=True)
  (6): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
)
(convf): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2))
  (1): ConvTranspose2d(512, 16, kernel_size=(8, 8), stride=(8, 8))
)
```

Figura 6.8: Architettura UP.

```
Embedding:
  (fc): Sequential(
    (0): Linear(in_features=13, out_features=256, bias=True)
    (1): LeakyReLU(negative_slope=0.01, inplace=True)
    (2): Linear(in_features=256, out_features=512, bias=True)
    (3): LeakyReLU(negative_slope=0.01, inplace=True)
    (4): Linear(in_features=512, out_features=1024, bias=True)
    (5): LeakyReLU(negative_slope=0.01, inplace=True)
  )
  (conv): Sequential(
    (0): Conv2d(4, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01, inplace=True)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.01, inplace=True)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.01, inplace=True)
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.01, inplace=True)
  )
)
```

Figura 6.9: Architettura Embedding.

```

ResNet():
    (conv1): Conv2d(17, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): LeakyReLU(negative_slope=0.01, inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
        (0): BasicBlock(
            (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (1): BasicBlock(
            (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
    )
    (layer2): Sequential(
        (0): BasicBlock(
            (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (downsample): Sequential(
                (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
                (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            )
        )
        (1): BasicBlock(
            (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
    )
    (layer3): Sequential(
        (0): BasicBlock(
            (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
            (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (downsample): Sequential(
                (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            )
        )
        (1): BasicBlock(
            (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
            (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
    )
    (layer4): Sequential(
        (0): BasicBlock(
            (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (downsample): Sequential(
                (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            )
        )
        (1): BasicBlock(
            (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): LeakyReLU(negative_slope=0.01, inplace=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
    )

```

Figura 6.10: Architettura RESNET.



Figura 6.11: Input Room Location Location

Per poter ottenere il risultato avremo di bisogno di tre elementi in input: l'immagine, il room Type e il target che verrà utilizzato per il calcolo dell'efficienza.

L'immagine in input consiste in un tensore con più immagini, come mostrato in figura 6.11, dove le prime due immagini corrispondono al progetto e al boundary della planimetria, mentre tutti i rettangoli mostrano le possibili stanze e l'ordine in cui esse vengono processate e allocate.

Il room Type è un array di 15 posizioni dove tutte sono impostate a 0 ad eccezione della posizione effettiva della stanza dove vi sarà 1.

Nel codice i contributors hanno identificato i tipi di stanze come **room_label**.

Codice 6.6: Tipi di Stanze

```
1 room_label = [(0,LivingRoom),  
2                 (1,MasterRoom),  
3                 (2,Kitchen),  
4                 (3,Bathroom),  
5                 (4,DiningRoom),  
6                 (5,ChildRoom),  
7                 (6,StudyRoom),  
8                 (7,SecondRoom),  
9                 (8,GuestRoom),  
10                (9,Balcony),  
11                (10,Entrance),  
12                (11,Storage),  
13                (12,Wall-\in),  
14                (13,External),  
15                (14,ExteriorWall),  
16                (15,FrontDoor),  
17                (16,Interior)]
```

L'insieme delle immagini viene fornita alla **resnet**, la quale restituirà la sua predizione. Il roomType viene dato come input al modello **embedding** che restituirà uno score. I risultati dei due modelli vengono concatenati lungo il primo asse tramite la funzione *cat* di pytorch. Il tensore ottenuto dalla concatenazione viene fornito al modello **connect**, il risultato di connect viene elaborato da una **SoftMax** ottenendo l'output finale.

Per valutare la bontà del modello si distingue in categorie predette e categorie specifiche predette. L'**accuracy** totale è uguale a 1.99 .

Num target Category	648
Num predict Category	666
Num predict Category right	634
Percentage preected specific category	97%
Percentage wrong category (false positive)	0.05%
Percentage rigth category (true positive)	98%

Tabella 6.1: Valutazione predizione Categorie

Num target specific category	162
Num predict specific category	180
Num predict specific category right	149
Percentage preected specific category	90%
Percentage wrong specific category (false positive)	19%
Percentage rigth specific category (true positive)	92%

Tabella 6.2: Valutazione predizione Categorie specifiche

Come si può vedere dai valori riportati nella tabella il modello predice in entrambi i casi più valori rispetto ai valori target, il motivo potrebbe essere la versione non ottimale dei modelli caricati i quali portano ad avere maggior rumore.

Nonostante questa problematica in entrambi i casi i modelli riescono a raggiungere ottime prestazioni per predizioni corrette, il 98% per la predizione delle categorie e il 92% nella predizione di categorie specifiche.

```

class ResNetBasicModule(nn.Module):
    def __init__(self, name, block, layers, input_channel, output_channel, pretrained=False, pretrained_path = None):
        super(ResNet, self).__init__()
        self._name = name
        self.inplanes = 64
        self.conv1 = nn.Conv2d(input_channel, 64, 7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.layers = _make_layer(block, 64, layers[0], stride=1, dilation=1)
        self.layers = _make_layer(block, 128, layers[1], stride=2, dilation=1)
        self.layers = _make_layer(block, 256, layers[2], stride=2, dilation=1)
        self.layers = _make_layer(block, 512, layers[3], stride=1, dilation=1)

        initialize_weights(self)

    if pretrained:
        print("Load the pretrained model...")
        if '34' in self.name :
            pretrained_model = '../pretrained_model/resnet34-333f7ec4.pth'
        else:
            pretrained_model = '../pretrained_model/resnet18-5c106cde.pth'

    if '38' in self.name:
        pretrained_model = '../pretrained_model/resnet18-5c106cde.pth'
    if pretrained_path is None:
        pretrained_model = pretrained_path
    else:
        load_state_dict(nn.Module.load_state_dict, pretrained_model)

    self.bn1 = resnet.bn1
    self.layers = resnet.layer1
    self.layer2 = resnet.layer2

    _make_layer(self, block, planes, blocks, stride=1, dilation=1):
        downsample = None
        if stride > 1 or self.inplanes != planes*block.expansion:
            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes*block.expansion, 1, stride=stride, bias=False),
                nn.BatchNorm2d(planes*block.expansion)
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride=stride, dilation=dilation, downsample=downsample))
        self.inplanes = planes * block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes, dilation=dilation))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        return x

    class UP(nn.Module):
        def __init__(self, name, input_channel, output_channel, reshape):
            super(UP, self).__init__()
            self.name = name
            self._name = name
            self.reshape = reshape
            temp_channel = 1024
            input_channel = input_channel + 256

            up_type = name.split('_')[1]
            if up_type == 'up1':
                dilation = [16, 16, 16, 24]
                self.asp1 = nn.Sequential(
                    nn.Conv2d(input_channel, temp_channel//2, 3, padding=dilations[0], dilation=dilations[0]),
                    nn.BatchNorm2d(temp_channel//2),
                    nn.LeakyReLU(inplace=True),
                    nn.Conv2d(temp_channel//2, temp_channel//2, 1),
                    nn.BatchNorm2d(temp_channel//2),
                    nn.LeakyReLU(inplace=True),
                    nn.Conv2d(temp_channel//2, temp_channel//2, 1)
                )
            self.asp2 = nn.Sequential(
                nn.Conv2d(input_channel, temp_channel//2, 3, padding=dilations[1], dilation=dilations[1]),
                nn.BatchNorm2d(temp_channel//2),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel//2, temp_channel//2, 1),
                nn.BatchNorm2d(temp_channel//2),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel//2, temp_channel//2, 1)
            )
            self.asp3 = nn.Sequential(
                nn.Conv2d(input_channel, temp_channel//2, 3, padding=dilations[2], dilation=dilations[2]),
                nn.BatchNorm2d(temp_channel//2),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel//2, temp_channel//2, 1),
                nn.BatchNorm2d(temp_channel//2),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel//2, temp_channel//2, 1)
            )
            self.asp4 = nn.Sequential(
                nn.Conv2d(input_channel, temp_channel//2, 3, padding=dilations[3], dilation=dilations[3]),
                nn.BatchNorm2d(temp_channel//2),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel//2, temp_channel//2, 1),
                nn.BatchNorm2d(temp_channel//2),
                nn.LeakyReLU(inplace=True),
                nn.Conv2d(temp_channel//2, temp_channel//2, 1)
            )
            self.convT = nn.Sequential(
                nn.Conv2d(temp_channel//2, temp_channel//2, 3, padding=2, dilation=2),
                nn.ConvTranspose2d(temp_channel//2, output_channel, 9, stride=2)
            )

        initialize_weights(self)

        def forward(self, x):
            x1 = self.asp1(x)
            x2 = self.asp2(x)
            x3 = self.asp3(x)
            x4 = self.asp4(x)
            x = x1 + x2 + x3 + x4
            x = self.convT(x)

            if self.reshape:
                x = x.view(x.size(0), x.size(2), -1)

            return x

```

Figura 6.12: Implementazione ReSNet Location.

Figura 6.13: Implementazione UP.

```

class Embedding(BasicModule):
    def __init__(self, name, input_channel, output_channel, reshape= False):
        super(Embedding, self).__init__()
        self.name = name

        self.fc = nn.Sequential(
            nn.Linear(input_channel, 256),
            nn.LeakyReLU(inplace=True),
            nn.Linear(256, 512),
            nn.LeakyReLU(inplace=True),
            nn.Linear(512, 1024),
            nn.LeakyReLU(inplace=True)
        )
        self.conv = nn.Sequential(
            nn.Conv2d(4, 32, 3, 1, 1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(32, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(64, 128, 3, 1, 1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(128, output_channel, 3, 1, 1),
            nn.BatchNorm2d(output_channel),
            nn.LeakyReLU(inplace=True)
        )

        initialize_weights(self)

    def forward(self, x):
        x = self.fc(x)
        x = x.view(-1, 4, 16, 16)
        x = self.conv(x)
        return x

```

Figura 6.14: Implementazione Embedding.

6.2 Room Partition

A causa di incompatibilità con GoogleColab non è stato possibile svolgere delle sperimentazioni su Room Partition per la fase di test del validation set. Per questo motivo viene riportata solo l'implementazione del modello utilizzato per la Room Partition e la sua instanziazione senza la visione dei risultati, in questo caso il modello non estende la classe BasicModule.

Codice 6.7: Instanziazione

```
1     fp_rnn = models.FloorPlanRNN(opt, log_file)
```

```
class FloorPlanRNN(Module):
    def __init__(self, opt, log_file):
        super(FloorPlanRNN, self).__init__()
        # specify the training losses you want to print out.
        self.loss_names = ['G_GAN', 'D_L1', 'D_real', 'D_fake', 'D', 'GP']
        self.model_names = ['G', 'D']

        self.netG = None
        self.netD = None
        self.optimizer_G = None
        self.optimizer_D = None
        self.device = None
        self.lambda_L1 = opt.lambda_L1
        self.step_size = opt.step_size
        self.gpu_ids = opt.gpu_ids
        self.log_file = log_file

        self.print_networks(log_file)

        # configure nodes
        self.netG = BottleneckDecoder().to(self.device)
        self.netD = MultilayerDiscriminator().to(self.device)
        self.netB = NlayerDiscriminator().to(self.device)
        self.print_networks(log_file)

        # criterion and optimizer
        utils.log_log_file('Building criterion and optimizer...')
        lr = opt.lr
        self.optimizers = []
        self.optimizer_G = optim.Adam(self.netG.parameters(), lr=lr, betas=(0.9, 0.999), weight_decay=opt.weight_decay)
        self.optimizer_D = optim.Adam(self.netD.parameters(), lr=lr, betas=(0.9, 0.999), weight_decay=opt.weight_decay)
        self.optimizer_B = optim.Adam(self.netB.parameters(), lr=lr, betas=(0.9, 0.999), weight_decay=opt.weight_decay)
        self.optimizers.append(self.optimizer_G)
        self.optimizers.append(self.optimizer_D)
        self.optimizers.append(self.optimizer_B)
        self.criterion = nn.L1Loss()
        self.criterion.to(self.device)
        self.loss_fn = nn.SmoothL1Loss(reduction='sum').to(self.device)

        self.set_input(self, input):
            b_imgs, packed_r_centers, packed_r_types, insides, packed_r_boxes, packed_fp_states, _, data_ids = input
            self.data_ids = data_ids
            self.b_imgs = b_imgs.to(self.device)
            self.packed_r_centers = packed_r_centers.to(self.device)
            self.packed_r_types = packed_r_types.to(self.device)
            self.insides = insides.to(self.device)
            self.packed_r_boxes = packed_r_boxes.to(self.device)
            self.fp_states = fp_states.to(self.device)
            self.packed_fp_states = packed_fp_states.to(self.device)

            self.step_size = self.packed_r_centers[1]
            self.sorted_indices = self.packed_r_centers[2]
            self.unsorted_indices = self.packed_r_centers[3]
            self.fp_num = self.b_imgs.size(0)
            self.room_num = self.step_size.sum()
```

Figura 6.15: Implementazione RoomLocation 1/5.

```
def optimize_parameters(self):
    self.forward() # compute fake images: G(A)
    # update D
    self.set_requires_grad(self.netD, True)
    self.set_requires_grad(self.netG, False)
    self.optimizer_D.zero_grad() # set D's gradients to zero
    self.backward_D() # calculate gradients for D
    self.optimizer_D.step() # update D's weights

    # update G
    self.set_requires_grad(self.netD, False)
    self.set_requires_grad(self.netG, True)
    self.optimizer_G.zero_grad() # set G's gradients to zero
    self.backward_G() # calculate gradients for G
    self.optimizer_G.step() # update G's weights

def forward(self):
    packed_r_centers = self.packed_r_centers.data
    packed_r_types = self.packed_r_types.data
    states = self.b_imgs
    insides = self.insides
    step_sizes = self.step_size
    num = len(step_sizes)

    pred_r_boxes_list = []
    pred_fp_states_list = []

    ind = 0
    for n in range(num):
        sz = step_sizes[n]
        states = states[:, :, :, :, :]
        r_centers = packed_r_centers[:, ind + sz, :, :, :]
        r_types = packed_r_types[:, ind + sz, :, :, :]
        insides = insides[:, ind + sz, :, :, :]
        output, states = self.sub_step(states, r_centers, r_types, insides)
        pred_r_boxes_list.append(output)
        pred_fp_states_list.append(states)
        ind += t.sum(step_sizes[:n + 1])

    self.pred_packed_r_boxes = t.cat(pred_r_boxes_list, 0) # tensor
    self.pred_packed_fp_states = t.cat(pred_fp_states_list, 0) # tensor
```

Figura 6.16: Implementazione RoomLocation 2/5.

```

def sub_step(states, centers, types, insides):
    types = types.view(1, 1, 1, 1).expand_as(states)
    centers = centers.view(1, 1, 1, 1).expand_as(states)
    norm_states = self.normalize(states)
    input = t.cat([norm_states, centers, types / (types.num_category - 1)], dim=1)
    input = input.contiguous()
    if not self.training:
        input = Variable(input, requires_grad=True)

    r_boxes = self.netG(input)
    r_masks = self.netR(r_boxes)
    r_center = self.netC(r_boxes)
    new_states = states * (1 - r_masks) + r_masks * (types)
    return r_boxes, new_states

def backward_(self):
    """Calculate GAN loss for the discriminator"""
    # Takes state backprop to the generator by detaching fake_B
    fp_state_real = self.normalize(self.netD_fp.state.data)
    pred_state_fake = self.netD(fp_state_real)
    pred_state_fp = self.netD(fp.state.data)
    self.loss_D_fp = self.criterionGAN(pred_state_fp, False)

    # real
    fp_state_real = self.normalize(self.netD_packed_fp.state.data)
    fp_state_real = fp_state_real.unsqueeze(1)
    gt_state_real = self.netD(gt.state_real)
    self.loss_D_p_real = self.criterionGAN(gt.state_real, True)

    # combine loss and calculate gradients
    self.loss_GP = self.gradient_penalty(self.netD, fp.state_real, fp.state_fake, self.device)
    self.loss_D = self.loss_D_fp + self.loss_D_p_real + self.loss_GP
    self.loss_D.backward,retain_graph=True

    def backward_(self):
        """Calculate GAN loss and L1 loss for the generator"""
        # First, G(A) should fake the discriminator
        fp_state_fake = self.normalize(self.netD_packed_fp.state.data)
        pred_state_fake = self.netD(fp.state.data)
        self.loss_GAN = self.criterionGAN(pred_state_fake, True)

        # Second, G(A) = B
        self.loss_G_L1 = self.criterionL1(self.netD_packed_r_boxes.state, gt_packed_r_boxes.data) / self.room_nus

        # combine loss and calculate gradients
        self.loss_G = self.loss_G_GAN + self.loss_G_L1 + self.lmbda_L1
        self.loss_G.backward()

        def evaluate(self):
            with torch.no_grad():
                self.forward() # compute fake images: G(A)

            # #discriminator loss
            fp_state_fake = self.normalize(self.netD_packed_fp.state.data)
            pred_state_fake = self.netD(fp.state.data)
            self.loss_D_p_fake = self.criterionGAN(pred_state_fake, False)

            # #generator loss
            fp_state_real = self.normalize(self.netD_packed_fp.state.data)
            fp_state_real = fp_state_real.unsqueeze(1)
            gt_state_real = self.netD(gt.state_real)
            self.loss_D_p_real = self.criterionGAN(gt.state_real, True)

            # combine loss and calculate gradients
            self.loss_GP = self.gradient_penalty(self.netD, fp.state_real, fp.state_fake, self.device, phrases='test')
            self.loss_G = self.loss_D_p_fake + self.loss_D_real + self.loss_GP

            # #generator loss
            self.loss_G_WN = self.criterionGAN(self.netD_packed_r_boxes.state, self.netD_packed_fp.state.data) / self.room_nus
            self.loss_G = self.loss_G_GAN + self.loss_G_WN + self.loss_G_L1 + self.loss_GP

        def trainInit():
            self.netD.train()
            self.netD_p.train()
            self.netD_fp.train()
            self.netC.train()
            self.netR.train()

        def evalInit():
            self.netD.eval()
            self.netD_p.eval()
            self.netD_fp.eval()
            self.netC.eval()
            self.netR.eval()

        def set_requires_grad_(net, requires_grad=False):
            for param in net.parameters():
                param.requires_grad_ = requires_grad

        def normalize(self, data):
            data = t.div(data, util.total_num_category - 1)
            data = t.clamp(data, 0.5)
            data = t.div(data, 0.5)
            return data

```

Figura 6.17: Implementazione RoomLocation 3/5.

Figura 6.18: Implementazione RoomLocation 4/5.

```

def get_current_losses(self):
    """Return training losses / errors. train.py will print out these errors on console, and save them to a file"""
    errors_ret = OrderedDict()
    for name in self.loss_names:
        if isinstance(name, str):
            errors_ret[name] = float(getattr(self, 'loss_-' + name))
        errors_ret[name] = errors_ret[name]
    return errors_ret

def print_networks(self, log_file):
    for name in self.model_names:
        if isinstance(name, str):
            net = getattr(self, 'net' + name)
            num_params = 0
            for param in net.parameters():
                num_params += param.numel()
            if True:
                print(net)
            utils.log(log_file, '[Network %s] Total number of parameters : %.3f M' % (name, num_params / 1e6))
    utils.log(log_file, '-----')

def save_networks(self, epoch):
    for name in self.model_names:
        if isinstance(name, str):
            save_filename = '%s_net_%s.pth' % (name, epoch)
            save_path = os.path.join(self.save_dir, save_filename)
            net = getattr(self, 'net' + name)
            t.save(net.state_dict(), save_path)

def load_networks(self, opt, log_file):
    if opt.load_netG_path:
        utils.log(log_file, 'Loading the model: {}.'.format(opt.load_netG_path))
        self.netG.load_state_dict(t.load(opt.load_netG_path))
    if opt.load_netD_path:
        utils.log(log_file, 'Loading the model: {}.'.format(opt.load_netD_path))
        self.netD.load_state_dict(t.load(opt.load_netD_path))

```

Figura 6.19: Implementazione RoomLocation 5/5.

6.3 Room Type

Come discusso nel capitolo 3.2, il topic di questo modello consiste nell'utilizzo di un BCVAE basato su VAE, variational autoencoder. Nella figura 6.20 viene mostrata l'implementazione per il CVAE.

Una volta istanziato il modello:

Codice 6.8: Instanziazione CVAE

```
1 cvae = models.cvae(  
2     modul_name=opt.module_name,  
3     model_name=opt.model_name,  
4     input_dim=opt.max_room_num,  
5     hidden_dim1=128,  
6     hidden_dim2=64,  
7     z_dim=opt.noise_dim  
8 )
```

Dove il nome del modello e il nome del modulo servono per identificare alla classe madre quale classificatore istnaziare, l'input dim indica il massimo numero di sample che può elaboare, i tre valori finali indicano le dimensioni della rete. Tramite la funzione della classe madre, **load_model** viene caricato il modello pretrainato. Ottenendo la figura 6.21.

```

1  class CVAE(BasicModule):
2      def __init__(self, name, input_dim, hidden_dim1, hidden_dim2, z_dim):
3          super(CVAE, self).__init__()
4          self.name = name
5          # encoder part
6          self.fc1 = nn.Linear(input_dim + 64, hidden_dim1)
7          self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
8          self.fc31 = nn.Linear(hidden_dim2, z_dim)
9          self.fc32 = nn.Linear(hidden_dim2, z_dim)
10         # decoder part
11         self.fc4 = nn.Linear(z_dim + 64, 96)
12         self.fc5 = nn.Linear(64, 64)
13         self.fc6 = nn.Linear(64, input_dim)
14
15         self.relu = nn.ReLU()
16
17         self.embed_feat = [16, 16, 32, 32, 16, 16]
18         self.embed = make_layers(self.embed_feat, in_channels=1)
19
20         initialize_weights(self)
21
22     def encoder(self, x):
23         h = self.relu(self.fc1(x))
24         h = self.relu(self.fc2(h))
25         return self.fc31(h), self.fc32(h) # mu, log_var
26
27     def sampling(self, mu, log_var):
28         std = torch.exp(0.5 * log_var)
29         eps = torch.randn_like(std)
30         return eps.mul_(std).add_(mu) # return z sample
31
32     def decoder(self, z):
33         h = self.relu(self.fc4(z))
34         h = self.relu(self.fc5(h))
35         return torch.sigmoid(self.fc6(h))
36
37     def embed_feat(self, img):
38         emb = self.embedding
39         return emb
40
41     def forward(self, x, img):
42         emb = self.embedding()
43         i_2 = torch.cat([x, emb], 1)
44         mu, log_var = self.encoder(i_2)
45         z = self.sampling(mu, log_var)
46         o_2 = torch.cat([z, emb], 1)
47         return self.decoder(o_2), mu, log_var
48
49
50     def cvae(modul_name, model_name, **kwargs):
51         name = modul_name + '_' + model_name
52         return CVAE(name, **kwargs)

```

CVAE:

- (fc1): Linear(in_features=83, out_features=128, bias=True)
- (fc2): Linear(in_features=128, out_features=64, bias=True)
- (fc31): Linear(in_features=64, out_features=32, bias=True)
- (fc32): Linear(in_features=64, out_features=32, bias=True)
- (fc4): Linear(in_features=96, out_features=96, bias=True)
- (fc5): Linear(in_features=96, out_features=64, bias=True)
- (fc6): Linear(in_features=64, out_features=16, bias=True)
- (relu): ReLU()
- (embed): Sequential
- (0): Conv2d(1, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
- (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
- (2): ReLU(inplace=True)
- (3): Conv2d(16, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
- (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
- (5): ReLU(inplace=True)
- (6): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
- (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
- (8): ReLU(inplace=True)
- (9): Conv2d(32, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
- (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
- (11): ReLU(inplace=True)
- (12): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
- (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
- (14): ReLU(inplace=True)
- (15): Conv2d(64, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
- (16): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
- (17): ReLU(inplace=True)
- (18): Flatten(start_dim=1, end_dim=-1)

Figura 6.20: Implementazione di CVAE.

Figura 6.21: Caricamento di CVAE.

Come discusso nel capitolo specifico il modello prenderà come input un insieme di tipologie delle stanze e l'immagine su cui calcolare la predizione. Per questo modello il tipo di input è della seguente tipologia:

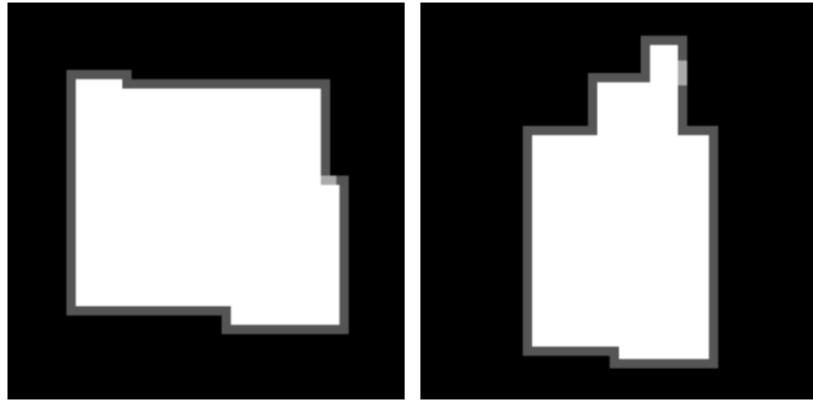


Figura 6.22: Input CVAE RoomType.

Per la valutazione di questo modello abbiamo a disposizione tre metriche che riportiamo nella tabella sottostante con i valori ottenuti nella fase di sperimentazione.

Loss	5.173709869384766
Binary entropy loss	1.7538410425186157
Kullback-Leibler divergence	6.839737892150879

Tabella 6.3: Valori di perdita.

Il valore **Loss** è ottenuto sommando la **BCE** con la **KLD** e dividendola per due. Per capire i risultati ecco le definizioni delle loss utilizzate:

- Binary Cross-Entropy Loss: La binary cross-entropy loss è comunemente utilizzata come funzione di perdita per problemi di classificazione binaria, tipicamente in output di modelli di machine learning che producono una previsione binaria (0 o 1).
 - *Quando è migliore*: È migliore quando si avvicina a zero, indicando che le previsioni del modello sono molto vicine ai valori reali.
 - *Quando è peggiore*: È peggiore quando si avvicina a valori più alti, poiché significa che le previsioni del modello si discostano maggiormente dai valori reali.

- Kullback-Leibler Divergence (KL Divergence): La KL divergenza è utilizzata per misurare la differenza tra due distribuzioni di probabilità. Nei modelli di apprendimento automatico, spesso è usata nei problemi di ottimizzazione in cui si desidera ridurre la differenza tra due distribuzioni, ad esempio tra la distribuzione reale e quella prevista dal modello.
 - *Quando è migliore*: È migliore quando si avvicina a zero, indicando che le distribuzioni sono simili.
 - *Quando è peggiore*: È peggiore quando si avvicina a valori più alti, indicando una grande differenza tra le distribuzioni, il che significa che le previsioni del modello sono molto diverse dalla distribuzione reale.

Quindi valori bassi sono desiderabili in entrambi i casi, poiché indicano che il modello si adatta bene ai dati di addestramento o che la differenza tra le distribuzioni è minima. Viceversa, valori più alti indicano una peggiore adattabilità o una maggiore discrepanza tra le distribuzioni.

Nel nostro esperimento possiamo notare che i valori che ci vengono restituiti dal modello non sono ottimali ma indicano un alto tasso di errore. Questo potrebbe essere causato da molti fattori ad esempio, degli input non adatti oppure ad una versione non efficiente di modello caricato, poichè si utilizzano i checkpoint forniti dai contributors.

Capitolo 7

Prove Sui Modelli

In questo capitolo si vedranno delle prove e i relativi risultati sui modelli precedentemente esposti. Le prove consisteranno nella simulazione di utilizzo dei vari modelli cercando di emulare il processo di generazione di iPLAN da input a output come mostrato nel paper in figura 3.1.

7.1 RoomType

Il modello prende in input lo stesso input che abbiamo visto per il roomType nel capitolo sulla sperimentazione, im. 6.22.

Data l'immagine in input il modello restituisce le seguenti informazioni:

- *Name*: Nome del file.
- *gt_rTypes*: Ground Truth del tipo.
- *gt_rBoxes* : Ground Truth delle boxe delle stanze.
- *Boundary*: Bordi delle stanze ottenute dal modello.
- *rTypes*: Identifica il numero e il tipo di stanze generate dal modello, viene ottenuto da RoomType.
- *rBoxes*: Identifica la boxe della stanza predetta, verrà predetto da RoomPartition.
- *rCenters*: Identifica il centro della stanza predetta, verrà predetto da LocationRoom.

Ecco un esempio

Codice 7.1: Instanziazione CVAE

```
1 name: 0
2 gt_rTypes: [0 1 2 3 7 9]
3 gt_rBoxes:
4 [[ 14 55 101 92]
5 [ 80 36 110 62]
6 [ 26 57 41 72]
7 [ 65 36 78 53]
8 [ 43 36 63 62]
9 [103 64 114 92]]
10 Boundary:
11 [[ 19 82 1 1]
12 [ 27 82 1 1]
13 [ 43 82 0 0]
14 [ 43 92 1 0]
15 [114 92 2 0]
16 [114 64 3 0]
17 [110 64 2 0]
18 [110 36 3 0]
19 [ 43 36 0 0]
20 [ 43 57 3 0]
21 [ 26 57 0 0]
22 [ 26 74 3 0]
23 [ 14 74 0 0]
24 [ 14 82 1 0]]
25 rTypes: []
26 rBoxes: []
27 rCenters: []
```

Per ottenere questi valori i contributors utilizzano una funzione chiamata `getList`, dove viene fornito come parametro il path alla cartella contenente i file ”.mat”, per poi effettuare il controllo nella riga 10 il quale consiste nell’accertarsi che il campo **rTypes** sia vuoto poichè sarà il modello a inserire i valori in quel campo. La prima immagine mostra il codice originale mentre la seconda il codice modificato per l’esecuzione. Per caricare il file ”.mat” viene utilizzata la funzione `loadmat` della libreria **scipy.io**. In entrambi i codici scritti di seguito nelle righe 9 il valore data finale è tra apici [’data’].

Codice 7.2: getLits RoomType codice originale.

```
1 def getList(path):
2     mat_list = os.listdir(path)
3     mat_list.sort()
4     temp_list = []
5     for name in mat_list[100:200]:
6         mat_path = os.path.join(path, name)
```

```

7     data = sio.loadmat(mat_path,
8                         squeeze_me=True,
9                         struct_as_record=False)[data]
10    if type(data.rTypes) is not np.ndarray or len(data.
11             rTypes) == 0:
12        temp_list.append(name)
13
13 return temp_list

```

Codice 7.3: getLits RoomType modificato.

```

1 def getList(path):
2     mat_list = os.listdir(path)
3     mat_list.sort()
4     temp_list = []
5     for name in mat_list[:]:
6         mat_path = os.path.join(path, name)
7         data = sio.loadmat(mat_path,
8                             squeeze_me=True,
9                             struct_as_record=False)[data]
10        if type(data.rTypes) is not np.ndarray or len(data.
11                  rTypes) == 0:
12            temp_list.append(name)
13
13 return temp_list

```

In questo modello si deve scegliere il numero massimo di stanze che si possono generare, inserendo in un array di 13 elementi il numero massimo di generazione seguendo le posizioni indicate nel room_label 6.6. Il modello viene istanziato nel seguente modo:

Codice 7.4: Istanziazione modello RoomType.

```

1 max_room_per_type = [1, 2, 1, 2, 1, 1, 1, 3, 1, 3, 1, 1, 1]
2 max_room_num = np.sum(np.array(max_room_per_type))
3 noise_dim = 32
4 load_cvae_path = /content/drive/MyDrive/Colab_Notebooks/
      Deep_Learning/Progetto/Model/room_type/roomtype_cvae_150.
      pth # e' una stringa quindi tra apici
5
6 cvae = models.cvae(
7     modul_name= roomtype, # e' una stringa quindi tra apici
8     model_name= cvae, # e' una stringa quindi tra apici
9     input_dim=max_room_num,
10    hidden_dim1=128,
11    hidden_dim2=64,
12    z_dim=noise_dim
13 )

```

7.1.1 Valutazione Risultati

Una volta che il modello prende in input i dati come visto nella sezione precedente, il modello dovrà predire i tipi di stanze e riempirà il campo **rTypes** di [7.1](#). Di seguito sono forniti i risultati per i due input:

Codice 7.5: Risultati RoomType.

```
1 Ground Truth immagine 0: [0 1 2 3 7 9]
2 Valori predetti immagine 0: [0 1 2 3 7 9]
3
4 Ground Truth immagine 1: [0 1 2 3 5 7 9]
5 Valori predetti immagine 1: [0 1 2 3 7 9]
```

Come si può osservare nella seconda planimetria è stata predette una stanza in meno che corrisponde all'assenza di una *ChildRoom*.

7.2 Locating Room

Anche per questo modello i dati vengono caricati tramite la funzione getList dove in questo caso il controllo da fare è quello che i **rCenters** siano vuoti, poichè sarà questo modello a riempire quel campo con la sua predizione.

Codice 7.6: getLits RoomLocation codice originale.

```
1 def getList(path):
2     mat_list = os.listdir(path)
3     mat_list.sort()
4     temp_list = []
5     for name in mat_list[100:200]:
6         mat_path = os.path.join(path, name)
7         data = sio.loadmat(mat_path, squeeze_me=True,
8                             struct_as_record=False)[data] # data e' una
9                             stringa
10        if len(data.rCenters) == 0:
11            temp_list.append(name)
12    return temp_list
```

Codice 7.7: getLits RoomLocation modificato.

```
1 def getList(path):
2     mat_list = os.listdir(path)
3     mat_list.sort()
4     temp_list = []
5     for name in mat_list[:]:
6         mat_path = os.path.join(path, name)
7         data = sio.loadmat(mat_path, squeeze_me=True,
8                             struct_as_record=False)[data] # data e' una stringa
8         if len(data.rCenters) == 0:
9             temp_list.append(name)
10    return temp_list
```

Per il funzionamento del modello vengono utilizzate altre funzioni che vengono riportate alla fine del capitolo seguente.

```
1 living_model = living.model(
2     module_name= living, #e' una stringa
3     model_name= resnet18_fc1, #e' una stringa
4     input_channel=3,
5     output_channel=2,
6     pretrained=False,
7 )
8 living_connect = living.connect(
9     module_name= living, #e' una stringa
10    model_name= resnet18_fc1, #e' una stringa
11    input_channel=512,
12    output_channel=2,
13    reshape=True
14 )
15
16 location_model = location.model(
17     module_name= location, #e' una stringa
18     model_name= resnet18_up1, #e' una stringa
19     input_channel=13 + 4,
20     output_channel=13 + 3,
21     pretrained=True,
22     pretrained_path= path_model_roomLocation_resnet
23 )
24
25 location_connect = location.connect(
26     module_name= location, #e' una stringa
27     model_name= resnet18_up1, #e' una stringa
28     input_channel=512,
29     output_channel=13 + 3,
30     reshape=False
31 )
32 location_embedding = location.embedding(
33     module_name= location, #e' una stringa
34     model_name= resnet18_up1, #e' una stringa
35     input_channel=13,
36     output_channel=256,
37     reshape=False
38 )
```

7.2.1 Valutazione Risultati

Una volta dato l'input al modello come mostrato nel capitolo precedente il risultato finale per i due input è il seguente:

```
'name': '0',
'gt_rTypes': array([0, 1, 2, 3, 7, 9]),
'gt_rBoxes': array([[ 14,   55,  101,   92],
                   [ 80,   36,  110,   62],
                   [ 26,   57,   41,   72],
                   [ 65,   36,   78,   53],
                   [ 43,   36,   63,   62],
                   [103,   64,  114,   92]]),
'Boundary': array([[ 19,   82,     1,     1],
                   [ 27,   82,     1,     1],
                   [ 43,   82,     0,     0],
                   [ 43,   92,     1,     0],
                   [114,   92,     2,     0],
                   [114,   64,     3,     0],
                   [110,   64,     2,     0],
                   [110,   36,     3,     0],
                   [ 43,   36,     0,     0],
                   [ 43,   57,     3,     0],
                   [ 26,   57,     0,     0],
                   [ 26,   74,     3,     0],
                   [ 14,   74,     0,     0],
                   [ 14,   82,     1,     0]]),
'rTypes': array([0]),
'rBoxes': array([], dtype=int64),
'rCenters': array([[58, 71]])

'name': '1',
'gt_rTypes': array([0, 1, 2, 3, 5, 7, 9]),
'gt_rBoxes': array([[ 28,   22,   65,  106],
                   [ 67,   72,   97,  106],
                   [ 25,   22,   42,   36],
                   [ 28,   38,   36,   50],
                   [ 67,   52,   97,   70],
                   [ 67,   22,   97,   50],
                   [ 99,   72,  103,  106]]),
'Boundary': array([[ 59,  101,     0,     1],
                   [ 59,  106,     1,     0],
```

```
[103, 106,    2,    0],  
[103,  72,    3,    0],  
[ 97,  72,    2,    0],  
[ 97,  22,    3,    0],  
[ 25,  22,    0,    0],  
[ 25,  36,    1,    0],  
[ 28,  36,    0,    0],  
[ 28, 100,    1,    0],  
[ 59, 100,    0,    0]]),  
'rTypes': array([0]),  
'rBoxes': array([], dtype=int64),  
'rCenters': array([[64, 69]])
```

Come si può vedere nel file caricato da *getList* entrambi gli input hanno una sola stanza predetta in **rTypes**, che per entrambe le planimetrie è una *LivingRoom*, all'interno di **rCenter** si trova il centrodice della stanza che consiste in una combinazione di due elementi cioè le coordinate **y** e **x**.

7.2.2 Ulteriori funzioni

```

def get_rcenters(fp, location_model, location_connect, location_embedding, mask_size=4):
    pred_rCenters = []
    pred_rTypes = []

    living_node = fp.living_node
    h, w = living_node['centroid']
    pred_rCenters.append(np.array([int(h), int(w)]))
    pred_rTypes.append(0)

    input_location = fp.get_composite_location(num_extra_channels=0)
    continue_rTypes = fp.continue_rTypes.squeeze(0)
    input_location = input_location.unsqueeze(0).cuda()

    room_num = continue_rTypes.shape[0]

    iter = 0
    flag = True
    while room_num > 0:
        update_id = np.zeros(room_num, dtype=bool)
        iter = iter + 1

        np.random.shuffle(continue_rTypes)
        with torch.no_grad():
            for n in range(room_num):
                score_model = location_model(input_location)
                r_t = torch.LongTensor(continue_rTypes[n+1]).cuda()
                one_hot_label = torch.nn.functional.one_hot(r_t, num_classes=13)
                score_embedding = location_embedding(one_hot_label.float())
                score_temp = torch.cat([score_model, score_embedding], 1)
                score_connect = location_connect(score_temp)

                score_softmax = torch.softmax(score_connect, dim=1)
                output = score_softmax.cpu().numpy()
                predict = np.argmax(output, axis=1)[0]

                center = find_center(predict, r_t)
                h, v = center

                if h == 0 or w == 0 or input_location[0, 0, h, w] == 0:
                    continue

                flag = check_rcenters(center, pred_rCenters)

                if flag:
                    pred_rCenters.append(np.array([h, w]))
                    pred_rTypes.append(r_t.cpu().numpy().reshape(1)[0])
                    update_id[n] = 1
                    min_h = maxh - mask_size, 0
                    max_h = minh + mask_size, 128 - 1
                    min_w = maxw - mask_size, 0
                    max_w = minw + mask_size, 128 - 1
                    input_location[0, r_t + 4, min_h:max_h + 1, min_w:max_w + 1] = 1.0
                    input_location[0, 3, :, :] = input_location[0, 4:, :, :].sum(0)

                    iter = 0
                    continue_rTypes = continue_rTypes[update_id]
                    room_num = continue_rTypes.shape[0]

    if iter > 5:
        if room_num == 0: # 0: all room centers are located, 1: one room missing, 2: two room missing
            flag = True
        else:
            flag = False
        break

    return pred_rCenters, pred_rTypes, flag

```

Figura 7.1: GetCenter 1/2.

Figura 7.2: GetCenter 2/2.

```

def find_center(predict, r_t, mask_size=4):
    index_point = []
    index = np.where(predict == r_t.cpu().numpy())
    for ind in range(index[0].shape[0]):
        index_point.append((index[0][ind], index[1][ind]))

    num_point = 0
    if len(index_point) > 0:
        min_h = np.min(index[0])
        min_w = np.min(index[1])
        max_h = np.max(index[0])
        max_w = np.max(index[1])

        if max_h - min_h + 1 <= 2 * mask_size and max_w - min_w + 1 <= 2 * mask_size:
            predict_h = (min_h + max_h) // 2
            predict_w = (min_w + max_w) // 2
            num_point = len(index_point)
        else:
            predict_h, predict_w = index_point[0]
            for point in index_point:
                new_num_point = 0
                for other_point in index_point:
                    if abs(other_point[0] - point[0]) <= mask_size and abs(
                        other_point[1] - point[1]) <= mask_size:
                        new_num_point += 1
                if new_num_point > num_point:
                    predict_h, predict_w = point
                    num_point = new_num_point
    if num_point > 0:
        return np.array([predict_h, predict_w])
    else:
        return np.array([0, 0])

def check_rcenters(center, rCenters):
    flag = True
    center = np.array(center)
    for i in range(len(rCenters)):
        temp = np.array(rCenters[i])
        dist = np.linalg.norm(center - temp)
        if dist < 9:
            flag = False
    return flag

```

Figura 7.3: FindCenter.

Figura 7.4: check Center.

7.3 Room Partition

Come per i modelli precedenti i dati vengono caricati con la funzione `getList`. Prima di esporre le differenze tra le due, che in questo contesto saranno molto evidenti, bisogna considerare altre modifiche effettuate per far funzionare il modello, inoltre come per `LocatingRoom` alla fine verranno mostrate delle funzioni supplementarie utilizzate per l'ottenimento dell'output.

7.3.1 Modifiche

Tra le funzioni necessarie per l'instanziazione del modello e il suo funzionamento vi è la funzione **`LossFun`**, la quale secondo i contributors bisognerebbe richiamarla nel seguente modo:

```
from room_partition.models.loss_layer import LossFun
```

Il problema consiste che nella versione rilasciata **`loss_layer`** che è un file ".py" richiamabile attraverso la definizione di un file `__init__.py` non si trova nella directory **`models`** di **`room_partition`**, per questo motivo tramite l'ausilio della libreria **`shutil`** la directory viene spostata nel luogo indicato dal file `__init__.py` per poi richiamare **`LossFun`**.

```
1 import shutil
2 cartella_origine = /content/iPLAN-Interactive-and-Procedural-
    Layout-Planning/room_partition/utils #sono stringhe
3 cartella_destinazione = /content/iPLAN-Interactive-and-
    Procedural-Layout-Planning/room_partition/models #sono
        stringhe
4 shutil.move(cartella_origine, cartella_destinazione)
5
6 from room_partition.models.loss_layer import LossFun
```

7.3.1.1 Parametri Input

Di default per questo modello i contributors faceva si che i file fossero passati tramite la libreria **argparse**, quindi che il file dovesse essere eseguito tramite linea di comando. I parametri sono mostrati nella figura seguente:

```
parser = argparse.ArgumentParser()
parser.add_argument('--load_netG_path', type=str, default='/content/drive/MyDrive/Colab Notebooks/Deep_Learning/Progetto/Model/room_p
    help='path of net G')
parser.add_argument('--max_iter', type=int, default=200, help='maximum of iteration')
parser.add_argument('--gpu_ids', nargs='+', type=int, default=[0], help='id of GPU')
parser.add_argument('--lr', type=float, default=10)
parser.add_argument('--coverage', type=float, default=1)
parser.add_argument('--inside', type=float, default=0.5)
parser.add_argument('--mutex', type=float, default=0)
parser.add_argument('--root', type=str, default='/content/iPLAN-Interactive-and-Procedural-Layout-Planning/data', help='id of GPU')
opt = parser.parse_args()
```

Figura 7.5: Parametri input parse Room Partition.

Per ovviare a questo in modo da fare delle prove direttamente tramite il codice è stato definito un dizionario

```
opt = {
    'load_netG_path': '/content/drive/MyDrive/Colab Notebooks/Deep_Learning/Progetto/Model/room_partition/G_net_210.pth',
    'max_iter': 200,
    'gpu_ids': [0],
    'lr': 10,
    'coverage': 1,
    'inside': 0.5,
    'mutex': 0,
    'root': '/content/iPLAN-Interactive-and-Procedural-Layout-Planning/data'
}
```

Figura 7.6: Definizione Dizionario.

Utilizzando un dizionario il codice non funziona perché bisognerebbe cambiare il codice sorgente nei modelli poichè per accedere ad un elemento nel dizionario bisogna inserire la chiave mentre nel codice si considerano le chiavi come attributi di un oggetto. Per ovviare a questo problema si definisce la classe **AttrDict**, che permette di accedere agli elementi del dizionario considerano le chiavi di esso come un attributo.

```
class AttrDict(dict):
    """Classe wrapper per consentire l'accesso attributivo agli elementi del dizionario."""

    def __getattr__(self, name):
        return self[name]

    def __setattr__(self, name, value):
        self[name] = value
```

Figura 7.7: Classe Wrapper.

In modo da instanziare il dizionario nel seguente modo.

```
opt = AttrDict({
    'load_netG_path': '/content/drive/MyDrive/Colab_Notebooks/Deep_Learning/Progetto/Model/room_partition/G_net_210.pth',
    'max_iter': 200,
    'gpu_ids': [0],
    'lr': 10,
    'coverage': 1,
    'inside': 0.5,
    'mutex': 0,
    'root': '/content/iPLAN-Interactive-and-Procedural-Layout-Planning/data'
})
```

Figura 7.8: Definizione del dizionario tramite Wrapper.

7.3.1.2 Istanziazione Modello

Nel codice fornito dai contributors l’istanziazione del modello viene fatta nel seguente modo:

```
1 from room_partition import models
2
3 fp_rnn = models.FloorPlanRNNTTest(opt)
```

Ma non esiste nessun file ”.py” o modello che si chiama **FloorPlanRNNTest**. Si suppone che i contributors abbiano applicato delle modifiche al codice prima del rilascio e che non abbiano aggiornato il codice come opportuno. Per ovviare a questo problema, ci si è resi conto che all’interno della directory *models* contenuta in *room_partition* vi è un file **floorplan_rnn_test.py** il codice è stato modificato di conseguenza

```
1 from models.floorplan_rnn_test import FloorPlanRNN
2
3 fp_rnn = FloorPlanRNN(opt)
```

7.3.1.3 Caricamento File Pickle

Come mostrato nei paragrafi precedenti è possibile caricare i modelli già allenati e cambiare il path per prendere i suddetti pesi.
Ma FloorPlannRNN al suo interno istanzia due modelli **RenderNet** e **BoundingBoxNet**.

```
class RendererNet(nn.Module):
    def __init__(self):
        super(RendererNet, self).__init__()
        self.fc1 = (nn.Linear(4, 512))
        self.fc2 = (nn.Linear(512, 1024))
        self.conv1 = (nn.Conv2d(4, 32, 3, 1, 1))
        self.conv2 = (nn.Conv2d(32, 32, 3, 1, 1))
        self.conv3 = (nn.Conv2d(8, 16, 3, 1, 1))
        self.conv4 = (nn.Conv2d(16, 16, 3, 1, 1))
        self.conv5 = (nn.Conv2d(4, 8, 3, 1, 1))
        self.conv6 = (nn.Conv2d(8, 4, 3, 1, 1))
        self.relu = (nn.ReLU())
        self.pixel_shuffle = nn.PixelShuffle(2)
        self._initialize_weights()

    def _initialize_weights(self):
        print('load the pretrained Mask generation model')
        pretrain_model = "./weights/renderer.pkl"
        model_dict = torch.load(pretrain_model)
        self.load_state_dict(model_dict)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = x.view(-1, 4, 16, 16)
        x = self.relu(self.conv1(x))
        x = self.pixel_shuffle(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.pixel_shuffle(self.conv4(x))
        x = self.relu(self.conv5(x))
        x = self.pixel_shuffle(self.conv6(x))
        x = 50 * x
        x = torch.sigmoid(x)

        return x.view(-1, 1, 128, 128)

class BoundingBoxNet(nn.Module):
    def __init__(self, pretrain=False):
        super(BoundingBoxNet, self).__init__()
        self.seen = 0
        self.frontend_feat = [32, 64, 128, 64, 32]
        self.output_feat = [64, 4]
        self.frontend = make_layers(self.frontend_feat)
        self.output_layer = make_output_layer(self.output_feat, 32 * 4 * 4)
        self._initialize_weights(pretrain)

    def forward(self, x):
        x = self.frontend(x)
        x = self.output_layer(x)
        return x

    def _initialize_weights(self, pretrain=True):
        if pretrain:
            print('load the pretrained CSNet generation model')
            pretrain_model = "./data/pretrain.pth.tar"
            if not os.path.exists(pretrain_model):
                pretrain_model = './room_partition/weights/renderer.pkl'
            model_dict = torch.load(pretrain_model)
            self.load_state_dict(model_dict)

        else:
            for m in self.modules():
                if isinstance(m, nn.Conv2d):
                    nn.init.normal_(m.weight, std=0.01)
                    if m.bias is not None:
                        nn.init.constant_(m.bias, 0)
                elif isinstance(m, nn.Linear):
                    nn.init.normal_(m.weight, std=0.01)
                    if m.bias is not None:
                        nn.init.constant_(m.bias, 0)
```

Figura 7.9: Codice RenderNet.

Figura 7.10: Codice Bounding-BoxNet.

Entrambi caricano un modello preallenato nel formato ”.pkl”, nel file sorgente che si trova in ”/room_partition/models/net.py” nei righi 22 e 63. I path sono stati sostituiti direttamente nel file sorgente con il path destinazione corretto.

7.3.1.4 Modifica GetList

La funzione originale presentata dai contributors nella repository github è:

```
1 def getList(path):
2     mat_list = os.listdir(path)
3     mat_list.sort()
4     temp_list = []
5     for name in mat_list[100:200]:
6         mat_path = os.path.join(path, name)
7         data = sio.loadmat(mat_path, squeeze_me=True,
8                             struct_as_record=False)[data] # [data] e' tra
9                             apici
10        if len(data.rBoxes) == 0 and len(data.rCenters) !=0:
11            temp_list.append(name)
12    return temp_list
```

Dove la funzione effettua due controlli:

- `len(data.rBoxes) == 0`: che controlla che rBoxes sia vuoto poichè sarà compito del modello riempire il campo;
- `len(data.rCenters) !=0`: che serve per controllare che il campo contenente i centroidi non sia vuoto, poichè si useranno i centroidi per calcolare alla fine le bounding box.

Il problema consiste nel fatto che i dati forniti dai contributorsi *0.mat* e *1.mat* sono sprovvisti dei valori dei centroidi.

Per questo motivo la funzione è stata modificata nel seguente modo:

```
def getList(path):
    mat_list = os.listdir(path)
    mat_list.sort()
    temp_list = []
    for name in mat_list[:]:
        mat_path = os.path.join(path, name)
        data = sio.loadmat(mat_path, squeeze_me=True, struct_as_record=False)['data']
        data.rTypes = 1
        if name == "0.mat":
            center = []
            center.append([58,71])
            data.rCenters = center
        else:
            center = []
            center.append([64,69])
            data.rCenters = center
        if len(data.rBoxes) == 0 and len(data.rCenters) !=0:
            temp_list.append(name)
    return temp_list
```

Figura 7.11: Definizione funzione getList RoomPartition.

In quest funzione quello che si fa è di raggirare il problema inserendo manualmente i valori dei centroidi ottenuti precedentemente dall'esecuzione di **Locatin Room**.

7.3.1.5 Generazione Dataset per il modello

A causa delle modifiche riportate il getList non è stato possibile caricare il modello come fatto dai contributors perché altrimenti si sarebbero utilizzati i file ".mat" privi dei valori dei centroidi. Per questo motivo sono state implementate due funzioni per l'utilizzo diretto del file **data** e per la reimplementazione della funzione **get_input** per far sì che funzionasse con i valori data.

```
def getMath(path, i):
    mat_list = os.listdir(path)
    mat_list.sort()
    temp_list = []
    for name in mat_list[:]:
        mat_path = os.path.join(path, name)
        data = sio.loadmat(mat_path, squeeze_me=True, struct_as_record=False)[‘data’]
        data.rTypes = np.array([1])
        if name == “0.mat”:
            center = []
            center.append([58, 71])
            data.rCenters = np.array(center)
            if i == 0: return data
        else:
            center = []
            center.append([64, 69])
            data.rCenters = np.array(center)
            if i == 1: return data
```

Figura 7.12: Definizione funzione getMath RoomPartition.

Invece di tornare il path del dato restituisce l'oggetto **data**.

```
def get_input(self):
    ind = self.rTypes == 0
    self.continue_rTypes = self.rTypes[~ind]
    self.continue_rCenters = self.rCenters[~ind]
    rCenters = []
    for i in range(len(self.continue_rTypes)):
        r_C = np.zeros((128, 128))
        C_x, C_y = continue_rCenters[i]
        r_C[(C_x - 3) * 8 + 2, C_y - 3:(C_y + 2)] = 1
        rCenters.append(r_C)
    rCenters = np.stack(rCenters, 0)
    rCenters = torch.FloatTensor(rCenters)
    return self.img, rCenters, torch.from_numpy(self.continue_rTypes), self.inside_mask, self.exterior_boundary
```

Figura 7.13: Codice GetInput originale.

```
def get_input(data):
    img = fp.init_input(fp.exterior_boundary)
    h, w = img.shape
    img = np.zeros((h, w))
    inside_img = np.zeros((h, w))
    inside_img[118:128, 118:128] = 1
    inside_img < 13] = 1
    inside_mask = torch.from_numpy(inside_img)
    img = torch.from_numpy(img)

    ind = data.rTypes == 0
    ind = np.array(ind)
    continue_rTypes = data.rTypes[~ind]
    continue_rCenters = data.rCenters[~ind]

    rCenters = []
    for i in range(len(continue_rTypes)):
        r_C = np.zeros((128, 128))
        C_x, C_y = continue_rCenters[i]
        r_C[(C_x - 1):(C_x + 2), (C_y - 1):(C_y + 2)] = 1
        rCenters.append(r_C)

    rCenters = np.stack(rCenters, 0)
    rCenters = torch.FloatTensor(rCenters)

    return img, rCenters, torch.from_numpy(continue_rTypes), inside_mask, data.Boundary
```

Figura 7.14: Codice GetInput modificato.

Dove il codice originale è una funzione della classe **FloorPlane**.

Con l'ausilio delle due funzioni implementate i dati vengono estrapolati nel seguente modo:

```
1 img, rCenters, rTypes, inside, boundary = get_input(getMath(
    data_root, i ))
```

7.3.1.6 Modifica update rBoxes

Quando vengono salvati i nuovi rBoxes viene utilizzata la funzione **update_rBoxes**, ma i boxes contenuti sono scalati nel range [0,1], per questo motivo quando viene utilizzata la funzione **to_dict** per convertire il risultato in un dizionario i valori vengono salvati con il tipo **int**, quindi troncati. Per far sì che i risultati siano visibili nel file data finale viene modificato l'updatate nel seguente modo:

```
1 fp.update_rBoxes(pred_rBoxes.detach().cpu().numpy() * 127)
```

In questo modo i valori rBoxes saranno rimappati nel range della size delle immagini e si potranno utilizzare per avere un confronto con il GT. In questo modo gli **rBoxes** risultanti saranno del tipo:

```
'rBoxes': array([[ 35,   77,   61, 110]])
```

7.3.2 Valutazione Risultati

Alla fine il modello restituisce il file Data completamente riempito e due immagini.

```
'name': '0',
'gt_rTypes': array([0, 1, 2, 3, 7, 9]),
'gt_rBoxes': array([[ 14,   55, 101,   92],
                   [ 80,   36, 110,   62],
                   [ 26,   57,  41,   72],
                   [ 65,   36,  78,   53],
                   [ 43,   36,  63,   62],
                   [103,   64, 114,   92]]),
'Boundary': array([[ 19,   82,     1,     1],
                   [ 27,   82,     1,     1],
                   [ 43,   82,     0,     0],
                   [ 43,   92,     1,     0],
                   [114,   92,     2,     0],
                   [114,   64,     3,     0],
                   [110,   64,     2,     0],
                   [110,   36,     3,     0],
                   [ 43,   36,     0,     0],
                   [ 43,   57,     3,     0],
                   [ 26,   57,     0,     0],
                   [ 26,   74,     3,     0],
                   [ 14,   74,     0,     0],
                   [ 14,   82,     1,     0]]),
'rTypes': array([0], dtype=int64),
'rBoxes': array([[ 35,   77,   61, 110]]),
'rCenters': array([[]], dtype=int64)
'rCenters': array([[58,  71]], dtype=int64)
```

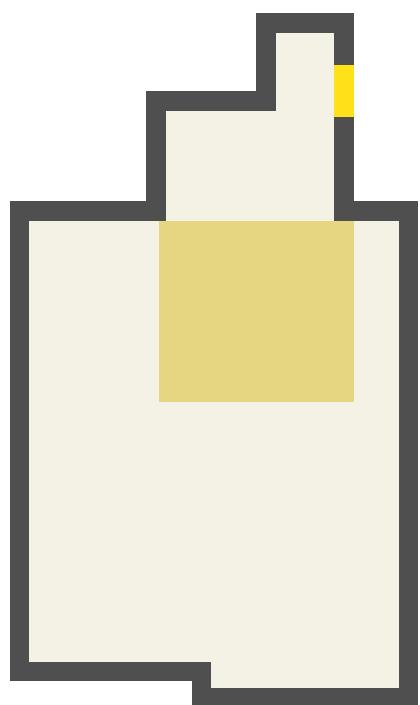


Figura 7.15: Output data0.

```
'name': '1',
'gt_rTypes': array([0, 1, 2, 3, 5, 7, 9]),
'gt_rBoxes': array([[ 28,   22,   65, 106],
                   [ 67,   72,   97, 106],
                   [ 25,   22,   42,  36],
                   [ 28,   38,   36,  50],
                   [ 67,   52,   97,  70],
                   [ 67,   22,   97,  50],
                   [ 99,   72, 103, 106]]),
'Boundary': array([[ 59, 101,     0,     1],
                   [ 59, 106,     1,     0],
                   [103, 106,     2,     0],
                   [103,  72,     3,     0],
                   [ 97,  72,     2,     0],
                   [ 97,  22,     3,     0],
                   [ 25,  22,     0,     0],
                   [ 25,  36,     1,     0],
                   [ 28,  36,     0,     0],
                   [ 28, 100,     1,     0],
                   [ 59, 100,     0,     0]]),
'rTypes': array([0], dtype=int64),
'rBoxes': array([[ 21,   65,   47,  97]]),
'rCenters': array([[64,  69]], dtype=int64)
```

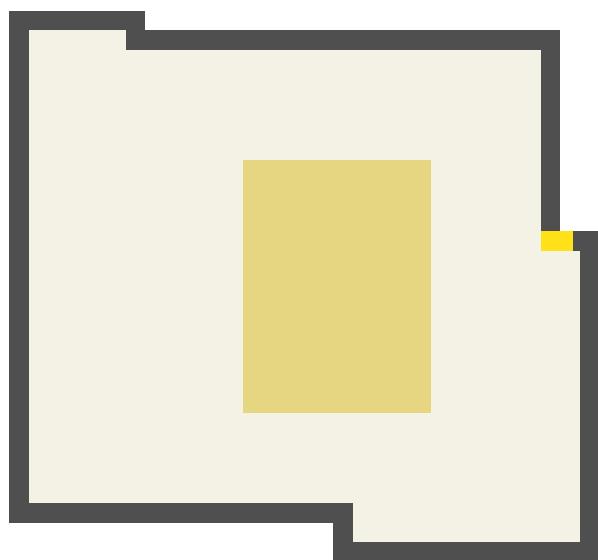


Figura 7.16: Out data1.

In entrambi i casi le stanze di cui si vuole ricavare il bounding boxe sono LivingRoom.

7.3.3 Ulteriori funzioni

Figura 7.17: Codice ObtainLiving.

```

def get_image(img, rBoxes, rTypes, inside, living_mask=None):
    if not living_mask:
        img[living_mask==1]=0
    for r_i in range(len(rTypes)):
        r_box = rBoxes[r_i, :1+2*len(rBox)]
        r_box = r_box.long()
        r_type = rTypes[r_i]
        mask = torch.zeros(inside.size())
        mask[r_box[1]:r_box[3]+1, r_box[0]:r_box[2]+1] = 1
        mask = mask * inside
        img = img * (1 - mask) + mask * r_type
    return img

```

Figura 7.18: Codice GetImage.

```

def fine_tuning(pred_rBoxes, boundary, living_mask, criterion, opt):
    lr = opt.lr
    pred_rBoxes = torch.autograd.Variable(pred_rBoxes, requires_grad=True)
    loss = criterion(pred_rBoxes, boundary, living_mask, opt)
    loss.backward()
    grad = pred_rBoxes.grad
    pred_rBoxes = pred_rBoxes - lr * grad
    return pred_rBoxes

```

Figura 7.19: Codice FineTurning.

```
def get_ratio(layout):
    spare_area = (layout == 16).sum()
    inside_area = ((layout < 13).sum() + spare_area
    ratio = spare_area / inside_area
    return ratio
```

Figura 7.20: Codice GetRatio.

Capitolo 8

Esecuzione Intera Pipeline

In questo capitolo si mostrano i risultati dell'esecuzione dell'intera pipeline. Invece di utilizzare nell'esecuzione dei modelli i file ".mat" del test dopo ogni esecuzione i file corrispondenti sono stati salvati per poi essere utilizzati come input per il modello successivo.

Dopo l'esecuzione di **Room Type** sull'input [6.22](#), il modello ha generato le seguenti stanze:

```
1 [0 1 2 3 7 9]
2 [0 1 2 3 7 7]
```

Si può notare che a differenza dei risultati precedenti vengono generate due stanze dello stesso tipo.

Tramite l'ausilio di *scipy.io* sono stati salvati i dati nuovi tramite la funzione *savemat*.

L'esecuzione di **Locatin Room** con i dati precedenti ha generato i seguenti centroidi:

```
1 [[ 58   71]
2   [ 93   48]
3   [ 33   65]
4   [ 72   43]
5   [ 53   47]
6   [108   78] ]
7
8 [[64  69]
9  [79  33]
10 [40  69]
11 [36  52]
12 [81  59]
13 [39  32]]
```

Nonostante in questa fase non sia ancora possibile valutare la bontà dei risultati è importante notare che il numero dei valori coincide con il numero

delle stanze predette.

Anche se nativamente i contributor non forniscono la possibilità di vedere l'immagine in questo step possiamo combinare i valori che abbiamo per avere un quadro generale



Figura 8.1: Stanze data0.

Possiamo faro lo stesso per data1.

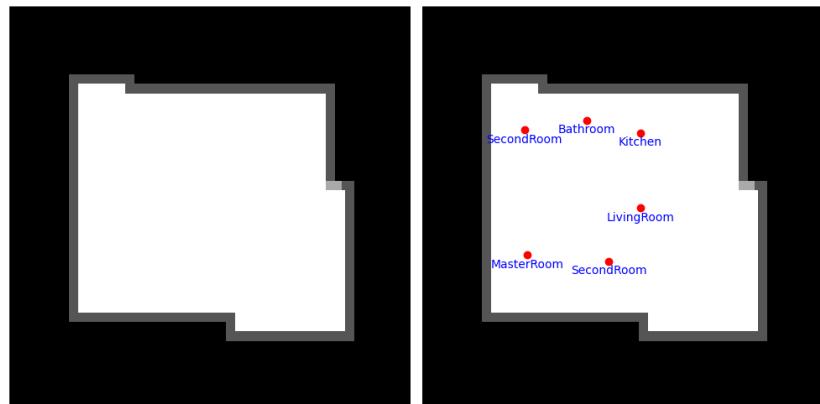


Figura 8.2: Stanze data1.

Eseguendo infine **Room Partition** con il fine data finale si ottengono i seguenti risultati:

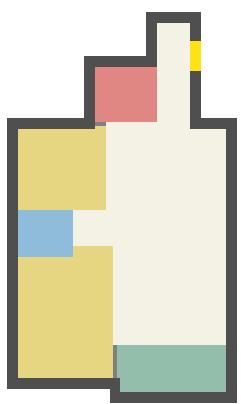


Figura 8.3: Risultato finale data0.

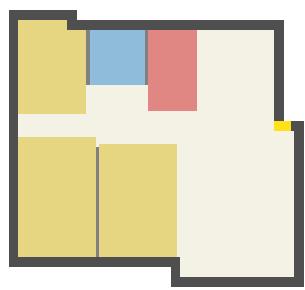


Figura 8.4: Risultato finale data1.

Di seguito sono riportati i file data finali:

```
1 name: 0
2 gt_rTypes: [0 1 2 3 7 9]
3 gt_rBoxes: [[ 14 55 101 92]
4   [ 80 36 110 62]
5   [ 26 57 41 72]
6   [ 65 36 78 53]
7   [ 43 36 63 62]
8   [103 64 114 92]]
9 Boundary: [[ 19 82 1 1]
10  [ 27 82 1 1]
11  [ 43 82 0 0]
12  [ 43 92 1 0]
13  [114 92 2 0]
14  [114 64 3 0]
15  [110 64 2 0]
16  [110 36 3 0]
17  [ 43 36 0 0]
18  [ 43 57 3 0]
19  [ 26 57 0 0]
20  [ 26 74 3 0]
21  [ 14 74 0 0]
22  [ 14 82 1 0]]
23 rTypes: [0 1 2 3 7 9]
24 rBoxes: [[ 35, 77, 61, 110],
25   [ 56, 25, 73, 40],
26   [ 36, 65, 50, 78],
27   [ 36, 42, 59, 65],
28   [ 63, 102, 92, 114]]
29 rCenters: [[ 58 71]
30   [ 93 48]
31   [ 33 65]
32   [ 72 43]
33   [ 53 47]
34   [108 78]]
```

Si può fare lo stesso per data1.

```
1 name: 1
2 gt_rTypes: [0 1 2 3 5 7 9]
3 gt_rBoxes: [[ 28 22 65 106]
4   [ 67 72 97 106]
5   [ 25 22 42 36]
6   [ 28 38 36 50]
7   [ 67 52 97 70]
8   [ 67 22 97 50]
9   [ 99 72 103 106]]
10 Boundary: [[ 59 101 0 1]
11   [ 59 106 1 0]
12   [103 106 2 0]]
```

```

13 [ 103 72 3 0]
14 [ 97 72 2 0]
15 [ 97 22 3 0]
16 [ 25 22 0 0]
17 [ 25 36 1 0]
18 [ 28 36 0 0]
19 [ 28 100 1 0]
20 [ 59 100 0 0]]
21 rTypes: [0 1 2 3 7 7]
22 rBoxes: [[ 21, 65, 47, 97],
23 [ 62, 27, 77, 52],
24 [ 40, 27, 57, 42],
25 [ 47, 64, 69, 98],
26 [ 21, 24, 39, 53],
27 [ 70, 96, 106, 103]]
28 rCenters: [[64 69]
29 [79 33]
30 [40 69]
31 [36 52]
32 [81 59]
33 [100 8]]
```

8.0.0.1 Valutazione Boxes

Per effettuare i confronti sulle Boxes il codice è stato rieseguito e ha fornito dei risultati differenti

```
1 {name: 0,
2   gt_rTypes: array([0, 1, 2, 3, 7, 9]),
3   gt_rBoxes: array([[ 14,   55, 101,   92],
4                     [ 80,   36, 110,   62],
5                     [ 26,   57,   41,   72],
6                     [ 65,   36,   78,   53],
7                     [ 43,   36,   63,   62],
8                     [103,   64, 114,   92]]),
9   Boundary: array([[ 19,   82,     1,     1],
10                  [ 27,   82,     1,     1],
11                  [ 43,   82,     0,     0],
12                  [ 43,   92,     1,     0],
13                  [114,   92,     2,     0],
14                  [114,   64,     3,     0],
15                  [110,   64,     2,     0],
16                  [110,   36,     3,     0],
17                  [ 43,   36,     0,     0],
18                  [ 43,   57,     3,     0],
19                  [ 26,   57,     0,     0],
20                  [ 26,   74,     3,     0],
21                  [ 14,   74,     0,     0],
```

```
22     [ 14,    82,     1,     0]]),
23   rTypes: array([0, 1, 2, 3, 7, 9]),
24   rBoxes: array([[ 35,    77,   61, 110],
25                 [ 56,    25,   73,   40],
26                 [ 36,    65,   50,   78],
27                 [ 36,    42,   59,   65],
28                 [ 63,   102,   92, 114]]),
29   rCenters: array([[ 58,    71],
30                     [ 94,    48],
31                     [ 33,    65],
32                     [ 73,    43],
33                     [ 53,    47],
34                     [108,    78]]})
```

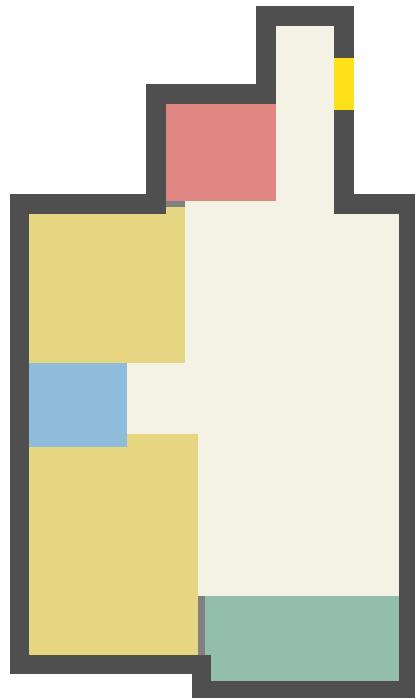


Figura 8.5: Stanza data0.

Adesso abbiamo tutte le informazioni per poter visualizzare i risultati

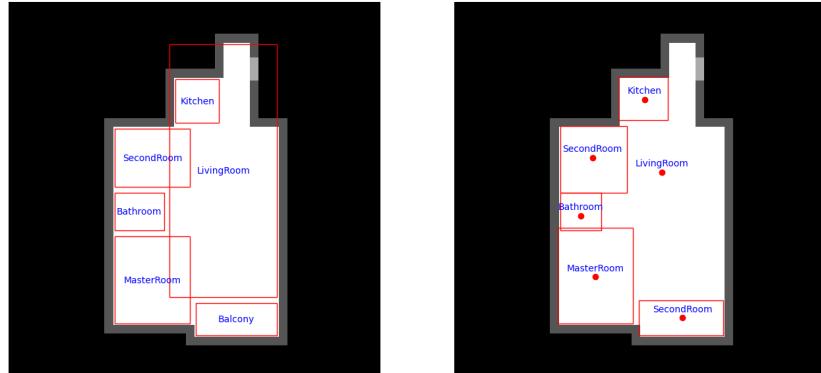


Figura 8.6: Confronto segmentazione.

Dove alla sinistra abbiamo le regione ottenute tramite il GT e alla destra quelle predette dal modello. Nelle **rBoxes** fornite dal modello è assente la partizione della LivingRoom che però possiamo inserire grazie al centro contenuto in **rCenters**. In questa planimetria l'unica differenza consiste nella differenza tra le box e che il balcone è diventato una stanza secondaria.

```
1 {name: 1,
2   gt_rTypes: array([0, 1, 2, 3, 5, 7, 9]),
3   gt_rBoxes: array([[ 28,    22,   65, 106],
4                     [ 67,    72,   97, 106],
5                     [ 25,    22,   42,  36],
6                     [ 28,    38,   36,  50],
7                     [ 67,    52,   97,  70],
8                     [ 67,    22,   97,  50],
9                     [ 99,    72,  103, 106]]),
10  Boundary: array([[ 59, 101, 0, 1],
11                    [ 59, 106, 1, 0],
12                    [103, 106, 2, 0],
13                    [103, 72, 3, 0],
14                    [ 97, 72, 2, 0],
15                    [ 97, 22, 3, 0],
16                    [ 25, 22, 0, 0],
17                    [ 25, 36, 1, 0],
18                    [ 28, 36, 0, 0],
19                    [ 28, 100, 1, 0],
20                    [ 59, 100, 0, 0]]),
21  rTypes: array([0, 1, 2, 3, 7, 7, 9]),
22  rBoxes: array([[ 21,   65,  47,  97],
23                  [ 62,   27,   77,  52],
24                  [ 40,   27,   57,  42],
```

```

25      [ 47,   64,   69,   98],
26      [ 21,   24,   39,   53],
27      [ 70,   96,  106,  103]]),
28  rCenters: array([[ 64,   69],
29                  [ 81,   34],
30                  [ 40,   69],
31                  [ 35,   49],
32                  [ 81,   59],
33                  [ 39,   30],
34                  [100,   88]]}),

```

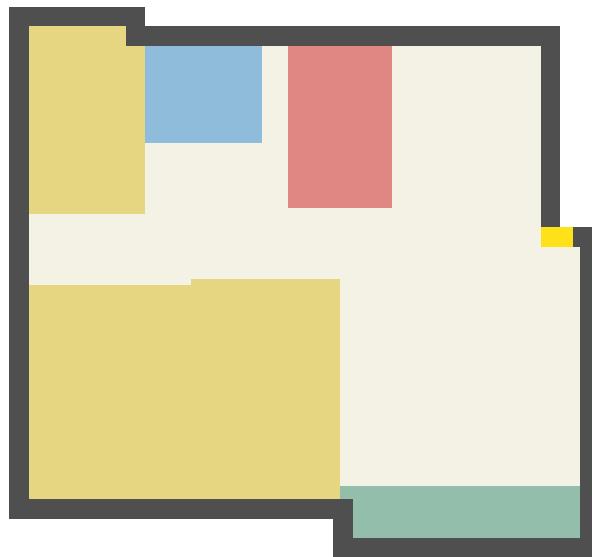


Figura 8.7: Stanza data1.

Come per l'esempio precedente a sinistra sono riportati i valori di GT e sulla destra i valori ottenuti dal modello. A differenza del risultato precedente questa planimetria differisce per le posizioni e la grandezza delle stanze, ma anche in questo esempio non viene fornita la box per la LivingRoom.



Figura 8.8: Confronto segmentazione.

8.0.1 Ulteriori Sperimentazioni

Per valutare il modello su altri set di dati in modo da poter visionare meglio l'efficenza del paper, come indicato dai contributor viene utilizzato **RPLAN-ToolBox** [1]. Il seguente ToolBox permette di ottenere i file ".mat" con le informazioni necessarie per il funzionamento di iPLAN.

Il ToolBoxe prende delle immagini tipo le seguenti, che sono tutte immagini presenti in RPLAN:

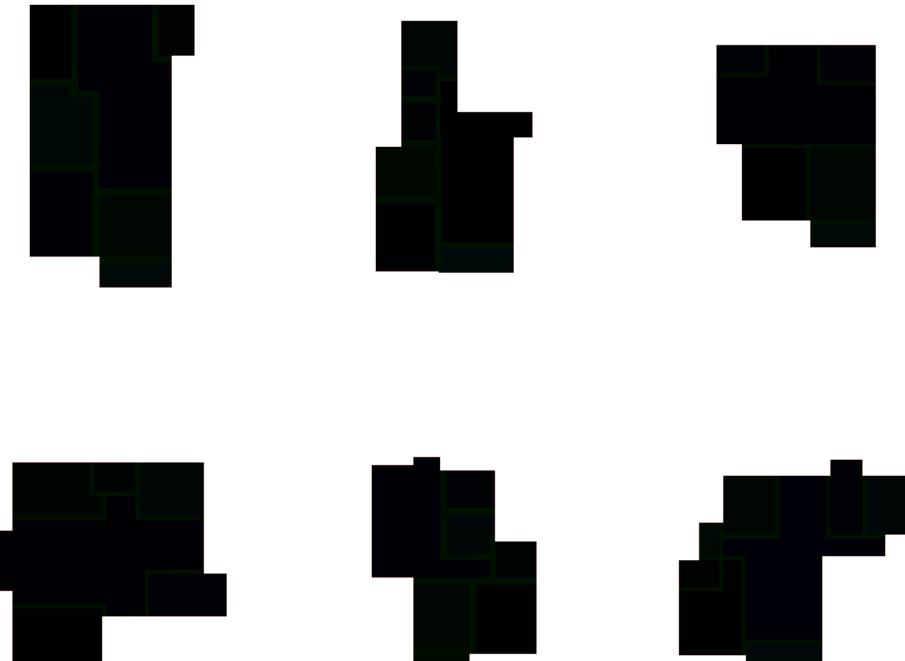


Figura 8.9: Esempio immagini RPLAN.

Per far funzionare il codice sonon state necessarie delle modifiche, la prima consiste nel modificare il file floorplan.py, presente nella cartella rplna, modificando il codice alla riga 208, eliminando un [0], quindi il codice diventa:

```
1 c = stats.mode(self.category[region.coords[:,0],region.coords  
[:,1]])[0]
```

Per rendere il file Data utilizzabile nella pipe iPLAN è stato necessario modificare anche il **to_dict** presente in RPLAN, il quale ha la seguente struttura:

```
def to_dict(self, xyxy=True, dtype=int):
    return {
        'name'      : self.name,
        'types'     : self.rooms[:, -1].astype(dtype),
        'boxes'     : (self.rooms[:, [1, 0, 3, 2]]).astype(dtype)
        if xyxy else self.rooms[:, :4].astype(dtype),
        'boundary'  : self.exterior_boundary[:, [1, 0, 2, 3]].astype(dtype)
        if xyxy else self.exterior_boundary.astype(dtype),
        'edges'     : self.edges.astype(dtype)
    }
```

In questo formato ci sono elementi che non ci interessano come *edges* e le chiavi non hanno lo stesso nome di quello che si aspetterebbe iPLAN per questo motivo la funzione viene modifcata nel modo seguente:

```
def to_dict(self, xyxy=True, dtype=int):
    return {
        'name': self.name,
        'gt_rTypes': self.rooms[:, -1].astype(dtype),
        'gt_rBoxes': (self.rooms[:, [1, 0, 3, 2]]).astype(dtype)
        if xyxy else self.rooms[:, :4].astype(dtype),
        'Boundary': self.exterior_boundary[:, [1, 0, 2, 3]].astype(dtype)
        if xyxy else self.exterior_boundary.astype(dtype),
        'rTypes' : np.array([]).astype(dtype),
        'rBoxes' : np.array([]).astype(dtype),
        'rCenters': np.array([]).astype(dtype)
    }
```

In questo modo possiamo ottenere i file "mat".

Le immagini di RPLAN sono nel formato 256×256 , mentre iPLAN richiede immagini con dimensioni 128×128 . Per risolvere questa discrepanza, i valori sono stati normalizzati nel formato richiesto da iPLAN.

Quindi il *to_dict* finale è:

```
def to_dict(self, xyxy=True, dtype=int):
    return {
        'name': self.name,
        'gt_rTypes': self.rooms[:, -1].astype(dtype),
        'gt_rBoxes': ((self.rooms[:, [1, 0, 3, 2]]))
```

```

.astype(dtype) / 255) * 127
if xyxy else (self.rooms[:, :4].astype(dtype) / 255) * 127,
'Boundary': (self.exterior_boundary[:, [1, 0, 2, 3]]
.astype(dtype) / 255) * 127
if xyxy else (self.exterior_boundary
.astype(dtype) / 255) * 127,
'rTypes': np.array([]).astype(dtype),
'rBoxes': np.array([]).astype(dtype),
'rCenters': np.array([]).astype(dtype)
}

```

In questo modo è stato possibile effettuare altri test sul modello, se per esempio prendiamo le immagini precedenti otteniamo:



Figura 8.10: Esempio immagini RPLAN.

Conclusione

Questo articolo illustra l’impiego di tecniche di deep learning per la generazione automatizzata di planimetrie partizionate, partendo da una piantina e dalle specifiche delle stanze richieste. Nel corso del progetto, sono stati impiegati modelli avanzati, tra cui ResNet, GAN, BCVAE, UP, e reti convoluzionali, evidenziando un approccio completo e sofisticato.

Nonostante le promesse dei contributor, l’intervento umano all’interno della pipeline risulta meno immediato di quanto annunciato. Ad eccezione dell’ inserimento delle tipologie di stanze richieste e del numero massimo di stanze per tipo, non è possibile aggiungere ulteriori informazioni o passaggi esecutivi.

La repository GitHub allegata al paper presenta alcune criticità, come errori nell’annidamento delle cartelle e richiami a porzioni di codice inesistenti in determinati contesti, il che potrebbe essere attribuito a una versione non aggiornata della repository.

Un punto degno di nota è l’assenza della box relativa alla LivingRoom nei file data finali.

In aggiunta, sono state condotte prove supplementari per valutare l’impatto della modifica del numero di stanze al fine di influenzare il risultato finale della pipeline. Tuttavia, è emerso che tale modifica non è stata possibile a causa di un’incongruenza nella versione dei checkpoint. Questo problema potrebbe derivare dal fatto che la versione dei checkpoint fornita dai contributor sia utilizzabile specificamente per i tipi di stanze da loro utilizzati. Questa situazione ha creato difficoltà nell’esecuzione del modello su altri dati e nell’illustrare ulteriori esempi.

Nell’immagine 3.1 del paper, viene delineato il processo dall’input all’output utilizzando i modelli nell’ordine di Room Types (predizione delle stanze e del tipo), Locating Room (predizione dei centroidi delle stanze), e Partitioning Room (partizionamento della planimetria). Tuttavia, è rilevante notare che, nonostante l’utilizzo sequenziale di tali modelli, l’esecuzione pratica si arresta alla generazione dell’immagine di Partitioning Room, senza ottenere l’output completo che includa porte e finestre. Questo risultato, riscontrato

nell'esecuzione del codice fornito dai contributori, solleva delle considerazioni sulla versione attuale del codice proposto.

Per verificare il funzionamento del modello, oltre a rieseguire la pipeline con gli stessi input per osservare la creazione di planimetrie differenti, sono stati adattati altri file presenti nel dataset RPLAN.

Nel corso delle analisi, è importante notare che, nonostante le differenti versioni di generazione presentate nel paper iPLAN, il codice fornito si limita a implementare soltanto una di esse, presubilmente il modello migliore tra quelli prodotti.

Bibliografia

- [1] A&C Black Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 38(6), 2019.
- [2] LIFULL Co. Lifull home's snapshot data of rentals. *arXiv*, 2015.
- [3] Roberto J Rengel. The interior plan: Concepts and exercises. *A&C Black*, 2011.
- [4] Daniel Ritchie, Kai Wang, and Yu an Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models, 2018.
- [5] Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)*, 37(4):70, 2018.
- [6] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019.
- [7] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs, 2018.
- [8] Oron Ashual and Lior Wolf. Specifying object attributes and relations in interactive scene generation, 2019.
- [9] Yikang Li, Tao Ma, Yeqi Bai, Nan Duan, Sining Wei, and Xiaogang Wang. Pastegan: A semi-parametric method to generate image from scene graph, 2019.
- [10] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

- [11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [12] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.
- [13] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data, 2020.
- [14] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.
- [15] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020.
- [16] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2019.
- [17] Stanislas Chaillou. Archigan: Artificial intelligence x architecture. In *Architectural Intelligence*, pages 117–127. Springer, 2020.
- [18] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation, 2018.
- [19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.
- [20] Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. Encoding in style: a stylegan encoder for image-to-image translation, 2021.
- [21] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.
- [22] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation, 2018.
- [23] Peihao Zhu, Rameen Abdal, Yipeng Qin, and Peter Wonka. SEAN: Image synthesis with semantic region-adaptive normalization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020.

- [24] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation, 2020.
- [25] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement networks, 2021.
- [26] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028*, 2017.
- [27] Chun-Yu Sun, Qian-Fang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019.
- [28] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. Graph2plan. *ACM Transactions on Graphics*, 39(4), aug 2020.
- [29] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2195–2203, 2017.
- [30] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [31] Feixiang He, Yanlong Huang, and He Wang. iplan: Interactive and procedural layout planning. *arXiv preprint arXiv:2203.14412*, 2022.