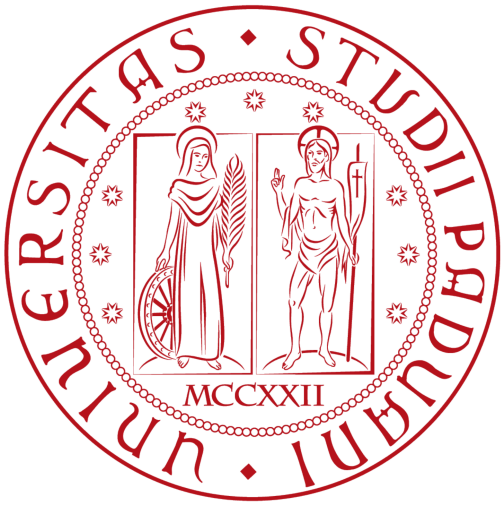
Università degli Studi di Padova
Dipartimento di Matematica "Tullio Levi-Civita"
Laure Magistrale in Informatica



Predizione di traiettorie di pedoni in ambiente indoor

Laureando
Alessio Lazzaron

Relatore
Prof. Lamberto Ballan

Co-Supervisori
Dr. Luigi Filippo Chiara, Dr. Pasquale Coscia

Anno Accademico 2021-2022
25 Febbraio 2022

Cercate di lasciare questo mondo un po' migliore di quanto non l'avete trovato.
- Baden Powell

Abstract

La capacità di prevedere il comportamento di persone in ambienti affollati è fondamentale in molte applicazioni come video-sorveglianza, guida autonoma o percezione automatica della scena. La predizione è un campo di ricerca ampiamente studiato in varie ambiti dell'intelligenza artificiale e negli anni sono state proposte diverse tipologie di algoritmi che si distinguono, oltre dalla metodologia usata, anche dal tipo di dati impiegati.

Principalmente la comunità scientifica si è focalizzata sulla predizione di traiettorie di pedoni o oggetti in movimento in scenari aperti, come strade o piazze. Tuttavia, con la nascita di nuove applicazioni, sia in ambiente domestico che industriale come ad esempio l'utilizzo di robot autonomi per pulire casa o spostare merci, si è portata l'attenzione anche sulla predizione in spazi più ristretti, con un numero potenzialmente maggiore di ostacoli.

L'obiettivo di questo lavoro di tesi è quello di esplorare l'utilizzo del dataset sCREEN, creato in un ambiente indoor, per predire posizioni future. Nella prima parte viene analizzato il dataset, che contiene dati sulle posizioni dei carrelli o cestini della spesa utilizzati dai clienti di un supermercato tedesco, e quali metodi di interpolazione e clusterizzazione sono stati proposti per estratte le traiettorie più significative e non ridondanti. Nella seconda parte, invece, vengono mostrati i modelli, basati anche su Deep Learning, per prevedere future posizioni ad istanti temporali fissati utilizzando come input la parte iniziale della traiettoria sotto osservazione. Infine, viene proposta un'analisi di vari protocolli di valutazione adatti a questa tipologia di previsione.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il computer è una macchina elettronica programmabile in grado di eseguire diverse operazioni da complessi calcoli matematici a elaborazioni dati. A livello storico il primo computer, che segnò il passaggio da macchina calcolatrice a dispositivo programmabile, fu la Macchina analitica di Charles Babbage creata nel 1833. Sebbene questa macchina non fu mai realizzata viene considerata il primo esemplare di computer della storia anche se era formata da ingranaggi e veniva alimentata a vapore. La caratteristica innovativa di questo progetto era l'architettura, molto simile a quella dei computer moderni, infatti prevedeva un input, un output, un unità di memoria e di calcolo. Negli anni il computer divenne un oggetto molto versatile capace di eseguire diverse operazioni velocizzando notevolmente attività svolte fino a prima dall'uomo.

Per svolgere una qualsiasi attività il computer ha bisogno di un algoritmo. Un algoritmo è una sequenza di istruzioni che devono essere eseguite per trasformare un determinato input in output. Per esempio un algoritmo può descrivere ad un computer come ordinare una lista di numeri data una sequenza di input. Tuttavia esistono problemi per i quali non può essere scritto un algoritmo, ad esempio data una casella di posta elettronica classificare quale email è di spam e quale no. In questo caso sebbene si conosca come è fatto l'input e l'output non si conosce il meccanismo di come classificare le mail dato che quello che può essere considerato spam cambia nel tempo e da individuo a individuo. Comunque affinché una mail sia considerata di spam deve avere determinate caratteristiche, che sebbene non sono conosciute, sono condivise con tutte le altre mail di spam, quindi hanno in comune un certo pattern. Al fine di classificare correttamente la posta elettronica sarà necessario individuare una buona approssimazione del pattern in comune tra tutte le mail di spam. Tipicamente questo meccanismo sta alla base del *Machine Learning*.

L'Apprendimento Automatico o Machine Learning è una branca dell'intelligenza artificiale che studia tecniche per sviluppare sistemi in grado di apprendere e migliorare automaticamente attraverso l'esperienza o l'utilizzo dei dati. [56]. Nello specifico con *Intelligenza Artificiale (IA)* si identifica una disciplina informatica che studia teorie, metodologie e tecniche per progettare sistemi hardware e software in grado di fornire ad un elaboratore elettronico prestazioni tali, che ad un osservatore comune, potrebbero sembrare intelligenza umana [46].

L'apprendimento automatico permette di realizzare modelli matematici che ottimizzano un criterio di prestazione attraverso dati di esempio o esperienze passate. In particolare il modello consiste in una serie di parametri e l'apprendimento è l'esecuzione di un programma per determinare i parametri del modello attraverso i dati di addestramento o l'esperienza. [7] Tipicamente nella costruzione di modelli matematici viene utilizzata la teoria statistica perché il compito principale è fare inferenze da un campione di dati.

Esistono diversi paradigmi di apprendimento automatico:

- *Apprendimento Supervisionato*: dato un insieme di esempi classificati, cioè un insieme di tuple contenenti l'esempio e la sua relativa classificazione ovvero $Tr = \{x^i, f(x^i)\}$. L'obbiettivo è apprendere una descrizione generale che incapsuli l'informazione contenuta negli esempi. Tale descrizione deve poter essere usata in modo predittivo ovvero dato un nuovo input $\hat{x}$ predire l'output associato $f(\hat{x})$. In questo paradigma di apprendimento si assume che un esperto fornisca la supervisione, cioè i valori della funzione f per le istanze x^i dell'insieme di apprendimento Tr. [47] Esempi di task di apprendimento supervisionato sono: Regressione e Classificazione.

- *Apprendimento non supervisionato*: dato un insieme di esempi $Tr = x^i$, estrarre regolarità e/o pattern che siano validi per tutto il dominio di ingresso. In questo paradigma di apprendimento non esiste alcun esperto che fornisca supervisione. [47] Un esempio di task non supervisionato è il Clustering.

- *Apprendimento con rinforzo o Reinforcement Learning*: paradigma di apprendimento automatico dove un agente interagisce con l'ambiente. L'agente si trova in uno stato s ed eseguendo una azione a ottiene uno stato successivo e una ricompensa r, la quale può essere positiva e quindi maggiore di 0, negativa ovvero minore di 0 o neutra. L'obiettivo dell'agente è massimizzare la funzione seguente chiamata funzione di ricompensa:

$$\sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{1.1}$$

dove $0 \leq \gamma < 1$. [47]

I modelli utilizzati in questo studio fanno riferimento ad una particolare sotto-categoria di modelli di Machine learning chiamata Deep Learning. Con *Deep Learning* si intende una sotto-categoria dell'apprendimento automatico basata sull'apprendimento di diversi livelli di rappresentazioni, corrispondenti a una gerarchia di caratteristiche, fattori o concetti, dove le astrazioni di livello superiore sono definite da quelle di livello inferiore. In particolare si utilizzano un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in modo sempre più completo [17]. Esempi di reti neurali sono: le reti neurali convoluzionali (CNN) e reti neurali ricorrenti (RNN). Come mostrato dalla figura 1.1, che rappresenta una classica architettura di una rete neurale formata da diversi neuroni, si può notare la presenza di molti strati che permettono di apprendere concetti complessi costruiti sopra altri più semplici, per questo motivo l'approccio è denominato Deep Learning.

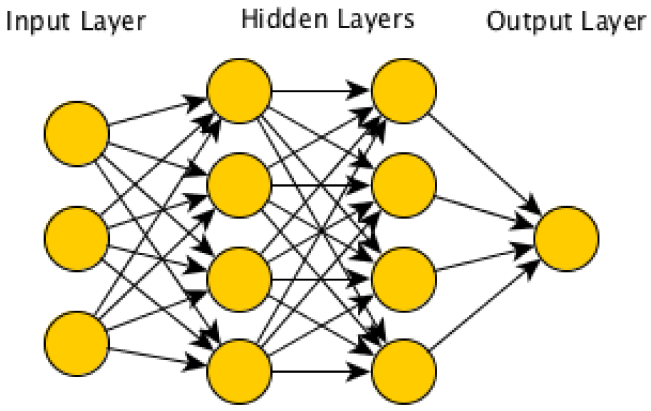


Figura 1.1: Esempio di rete neurale artificiale

1.1 Predizione di traiettorie

La capacità di prevedere i cambiamenti nell'ambiente e il comportamento dei soggetti è fondamentale in molte applicazioni come la sorveglianza, la guida autonoma, la comprensione della scena ecc. La previsione è un campo ampiamente studiato in varie ambiti dell'intelligenza artificiale e negli anni sono state proposte diverse tipologie di algoritmi che si distinguono, oltre alla metodologia usata, anche dal tipo di dati impiegati. Infatti un sottoinsieme di algoritmi si basa principalmente sull'aspetto visivo degli oggetti e della scena per sviluppare previsioni future. Al-

tri approcci, invece, utilizzano diverse forme di sensori come sensori indossabili o ambientali per conoscere gli stati passati dell'ambiente o degli oggetti.

Tipicamente una previsione può essere sotto forma di generazione di scene future o su aspetti più specifici degli oggetti presenti come ad esempio le loro traiettorie, pose, ecc.

Generalmente come riportano in [37] le tipologie di predizioni, e di conseguenza anche i modelli che le calcolano, si divino in quattro categorie principali

- *Predizione Video*: può essere considerata come la forma più generica di previsione e consiste nel predire la parte di video o di scena futura. Gli algoritmi di previsione video si basano su modelli generativi che producono immagini future data una breve osservazione o, in casi estremi, da una singola vista della scena. Sia i modelli ricorrenti che feedforward sono ampiamente utilizzati ma i modelli ricorrenti ottengo risultati migliori. Modelli che utilizzano Variational Autoencoder e Generative Adversarial Networks vengono ampliamenti utilizzati. Purtroppo, molti degli algoritmi utilizzati in questo ambito non tengono in considerazione, nella previsione, la possibilità di un oggetto di uscire o entrare nel campo visivo. Ad esempio, se un oggetto in movimento è presente nelle osservazioni e sta per uscire dal campo visivo in un prossimo futuro, gli algoritmi lo tengono in considerazione nelle scene future perché è presente durante il periodo di osservazione.

- *Predizione di azioni*: data l'osservazione delle attività correnti o le caratteristiche delle scene si prevede quale sarà l'azione successiva. Questo tipo di previsione viene molto utilizzato in diversi ambiti come attività di cucina, comprensione del traffico, previsione degli incidenti, sport ecc. Generalmente architetture neurali ricorrenti ottengono risultati decisamente migliori rispetto alle reti feedforward, dato che hanno la flessibilità di gestire lunghezze di osservazione variabili. Tuttavia questi particolari tipi di modelli come LSTM e GRU possono essere computazionalmente costosi.

- *Predizione di traiettorie*: è un campo ampiamente studiato nell'ambito della Computer Vision e consiste nel predire le posizioni future di un soggetto data una sezione del percorso che ha appena compiuto. Le applicazione riguardano, per esempio, guida intelligente e sorveglianza. Inoltre le traiettorie predette possono essere utilizzate direttamente, come nella pianificazione del percorso di veicoli autonomi o per prevedere eventi, anomalie o azioni future. Gli algoritmi di previsione della traiettoria si basano su una o più fonti di informazioni come le traiettorie passate dei soggetti, il contesto visivo circostante, gli attributi

degli oggetti, le letture dei sensori del veicolo, ecc. Un fattore molto comune che viene tenuto in considerazione da molti algoritmi di predizioni di traiettorie è l'iterazione tra i soggetti presenti, tipicamente le relazioni vengono catturate esplicitamente o implicitamente codificando le scene nel loro insieme.

- *Predizioni di movimento/pose*: viene spesso associato alla previsione di traiettorie tuttavia si basa principalmente sulla previsione dei cambiamenti nelle dinamiche, come ad esempio le pose degli agenti osservati. Tali previsioni possono essere fondamentali per molte altre applicazioni come la previsione di video o traiettorie. Anche in questa categoria vengono utilizzate reti profonde o particolare reti neurali ricorrenti per ottenere prestazioni migliori.

1.2 Obbiettivi

Il lavoro di tesi svolto ha lo scopo di verificare se è possibile, attraverso anche tecniche di Deep Learning, predire le traiettorie di diversi pedoni all'interno di un ambiente chiuso. Sebbene il task di predizione di traiettorie è ampiamente studiato dalla comunità scientifica, si limita solo a spazi aperti come predizione di percorsi di pedoni in strade o marciapiedi.

Il passaggio da un ambiente aperto e ampio ad uno chiuso e ristretto con più ostacoli che un pedone deve evitare è un cambiamento significativo che mette in discussione i risultati e le tecniche utilizzate da studi condotti precedentemente nel campo delle predizioni in ambiente aperto.

Un altro obbiettivo del lavoro di tesi è verificare se è possibile adattare un dataset, ideato per un diverso task, da utilizzare come insieme di dati di allenamento per i modelli di predizione di traiettorie. Nello specifico, a differenza di altri dataset composti da immagini prese da videocamere che ritraggono pedoni, il dataset utilizzato per il lavoro di tesi è composto da un insieme di coordinate cartesiane che identificano le posizioni dei pedoni nello spazio.

Capitolo 2

Background

La predizione di traiettorie di pedoni è un argomento molto studiato dalla comunità scientifica odierna perché le applicazione, specialmente negli ultimi anni stanno, diventando sempre più rilevanti. Poiché queste aree sono diventate più importanti e impegnative nel tempo, i metodi per affrontare questa tipologia di problemi si sono evoluti, passando da modelli fisici a modelli basati sui dati che utilizzano il Deep Learning.

In questo capitolo sarà trattato lo stato dell'arte nell'ambito della predizione di trattorie di pedoni, quindi saranno analizzati i principali modelli di intelligenza artificiali proposti per capire non solo le strategie che utilizzano ma anche il loro campo di applicazione infatti, modelli diversi possono lavorare più o meno bene in scenari differenti rendendoli specifici per un area singola di utilizzo.

2.1 Reti neurali Convoluzionali

Le *Covolutionale Neural Netowrk* [6] o reti neurali covolozionali, abbreviate con l'acronimo CNN, sono una particolare classe di reti neurali che permettono di estrarre rappresentazioni ad alto livello di immagini. Queste reti fanno uso di una particolare tecnica chiamata *convoluzione* ovvero un'operazione matematica su due funzioni e produce una terza funzione che esprime come la forma di una viene modificata dall'altra.

Le CNN sono state sviluppate negli anni 80' per riconoscere i caratteri numerici scritti a mano, per questo motivo hanno trovato subito impiego nel settore postale per l'identificazione dei codici postali. L'aspetto innovativi di queste reti era come venivano usate le immagini di input ovvero come se fossero grandi matrici di informazioni. Nella pratica una immagine a colori viene vista come una sovrapposizione di tre matrici dove ogni matrice rappresenta il livello di attivazione dei pixel

di un determinato colore. Sfortunatamente, come qualsiasi altro modello di rete neurale, anche le CNN richiedono un gran numero di dati per l'allenamento ed una abbondante potenza di calcolo, cosa che in quegli anni, non era presente e quindi ha portato l'abbandono della ricerca nello studio di nuovi campi di applicazione per questo particolare tipo di modello. Solo nel 2012 con gli studi del ricercatore Alex Krizhevsky, noto principalmente per i suoi lavori nell'ambito dell'intelligenza artificiale e sul Deep Learning, le CNN sono tornate all'attenzione della comunità scientifica. Inoltre, a differenza degli anni 80', ora sono disponibili grandi insiemi di dati per l'allenamento come il famoso dataset ImageNet e il miglioramento esponenziale dei processori ha permesso, oltre alla loro miniaturizzazione, un aumento significativo della capacità di calcolo.

Le reti neurali convoluzionali sono progettate specificamente per elaborare le immagini di input e di conseguenza hanno una architettura particolare che è composta da due blocchi principali:

- La prima sezione è la parte più caratteristica che permette, a questo tipo di rete neurale, di estrarre dalle immagini peculiarità e particolarità denominate *features*. Per fare questo viene applicata varie volte l'operazione di convoluzione con diversi filtri in modo da ottenere mappe di features diverse per poi comprimerle nella forma di un vettore.

- Il secondo blocco è più comune al resto delle reti neurali. Nel caso di una CNN utilizzata per la classificazione di oggetti, il vettore proveniente dallo strato precedente viene trasformato, con diverse combinazioni lineari e funzioni di attivazione, in un vettore con tanti elementi quanti il numero delle classi possibili di classificazione ed ogni valore esprime la probabilità che l'oggetto, rappresentato dall'immagine passata alla rete, appartenga ad un determinata classe.

Come per le normali reti neurali, i parametri degli strati sono determinati dalla retropropagazione del gradiente, con l'obbiettivo di minimizzare l'entropia incrociata ovvero la differenza di probabilità tra due variabili, durante la fase di addestramento. Nel caso della CNN, questi parametri si riferiscono alle caratteristiche dell'immagine.

Tipicamente la prima parte delle reti neurali convoluzionali è costituita da una successione di strati convoluzionali e strati di pooling. Lo *Strato convoluzionale* ha lo scopo di estrarre le mappe di features dalle immagini ricevute come input attraverso la convoluzione. Il principio è quello di "trascinare" un filtro, detto anche

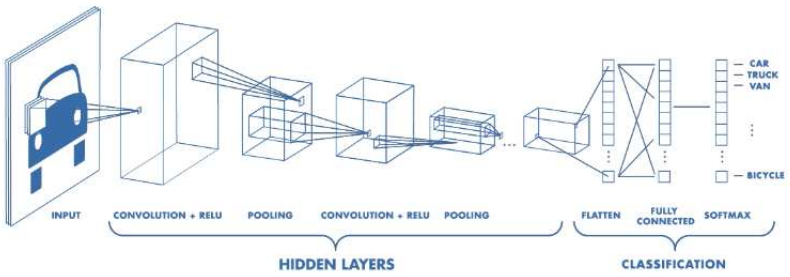


Figura 2.1: Esempio di rete neurale convoluzionale

kernel, su tutta l'immagine, che scorrendo esegue una moltiplicazione per elemento con la parte dell'immagine su cui si trova attualmente per poi sommare i risultati in un singolo valore di output. Il risultato è essenzialmente la somma pesata tra il kernel e la parte di immagine in quell'istante. Il problema della convoluzione è che ogni volta che viene processata una immagine, l'output dello strato ha dimensione minore rispetto all'immagine di partenza. Dato che in una normale CNN ci sono più strati di convoluzione la continua diminuzione delle dimensioni dell'output porta ad una perdita di informazioni tra due strati covoluzionali consecutivi. Per evitare ciò si aggiunge all'input di ogni strato convoluzionale uno strato di *padding* ovvero dei pixel aggiuntivi all'immagine, tipicamente con valore zero, per preservarne le dimensioni durante la covoluzione e non perdere così informazione.

Lo *Strato di pooling* si trova principalmente tra due strati diversi di convoluzione e applica l'operazione di pooling alle mappe di features che riceve dallo strato precedente. Il *pooling* consiste nel ridurre la dimensionalità delle mappe di features preservando le loro caratteristiche più importati, ciò migliora l'efficienza della rete ed evita l'ovverfitting ovvero l'eccessivo apprendimento. Tipicamente le strategie di pooling più utilizzate sono il *Max pooling* e il *Average pooling*. In entrambi i casi si va a suddividere la mappa delle features in sezioni uguali per poi elaborale e ridurne la loro dimensione ad un solo elemento. La differenza tra i due metodi è che nel primo si va a prendere il valore più grande in assoluto mentre nel secondo si calcola una media aritmetica.

Nel campo della traduzione automatica e nella descrizione testuale delle immagini è stato provato, in lavori come [8] e [21], che le CNN sono una valida alternativa a modelli più comunemente usati come le Recurrent Neural Networks. Tuttavia è stato dimostrato, nello studio [57] come questo modello riesca ad avere buone prestazioni

anche nell'ambito di predizioni di traiettorie di pedoni.

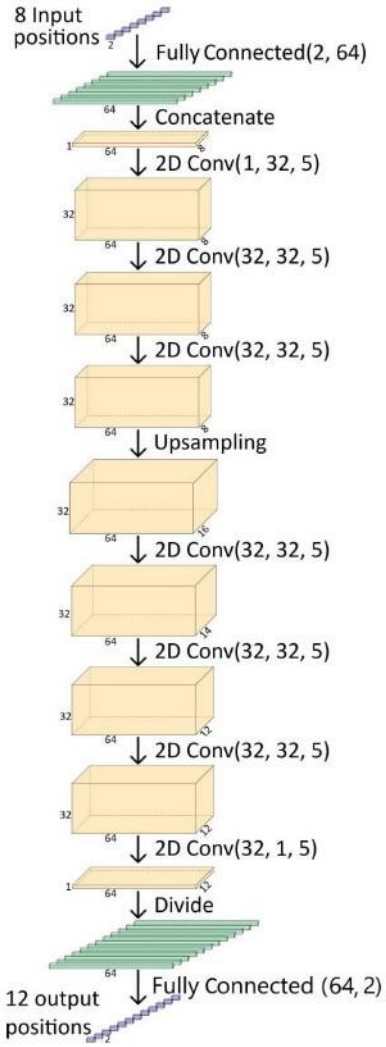


Figura 2.2: Rete neurale convoluzionale proposta per la predizione di traiettorie di pedoni.
Per lo strato completamente connesso nelle parentesi viene indicata la dimensione di input e output. Invece per gli strati convoluzionali nelle parentesi viene riportato i canali di input e di output e la dimensione del kernel

L'architettura proposta, raffigurata nella figura 2.2 ,è stata appositamente creata per permettere alla CNN di predire una sequenza di posizioni future di un pedone osservando un numero di posizioni passate. Nello specifico data una osservazione di 3.2 secondi suddivisa in 8 posizioni distinte fatte da uno specifico utente che termina all'istante t-1, il modello riesce a predire le successive 12 posizione a partire dal momento t fino a t+12. A differenza delle Recurrent Neural Networks dove le osservazioni vengono date alla rete singolarmente preservando un ordine cronologico, la CNN sviluppata riceve, come input tutti le 8 posizioni dell'osservazione e le codifica in un vettore di lunghezza 64 attraverso uno strato completamente connesso. Successivamente le 8 posizioni codificate vengono unite per formare una sola matrice di

dimensione 64x8 dove 64 è la dimensione di una singola posizione codificata e 8 è il numero complessivo delle posizioni di input. A questo punto è possibile applicare la covoluzione e per ogni strato di convoluzione all'immagine di input viene aggiunto un padding che dipende dalla dimensione del kernel che si sta usando in quel specifico strato. L'utilizzo del padding permette non solo di mantenere il numero di features in output uguale al numero di featurse in input ma anche di ampliare il modello con altri strati di convoluzione senza perdita di informazione.

Per risolvere il problema di differenza di dimensione tra input e output viene inserito uno strato di sovra-campionamento che raddoppia il numero di features quindi si passsa dalle 8 precedenti alle 16. Infine un secondo gruppo di strati di convoluzione viene aggiunto per portare la dimensione delle features a 12. Il vettore di output rappresenta le 12 coordinate, in ordine cronologico, predette.

Per determinare se l'architettura proposta di CNN è valida per la predizione di trattorie di pedoni si sono confortati i risultati ottenuti con quelli di due modelli molto utilizzati in letteratura in questo ambito ovvero un semplice LSTM e un Decoder-Encoder che utilizza le celle LSTM in entrambe le parti. Tutti i modelli sono stati allenati con i dataset ETH e UCY, che come descritti nel paragrafo 3.2, sono due dataset disponibili pubblicamente e ampiamente utilizzati in letteratura. Insieme contengono cinque scene, due in ETH, chiamate ETH e Hotel, e tre in UCY, chiamate rispettivamente Univ, Zara1 e Zara2. In totale, contengono più di 1400 traiettorie di pedoni, con posizioni raccolte ad intervalli regolari di 0,4 secondi. Questi due dataset sono stati utilizzanti congiuntamente e i dati per il training e per il test sono stati ottenuti con l'approccio *leave-one-out cross-validation* ovvero il modello viene addestrato su quattro scene e testato sulla quinta, e dato che ci sono cinque scene la procedura vine ripetuta cinque volte scegliendo ogni volta una scena diversa per il test.

Per poter paragonare le traiettorie predette e valutarne la bontà sono state utilizzate due metriche ovvero *Average Displacement Error (ADE)* e *Final Displacement Error (FDE)*. La metrica ADE si riferisce all'errore quadratico medio tra tutti i punti stimati di una traiettoria e quelli originali su tutti i pedoni.

$$ADE = \frac{\sum_{i=1}^{N} \sum_{T_{obs}}^{T_{pred}} [(\hat{x}_i^t - x_i^t)^2 + (\hat{y}_i^t - y_i^t)^2]}{n(T_{pred} - T_{obs})} \tag{2.1}$$

dove N è il numero di pedoni e le coordinate $(\hat{x}_i^t, \hat{y}_i^t)$ indicano le posizioni dei punti predette dal modello.

Mentre la FDE esprime la distanza tra la destinazione finale prevista e la destinazione reale come media su tutti i pedoni.

$$FDE = \frac{\sum_{i=1}^{N} \sqrt{(\hat{x}_i^{T_{pred}} - x_i^{T_{pred}})^2 + (\hat{y}_i^{T_{pred}} - y_i^{T_{pred}})^2}}{n} \tag{2.2}$$

I risultati ottenuti mostrano come il modello di CNN proposto riesca ad avere risultati migliori rispetto ai modelli di confronto LSTM e Encoder-Decoder. Inoltre sono stati fatti degli esperimenti per determinare la grandezza ottimale del kernel utilizzato negli starti convoluzionali e i risultati suggeriscono che all'aumentare delle dimensioni del filtro si ottengono predizioni più accurate. L'intuizione alla base del motivo per cui al crescere delle dimensione del kernel si ottengono predizioni più accurate è che più informazioni un kernel può elaborare allora può anche interpretare più adeguatamente comportamenti complessi nella traiettoria.

In conclusione è effettivamente possibile sviluppare un modello convoluzionale in grado di superare i modelli ricorrenti nella previsione di traiettorie pedonali.

2.2 Generative Adversarial Network

Prevedere il comportamento delle persone, specialmente in ambienti affollati, è un sfida complessa a causa delle proprietà intrinseche dell'uomo all'interno della folla. Infatti una persona decide il proprio percorso in base alle persone che ha attorno e gli esseri umani hanno un'innata abilità nel prevedere le decisioni dei vicini per muoversi in ambienti affollati. Inoltre i percorsi sono determinati anche da norme sociali come tenere la destra o rispettare lo spazio personale e tali norme rendono traiettorie fisicamente possibili in percorsi improbabili per le persone. Infine non è detto che osservando le decisioni passate di un pedone ci sia solo una predizione del suo percorso futuro.

In letteratura esistono diverse proposte per predire il futuro percorso di un pedone tuttavia questi metodi hanno due limitazioni:

- Non hanno la capacità di modellare le interazioni tra tutte le persone che compongono una determinata scena in modo efficiente;
- Tendono ad imparare un comportamento comune tra i pedoni.

La proposta implementata nello studio [24] permette di superare queste limitazioni utilizzando le *Generative Adversarial Networks*.

Le *Generative Adversarial Networks* [15], abbreviate con GAN, sono un approccio alla modellazione generativa che utilizza metodi di Deep Learning. La *modellazione generativa* è un'attività di apprendimento non supervisionata nel Machine Learning che prevede la scoperta e l'apprendimento automatico di regolarità nei dati di input in modo che il modello possa, utilizzando queste caratteristiche, generare o produrre nuovi esempi simili i dati originali.

Le GAN rappresentano una soluzione intelligente per allenare modelli generativi in modo supervisionato utilizzando due sotto-modelli chiamati tipicamente:

- Generatore: modello che viene appositamente allenato per creare esempi nuovi sempre più simili ai dati originali;
- Discriminatore: prova a classificare i dati come veri se provengono dal dataset di partenza o falsi se vengono generati dal generatore.

I due modelli vengono allenati insieme in un gioco a somma zero, ovvero il guadagno o la perdita di un modello è perfettamente bilanciata dalla perdita o guadagno dell'altro modello, fino a quando il discriminatore viene ingannato per almeno la metà delle volte cioè se classifica degli elementi creati dal generatore come esempi provenienti dai dati originali, questo significa che il modello generatore sta creando esempi estremamente realistici.

Il modello proposto in [24] ha lo scopo di generare molteplici traiettorie socialmente accettabili dato un insieme di percorsi osservati. Per far ciò è stata realizzata un architettura specifica di GAN, mostrata in figura 2.3, che prevede un RNN Encoder-Decoder come generatore e un RNN encoder come discriminatore

Il modello accetta in input le traiettorie di tutti i pedoni, presenti nella scena, dall'istante temporale $t = 1, ..., t_{obs}$ definite come $X = X_1, X_2, X_3, ..., X_n$ e predice i futuri percorsi $\hat{Y} = \hat{Y}_1, \hat{Y}_2, \hat{Y}_3, ..., \hat{Y}_n$ dal tempo $t = t_{obs} + 1, ..., t_{rped}$.

La procedura di allenamento è simile a una partita min-max a due giocatori con la seguente funzione obiettivo:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[\log(D(x)] + \mathbb{E}_{z \sim p_{(z)}}[\log(1 - D(G(z)))] \tag{2.3}$$

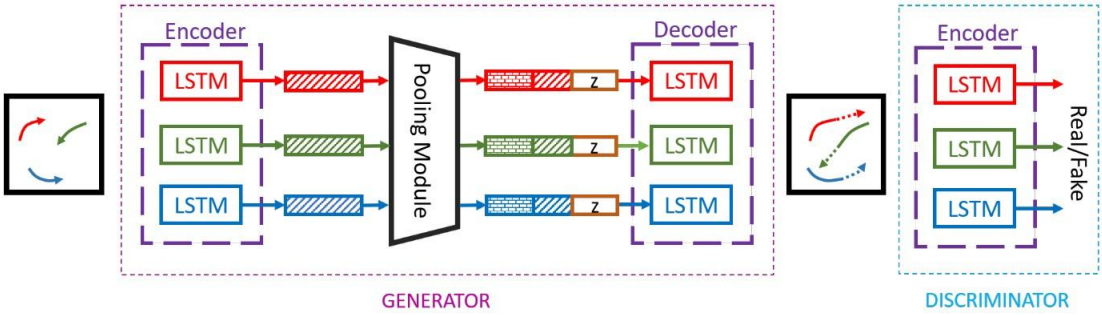


Figura 2.3: Architettura di Generative Adversarial Network proposta nello studio "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks"

Nel dettaglio $D(x)$ è la stima del discriminatore ovvero la probabilità che l'istanza x presa dai dati originali sia reale. $G(z)$ è l'output del generatore quando viene dato il rumore z preso da una distribuzione $\mathcal{N}(0,1)$ quindi $D(G(z))$ rappresenta la stima di probabilità calcolata dal discriminatore che un'istanza falsa sia reale.

Analizzando il funzionamento della rete come prima cosa il Generatore utilizza un *Multilayer perceptron* con ReLU come funzione non lineare per codificare ogni posizione, appartenente ad una traiettoria di uno specifico utente, da due coordinate a un vettore e_i^t di dimensione uguale a 16 che esprime le coordinate al tempo t del pedone i. Successivamente il vettore viene dato in input alle LSTM che compongono l'encoder. Purtroppo l'uso di una LSTM per persona non riesce a catturare le interazioni tra molti soggetti, per questo viene inserito *Pooling Module*, all'interno del Generatore, che collega gli strati nascosti del encoder a quelli del decoder, in questo modo si riescono ad estrarre iterazioni tra i pedoni della scena.

Tipicamente le normali GAN prendono in input del rumore per generare nuovi esempi e quindi non sarebbe possibile prevedere traiettorie future osservando l'andamento passato di un pedone. Tuttavia questo è possibile usando il Pooling Module che dopo aver osservato t_{obs} istanti comprime gli strati nascosti delle diverse LSTM in un unico tensore per inizializzare lo strato nascosto del decoder. Prima di questo il tensore deve essere codificato tramite Multilayer perceptron con ReLU come funzione non lineare. Infine le previsione saranno ottenute dalla decodifica dello strato nascosto delle LSTM che formano il decoder.

Per quanto riguarda il Discriminatore riceve in input $T_{real} = [X_i, Y_i]$ e $T_{fake} = [X_i, \hat{Y}_i]$ e li classifica in veri e falsi. Per permettere questa classificazione alla fine dell'encoder viene messo un Multilayer perceptron per ottenere un valore di probabilità che esprima l'appartenenza di input ad una determinata classe.

Il modello descritto finora produce buoni risultati, ma nel caso in cui la previsione è incerta ovvero che ci possono essere più traiettorie possibili allora la soluzione prodotta è una traiettoria che esprime la media di tutte quelle possibili. Inoltre gli autori hanno scoperto che le previsioni non erano molto sensibili ai cambiamenti nel rumore, utilizzato dal Generatore per creare nuovi esempi, e quindi erano molto simili tra di loro. Per evitare questo hanno sviluppato una nuova funzione di loss che incoraggiasse la rete a produrre campioni diversi. Questa funzione prevedeva la generazione di k possibili previsioni di output campionando casualmente z da $\mathcal{N}(0, 1)$ e poi scegliendo la migliore. La funzione di loss è definita come:

$$\mathcal{L}_{variety} = \min_k \|Y_i - \hat{Y}_i^{(k)}\|_2 \tag{2.4}$$

Come per i modelli precedentemente visti in questo capitolo anche questo particolare tipo di GAN è stato allenato con i dataset ETH e UCY e le traiettorie predette sono state valutate con le metrice ADE e FDE.

Sono stati condotti varie esperimenti mettendo in confronto il modello sviluppato dagli autori con e senza il Polling Module e la Variety loss e diversi modelli proposti in letteratura. I risultati ottenuti segnalano come la Generative Adversarial Network senza Pooling Module e Variety loss sia stata meno efficace rispetto ad una normale LSTM poiché ogni traiettoria predetta può essere una qualsiasi delle molteplici possibili traiettorie future. Invece il modello GAN con Variety loss e k uguale a 20 ha ottenuto i risultati migliori superando i modelli classici utilizzati in letteratura. Invece per quanto riguarda il modello con Variety loss e Pooling Module gli autori hanno ottenuti valori delle metriche leggermente peggiori rispetto al modello con solo la nuova funzione di loss.

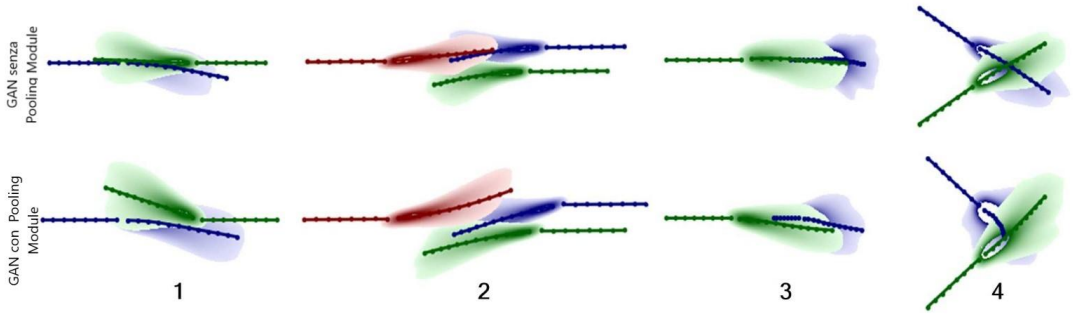


Figura 2.4: Confronto tra traiettorie generate da GAN senza e con Pooling Module in quattro scenari di prevenzione delle collisioni

Tuttavia come mostrato in figura 2.4 questo modello ha ottenuto traiettoria qualitativamente migliori da un punto di vista sociale ovvero riesce ad ottenere

percorsi più simili a quelli che farebbe una normale persona. Per ogni esempio nella figura 2.4 sono stati disegnati 300 campioni ed è stata tracciata una distribuzione approssimativa delle traiettorie insieme alla previsione della traiettoria media.

2.3 Reti neurali ricorrenti

Le *Reti neurali ricorrenti*, denominante con l'acronimo RNN, sono un particolare tipo di rete neurale artificiale appositamente adattata per lavorare con dati di serie temporali o dati che coinvolgono sequenze. Quindi le RNN possono essere impiegate in task di *Speech-to-text* ovvero la conversione di voce in testo scritto, *Sentiment Analysis* cioè un campo dell'elaborazione del linguaggio naturale che si occupa di costruire sistemi per l'identificazione ed estrazione di opinioni dal testo, oppure in task di *Automatic translation* e quindi di traduzione istantanea da una lingua ad un altra. Di conseguenza i dati che usano questo particolare tipo di rete neurale sono audio, frasi provenienti da diversi testi o addirittura documenti interi dove una parola viene inserita in un contesto ed è in funzione della precedente. Le normali reti neurali feed-forward sono concepite per dati slegati ovvero un insieme di punti indipendenti tra di loro, tuttavia se i dati esprimono una sequenza ovvero dove un certo dato è conseguenza di un altro allora bisogna modificare l'architettura della rete in modo che sia in grado di apprendere questa dipendenza.

Per fare in modo che le RNN riescano ad imparare la dipendenza tra un dato e i suoi successori è necessario che siano dotate di una "memoria" dove memorizzare ciò che è accaduto in istanti temporali passati. Questa memoria viene implementata attraverso una connessione retrograda nello strato nascosto. La figura 2.5 mostra la differenza tra una normale rete neurale feed-forward e un rete neurale ricorrente e si può notare come nella seconda sia presente una connessione retrograda nello strato nascosto che permette alla rete di "ricordare" cosa è successo all'istante temporale precedente.

L'apprendimento di una rete neurale si può vedere come un problema di ottimizzazione dove si cerca di minimizzare una funzione complessa con molti parametri. Per trovare i giusti parametri che portano ad un minimo della funzione non si utilizza un approccio analitico bensì iterativo che permette di muoversi sulla superficie della funzione a piccoli passi verso la direzione di un minimo. Una tecnica che permette questo è la Backpropagation con discesa del gradiente. La *Backpropagation* [11] è un modo per propagare all'indietro, nella rete e nel tempo, l'errore commesso nel calcolo dell'output per modificare i pesi delle connessioni neurali in modo da

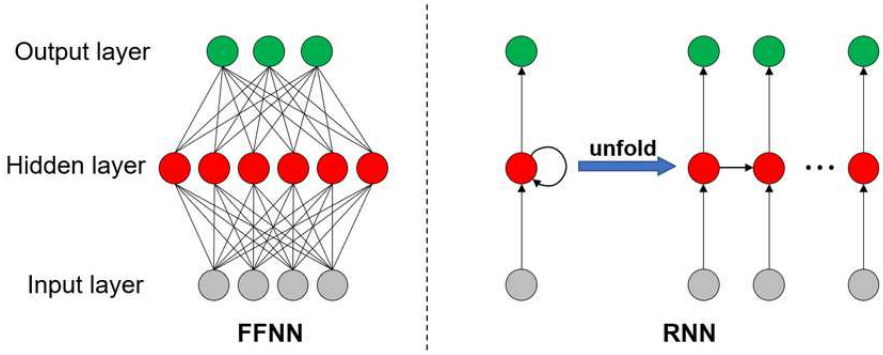


Figura 2.5: Feed-Forward Neural Netwrok contro Recurrent Neural Network

miglioralo. Mentre il *Gradiente* è definito come il vettore che ha come componenti le derivate parziali di una funzione con più variabili a valori reali.

La Backpropagation con discesa del gradiente ha essenzialmente due criticità che vengono accentuate nelle reti neurali ricorrenti:

- *Vanishing Gradient Problem*: è un fenomeno che crea difficoltà nell'addestramento delle reti neurali. Nella pratica, durante la Backpropagation, ogni parametro del modello riceve, ad ogni iterazione, un aggiornamento proporzionale alla derivata parziale della funzione rispetto al parametro considerato. Le reti neurali fanno uso principalmente di funzioni di attivazione non lineari come la tangente iperbolica o la funzione logistica, che hanno valori di gradiente compresi tra 0 e 1. Dato che l'algoritmo di Backpropagation, per ogni livello della rete, calcola i gradienti tramite la regola della catena allora il risultato del prodotto di n numeri, con valori tra 0 e 1, decresce esponenzialmente rispetto alla profondità della rete. Quindi la propagazione dell'errore ai livelli iniziali della rete non avverrà poiché avrà valore troppo basso.

- *Exploding Gradient Problem*: è il problema opposto del Vanishing Gradient ovvero è sempre un fenomeno che crea difficoltà nell'apprendimento dove i gradienti calcolati ad ogni livello della rete diventano sempre più grandi.

Questi due fenomeni impediscono alle reti neurali ricorrenti di apprendere dipendenze lontane all'interno della sequenza, per questo sono state sviluppate negli anni diverse architetture di reti ricorrenti che potessero mitigare questi problemi durante l'apprendimento.

2.3.1 Attenzione nelle reti neurali

Negli ultimi anni sempre più studi hanno dimostrato le notevoli capacità di apprendimento dai dati da modelli che sfruttano un approccio di Deep Learning. Inoltre i numerosi successi riportati dalle reti ricorrenti nei capi della traduzione istantanea o della trascrizione del parlato in testo hanno portato l'attenzione della comunità scientifica che continua a proporre nuove soluzione per migliorare questa tipologia di rete neurale.

Un approccio ideato per permettere alle reti neurali ricorrenti di apprendere dipendenze lontane all'interno di sequenze lunghe è il *Meccanismo di attenzione*.

In psicologia, *l'attenzione* è il processo cognitivo che permette ad un individuo di concentrarsi selettivamente su uno o pochi dettagli ignorando gli altri. Il *Meccanismo di attenzione* è un tentativo per implementare lo stesso concetto, che si può trovare negli esseri umani, nelle reti neurali affinché possano concentrarsi selettivamente su alcune caratteristiche rilevanti ignorandone altre. Il meccanismo di attenzione compare per la prima volta nel campo della automatic translation come miglioramento di reti neurali basate su encoder e decoder. Successivamente, questo meccanismo è stato utilizzato in altri campi applicativi come nell'elaborazione del parlato e in Vision Cognitive.

Prima dello studio [10], i modelli per la traduzione automatica si basava su reti ricorrenti che utilizzavano pile di celle LSTM all'interno sia dell'encoder che del decoder.

L'encoder LSTM viene utilizzato per elaborare l'intera frase di input e codificarla in un vettore di contesto. Tipicamente viene preso lo stato nascosto dell'ultima cella per diventare l'input del decoder, perchè rappresenta un buon riassunto della frase di input, mentre gli altri stati nascosti intermedi vengono ignorati. Il decoder invece ha il compito di produrre in sequenza le parole dato il vettore di contesto precedentemente calcolato.

Il principale svantaggio di questo approccio è la scarsa qualità del riassunto prodotto dall'encoder, portando un output sbagliato del decoder, in questo caso una sbagliata traduzione. In particolare l'encoder soffre del *Long-range dependency problem* [12] ovvero più la sequenza è lunga più l'encoder fatica ad apprendere le dipendenze al suo interno.

Infatti come riportato nello studio [10] *"Il principale problema con l'approccio encoder-decoder è che una rete neurale deve essere in grado di comprimere tutte le informazioni necessarie, di una frase sorgente, in un vettore di lunghezza fissa. Ciò può rendere difficile per la rete neurale far fronte a frasi lunghe"*.

Il meccanismo di attenzione aiuta a guardare tutti gli stati nascosti dalla sequenza dell'encoder per fare previsioni, a differenza dell'approccio standard encoder-decoder. La differenza tra una RNN con e senza il meccanismo di attenzione viene mostrato nella figura 2.6

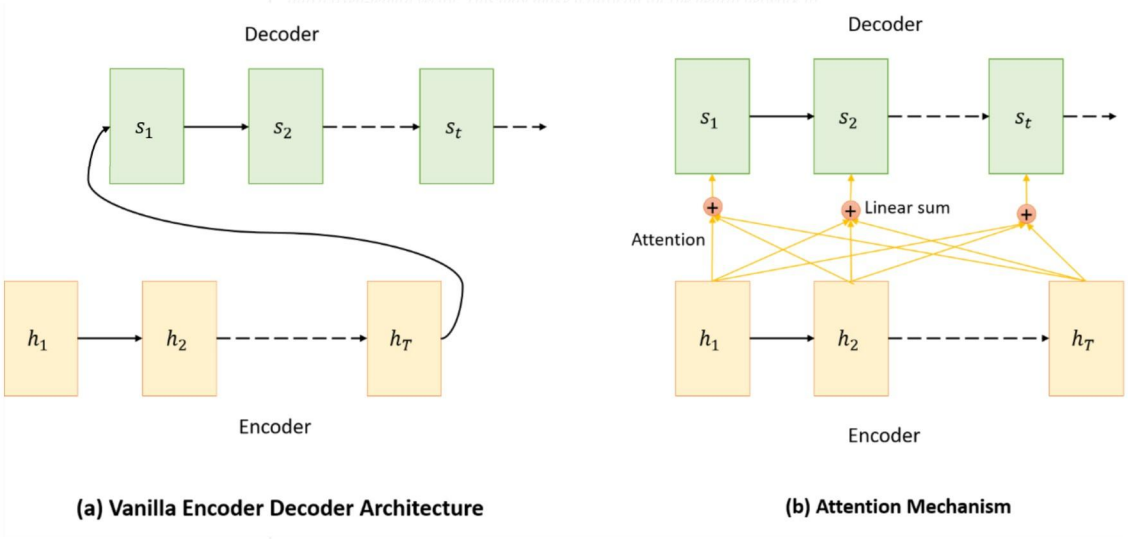


Figura 2.6: Differenza tra architettura ecoder-decoder e meccanismo di attenzione

In una semplice architettura encoder-decoder il decoder inizia a fare previsioni guardando solo l'output dell'ultima cella dell'encoder che ha condensato tutte le informazioni. Invece, l'architettura basata sull'attenzione collega ogni stato nascosto dell'encoder a ciascun nodo del decoder e fa previsioni dopo aver deciso quale nodo è più informativo.

Questa strategia viene anche utilizzata in ambiti diversi dalla traduzione di parole e testi, infatti con lo studio [20] viene utilizzata per definire un modello in grado di predire il percorso futuro di pedoni. Nel dettaglio viene proposto un nuovo metodo per prevedere le azioni future di un pedone dato uon storico dei suoi movimenti precedenti e di quelli dei suoi vicini. La novità del metodo proposto è l'utilizzo di due tipologie diverse di meccanismo di attenzione, uno definito "soft" e l'altro "hard-wired" per mappare le informazioni delle traiettoria dei pedoni limitrofi in modo da predire le posizioni future del pedone di interesse. Il modello proposto si divide in due principali parti: un encoder e un decoder.

L'encoder riceve in input una sequenza x, formata da diversi posizioni raccolte in ogni istante temporale, dalla quale genera una sequenza codificata h, quindi per un determinato pedone i sarà creata la sequenza $h_i = [h_1, h_2, ..., h_T]$. Per codificare

la sequenza x di un pedone viene utilizzata una singola cella LSTM, quindi:

$$h_t = LSTM(x_t, h_{t-1}) \tag{2.5}$$

Il decoder, invece è più complicato perché va ad utilizzare i due meccanismi di attenzione per creare il vettore di contesto con il quale andrà a predire le traiettorie future del pedone di interesse.

Il meccanismo di attenzione "soft" consiste nel creare il vettore di contesto C_t^s per il pedone preso in esame in modo da permettere alla rete di concentrarsi con diversi gradi sulla sequenza di input. Il vettore viene calcolato come una somma pesata degli stati nascosti:

$$C_t^s = \sum_{j=1}^{T_{obs}} = \alpha_{tj} h_j \tag{2.6}$$

dove i coefficienti α_{tj} vengono calcolati dalla seguente equazione:

$$\begin{aligned} a_{tj} &= \frac{exp(e_{tj})}{\sum_{k=1}^{T} exp(e_{tk})} \\ e_{tj} &= \alpha(s_{t-1}, h_j) \end{aligned} \tag{2.7}$$

mentre α è una rete neurale feed-forward per l'allenamento congiunto con altri componenti del sistema. Il calcolo di C_t^s è computazionalmente oneroso quindi non sarebbe stato efficiente da applicare a tutti i pedoni che si trovano vicino al utente di interesse dato che pedoni lontani nello spazio influiscono poco nelle decisioni. Per questo gli autori hanno sviluppato un meccanismo di attenzione "hard-wired" che permette di semplificare il calcolo del vettore di contesto. Infatti questa strategia non ha bisogno di calcolare per ogni istante t, di ogni pedone, il valore appropriato di α_{tj} per lo stato nascosto h_t ma questo valore viene stabilito a priori e rimarrà fisso per tutto l'allenamento della rete. I pesi di attenzione "hard-wired" sono stati progettati per rappresentare il concetto di distanza tra il pedone di interesse e tutti gli altri pedoni limitrofi, quindi persone che saranno più vicine avranno un peso associato maggiore rispetto a quelle lontane. Nella pratica i pesi vengono calcolati come:

$$w(n, j) = \frac{1}{dist(n, j)} \tag{2.8}$$

dove $dist(n, j)$ esprime la distanza tra il n-esimo pedone rispetto al pedone di interesse all'istante temporale j. Quindi se un pedone di interesse ha N utenti vicini e $h'_{(n,j)}$ è la rispettiva codifica dello stato nascosto del n-esimo pedone all'istante j, allora il vettore di contesto per il meccanismo di attenzione "hard-wired" è definito

come:

$$C_h = \sum_{n=1}^{N} \sum_{j=1}^{T_{obs}} w(n,j) h'_{(n,j)} \tag{2.9}$$

Infine attraverso un semplice layer di concatezione i diversi vettori vengono uniti per formare il vettore di contesto. Il vettore di contesto viene ricavato dalla seguente formula:

$$c_y^* = \tanh(W_c[C_t^s, C_h]) \tag{2.10}$$

dove W_c è una matrice di pesi per la concatenazione delle matrici che vengono appresi con la Backrpopagation durante la fase di allenamento.

Dopo aver determinato il vettore di contesto le predizioni delle future traiettorie saranno fate come:

$$y_t = LSTM(s_{t-1}, y_{t-1}, c_y^*) \tag{2.11}$$

dove s_{t-1} è lo stato nascosto del decoder all'istante precedente. Nella figura 2.7 viene riassunta l'architettura utilizzata.

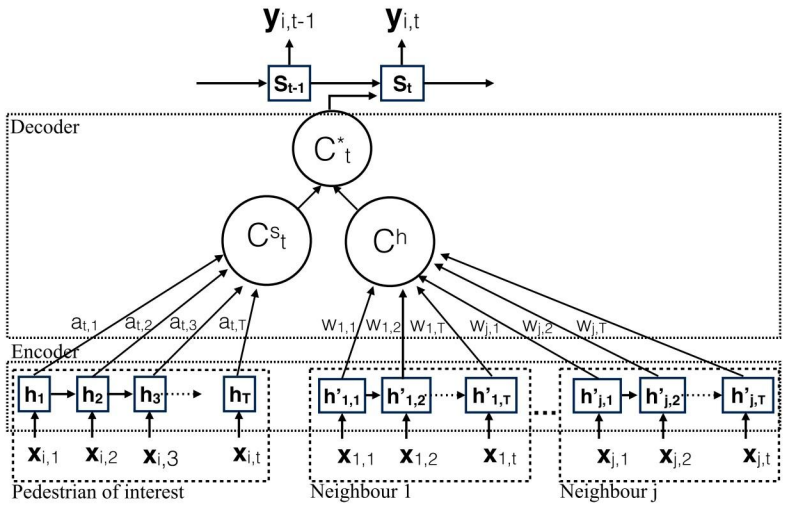


Figura 2.7: Architettura proposta che sfrutta il meccanismo di attenzione "soft" e "hard-wired"

Per determinare la bontà del modello sviluppato gli autori lo hanno confrontato con il modello implementato nello studio [5] chiamato Social-LSTM, considerato il modello di confronto principale, in uno scenario reale. Nello specifico si è andato a determinare il comportamento dei due modelli nel predire una traiettoria con un brusco cambiamento di direzione, ad esempio una persona che sta camminando verso la fermata dell'autobus ma poi si accorge che non ha comprato il biglietto e quindi torna indietro. In questo caso il modello di LSTM implementato da [5] è in grado di generare bruschi cambiamenti di direzione dato lo stato nascosto precedete tuttavia

non è in grado di predire comportamenti così radicali a lungo termine. Al contrario, il modello di attenzione che sfrutta i due meccanismi considera l'intera sequenza di stati nascosti sia per il pedone di interesse che per i suoi vicini e utilizza pesi dipendenti dal tempo che consentono di variare l'influenza, dei vari stati nascosti, in modo tempestivo.

Per determinare questa superiorità gli autori hanno allenato entrambi i modelli con le traiettorie provenienti dai dataset New York Grand Central e Edinburgh Informatics Forum che contengono rispettivamente 12600 e 90000 traiettorie. Successivamente i risultati dei due modelli modelli sono stati confrontati attraverso le metriche ADE e FDE che hanno mostrato come le previsioni del modello che utilizza il meccanismo di attenzione sono migliori rispetto a Social-LSTM.

2.3.2 Uso del contesto per previsioni

Come visto in precedenza predire il percorso che farà un determinato pedone non è un compito facile perchè le sue decisioni sono influenzate da molti fattori. Infatti quando una persona sta camminando in un luogo aperto pubblico tiene conto dei diversi tipi di oggetti che incontra aggiustando il suo percorso per evitare collisioni. Ad esempio, è meno probabile che in zone erbose si trovano pedoni rispetto ai marciapiedi o a strade pedonali. Inoltre quando si avvicina alla sua destinazione adotta comportamenti che possono provenire dalla sua esperienza o da stimoli visivi per evitare minacce e così selezionare il percorso più breve.

Già numerosi studi hanno dimostrato come i modelli LSTM siano in grado di catturare il modello di comportamento dei pedoni per predire il loro percorso futuro grazie alla loro capacità di apprendere, ricordare e dimenticare. Questa caratteristica li ha resi una delle soluzioni più adatte per risolvere problemi di sequenza a sequenza.

Molti modelli, che utilizzano LSTM, si sono concentrati sul modellare le iterazioni tra le persone presenti in una scena tuttavia per predire traiettorie più verosimile non è sufficiente analizzare solo il comportamento tra i pedoni in un ambiente, ma occorre integrare queste informazioni con il contesto della scena. Le informazioni sul contesto della scena sono fondamentali per migliorare la previsione delle posizioni future dei pedoni perchè permettono di aggiungere vincoli fisici e fornisco percorsi più realistici.

A questo scopo nello studio [32] è stato proposto un modello per predire traiettorie future di pedoni che tiene in considerazione non solo elementi sociali ma anche informazioni dello spazio circostante. Nello specifico il modello si basa sull'architettura di Social-LSTM, proposta in [5] che permette alla rete di catturare le

iterazioni tra pedoni vicini, ma vengono aggiunti nuovi fattori che codificano anche le interazioni tra le persone e lo spazio al fine di ottenere previsioni più accurate.

In particolare, il modello proposto, utilizza tre metodi diversi al fine di catturare le informazioni di contesto della scena, chiamati rispettivamente: *Social, Navigation e Semantic pooling*. Il primo meccanismo tiene conto delle iterazioni tra un pedone e le persone che ha accanto, fondendo i loro stati nascosti. Il meccanismo di Navigation sfrutta le osservazioni passate per determinare posizioni future di un pedone, mentre l'ultimo meccanismo usa le informazioni semantiche della scena per determinare quali sono le aree usate dalle persone. In figura 2.8 viene mostrato lo schema generale del modello.

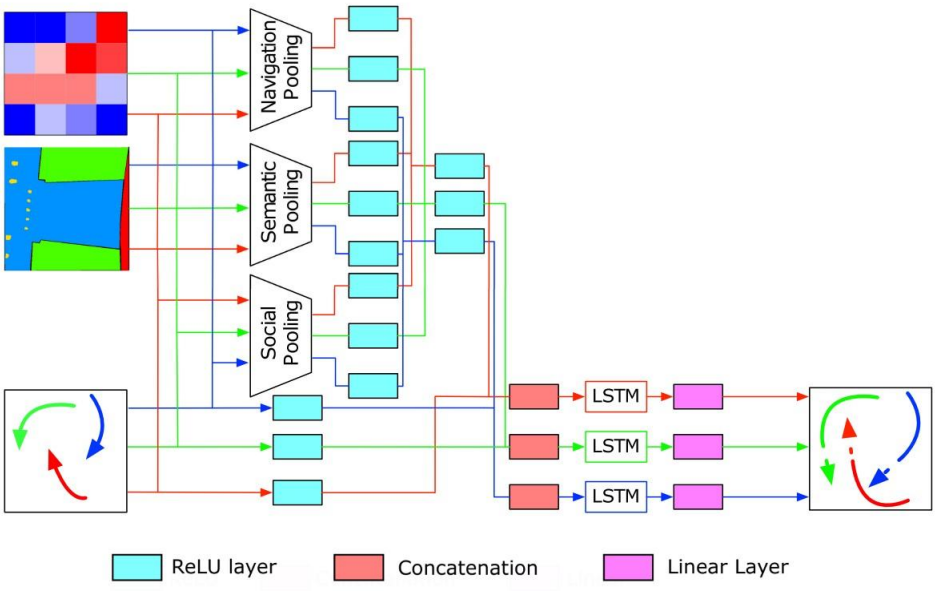


Figura 2.8: Architettura del modello Social Navigation Semantic LSTM

La traiettoria di un pedone i viene rappresentata da una sequenza di coordinate cartesiane e ogni traiettoria viene, successivamente, associata a una rete LSTM che è descritta dalle seguenti funzioni:

$$\begin{aligned} f_t^i &= \sigma(W_f x_t^i + U_f h_{t-1}^i + b_f) \\ i_t^i &= \sigma(W_i x_t^i + U_i h_{t-1}^i + b_i) \\ o_t^i &= \sigma(W_o x_t^i + U_o h_{t-1}^i + b_o) \\ c_t^i &= f_t^i \odot c_{t-1}^i + i_t^i \odot (W_c x_t^i + U_c h_{t-1}^i + b_c) \\ h_t^i &= o_t^i \odot \tanh(c_t^i) \end{aligned} \tag{2.12}$$

Le informazioni estratte da queste reti LSTM vengono utilizzate dai meccanismi di Social, Navigation e Semantic pooling per inizializzare tre diversi tensori:

- *Social Tensor*: esprime le relazione tra un pedone e le persone che incontra. Nello specifico si tengono in considerazione gli stati nascosti come mostrato dalla seguente formula:

$$H_t^i(m,n,:) = \sum_{j \in N_i} 1_{mn}[x_t^i - x_t^j, y_t^i - y_t^j] h_{t-1}^j \tag{2.13}$$

  dove $1_{mn}(x,y)$ controlla se il punto è all'interno della cella m,n

- *Navigation Tensor*. Per determinare quali fossero le aree pedonali ammissibili viene definita una *Navigation Map* $\mathcal{N}$ che conta la frequenza di attraversamento dei pedoni per una specifica zona nella scena. Questa mappa viene utilizzata per definire il Navigation Tensor come segue:

$$N_t^i(m,n) = \mathcal{N}_{mn} \tag{2.14}$$

  Questo tensore esprime la frequenza del vicinato dell'i-esimo pedone per la cella (m,n) considerando tutte le osservazioni passate per tale cella.

- *Semantic Tensor*: cattura le dinamiche dei pedoni legate alla semantica dello spazio circostante. A tal fine ogni pixel dell'immagine viene rappresentato da un vettore one-hot per esprime una delle seguenti classi semantiche: $\mathcal{C} =$ {erba, ostacolo, panchina, auto, strada, marciapiede }. In generale il Semantic Tensor viene calcolato mediante la seguente formula:

$$S_t^i(m,n,:) = \frac{1}{|S_{mn}|} \sum_{j \in S_{mn}} v_j \tag{2.15}$$

  dove v_j è il vettore one-hot per il pixel j mentre $|S_{mn}|$ è il numero di pixel nella cella (m,n).

Infine le informazioni espresse dai tre diversi tensori vengono utilizzate come input per una rete LSTM come descritto dalla seguente formula:

$$\begin{aligned} g_t^i &= \Phi(concat(a_t^i, n_t^i, s_t^i), W_g) \\ h_t^i &= LSTM(h_{t-1}^i, concat(e_t^i, g_t^i), W_h) \end{aligned} \tag{2.16}$$

dove Φ è la funzione di attivazione non lineare ReLU, a_t^i, n_t^i, s_t^i sono le codifiche in vettore dei tensori Social, Navigation e Semantic mentre e_t^i è la codifica in vettore delle coordinate cartesiane che descrivono la trattoria di uno specifico pedone.

Per determinare se il contesto della scena possa realmente portare a predizioni migliori sono stati testati tre diversi modelli denominati SN-LSTM, SS-LSTM e SNS-LSTM, dove nei primi due si è voluto verificare separatamente gli effetti dell'utilizzo dei soli tensori Navigation e Sematic mentre nel modello SNS-LSTM si sono combinate entrambe le informazioni.

Nello specifico i modelli dovevano predire una traiettoria di 4,8 secondi, ovvero di 12 punti, data una osservazione di 8 posizioni cioè di 3,2 secondi. Per confrontare i risultati dei vari modelli, al fine di determinare il migliore, sono state utilizzate le metriche ADE e FDE. Infine entrambe le reti neurali sono stati allenate sui dataset ETH e UCY [3] [40], dove il primo contiene due scene chiamate ETH e HOTEL mentre UCY contiene tre diverse scene chiamate UNIV/UCY, ZARA-01, ZARA-02. Entrambe le scene dei due dataset sono state acquisite da una videocamere che mantiene una prospettiva dall'alto e coinvolgono numerose situazioni difficili, come pedoni che interagiscono, persone in piedi e traiettorie altamente non lineari.

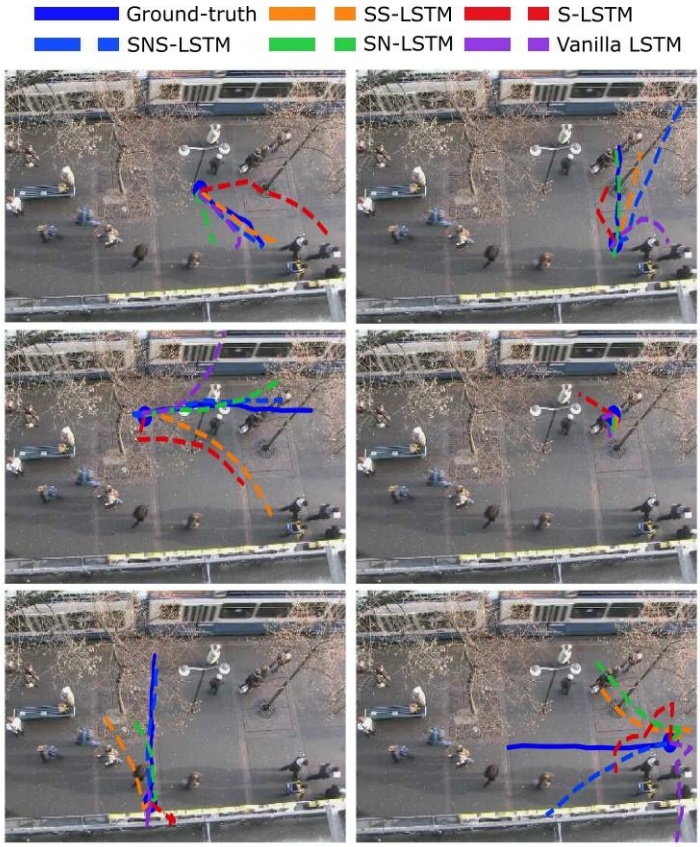


Figura 2.9: Alcuni esempi di traiettorie previste per il dataset HOTEL. La prima colonna mostra i casi in cui le posizioni previste sono molto vicine ai percorsi reali. La seconda colonna mostra i casi in cui l'SNS-LSTM sembra non essere in grado di prevedere correttamente le posizioni future.

Da un punto di vista quantitativo il modello SN-LSTM è un notevole migliora-

mene rispetto al modello S-LSTM che catturava solo le iterazioni sociali tra i pedoni nella scena. Un ulteriore miglioramento si ottiene con il modello SS-LSTM che introduce le informazioni di contesto, tuttavia si è notato come il semplice modello S-LSTM ha prestazioni migliori nella scena ZARA-02 anche rispetto al modello SNS-LSTM. Il motivo di questo fenomeno si può attribuire all'elevato numero di pedoni fermi in questa scena in cui la navigazione e i fattori semantici potrebbero non influenzare molto le previsioni. Comunque, a livello generale, il modello SNS-LSTM ottiene i risultati migliori confermando l'importanza di introdurre fattori di navigazione e semantici per ottenere previsioni più robuste.

Invece, da un punto di vista qualitativo, il modello SNS-LSTM riesce a predire traiettorie più verosimili e precise rispetto agli tipi di rete neurale, come mostrato nella prima colonna della figura 2.9. Invece la seconda colonna mostra alcuni casi dove il modello non è stato in grado di predire correttamente la traiettoria. In generale si è notato come situazioni con pedoni che si fermano bruscamente per poi accelerare subito dopo sono scene complesse che i diversi modelli testati non sono in grado di catturare.

2.3.3 Iterazione spaziale temporale con reti ricorrenti

Predire il percorso futuro delle persone in un ambiente reale è una sfida complessa per varie applicazioni principalmente dovuta alla natura umana, infatti quando un pedone cammina in un ambiente affolato tiene in considerazione sia interazioni spaziali che temporali, con altri pedoni, per evitare collisioni future. I modelli visti in precedenza non tengono in considerazione questi due aspetti contemporaneamente ignorando le correlazioni temporali delle interazioni con altri pedoni coinvolti in una scena.

Recenti lavori propongono metodi per catturare le interazioni spaziali dei pedoni basati su LSTM dove si condividono le informazioni codificati nello stato nascosto delle celle attraverso diverse strategie come il meccanismo di pooling o di attenzione. Mentre il primo metodo, usato in modelli descritti in [5] [32] [24], consiste nel sfruttare le dinamiche di movimento dei pedoni coinvolti in un quartiere locale o nell'intera scena attraverso delle mappe di occupazione. Differentemente il meccanismo di attenzione, implementato in diversi studi come [20] [26] [53] e visto nel paragrafo precedente, permette al modello di concertarsi solo su alcuni stati nascosti, considerati più informativi rispetto ad altri. E' un meccanismo utile per codificare le influenze relative e le potenziali interazioni spaziali tra i pedoni, dato che non tutti i pedoni hanno la stessa importanza, ovvero pedoni vicini contribuiranno mag-

giormente alla previsione delle traiettorie. Rispetto allo metodo di pooling, metodi basati sull'attenzione, assegnando un importanza diversa e adattativa ai pedoni e possono ottenere così una migliore comprensione dei comportamenti spaziali della folla.

Il modello *Spatial-Temporal Graph Attention network (STGAT)*, proposto nello studio [28], permette di utilizzare sia iterazioni temporali e spaziali degli utenti per predire i loro movimenti futuri grazie ad un architettura sequenza a sequenza.

Come viene mostrato dalla figura 2.10, STGAT è un modello complesso formato da tre parti principali:

- Un *Encoder* che è formato a sua volta da altri tre moduli: 2 tipi di LSTM e Graph Attention Network;
- Uno *Strato intermedio* che incapsula le informazioni spaziali e temporali di tutte le traiettorie osservate;
- Un *Decoder* che genera le future traiettorie in base allo Strato intermedio.

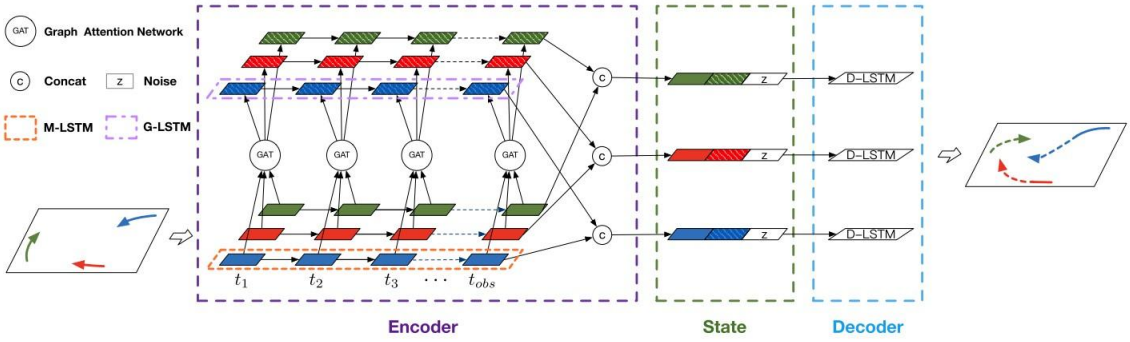


Figura 2.10: Architettura del modello STGAT

Data una scena con $p_1, p_2, ..., p_N$ pedoni con N il numero complessivo di persone presenti, dove la posizione al tempo t di un pedone i viene definita come $S_i^t = (x_i^t, y_i^t)$, allora il modello dovrà predire le future posizioni $S_i^{\hat{t}}$ con $\hat{t} = T_{obs}, ..., T_{pred}$ data una serie di posizioni passate raccolte dall'istante $t = 1, ..., T_{obs}$.

Come fato da altri modelli come [5], [50], [49], l'encoder va a codificare per ogni pedone il suo modello di movimento utilizzando una specifica cella LSTM. Si usa questo particolare tipo di rete ricorrente perché si è dimostrato come riesca a catturare con successo il modello di movimento nello spazio di un determinato pedone. Nell'implementazione di STGAT prima di essere codificate, con l'encoder LSTM, le

singole posizioni delle traiettorie vengono modificate per ottenere le posizioni relative rispetto a quelle dell'istante precedente secondo la formula:

$$\begin{aligned} \Delta x_i^t &= x_i^t - x_i^{t-1} \\ \Delta y_i^t &= y_i^t - y_i^{t-1} \end{aligned} \tag{2.17}$$

Successivamente le posizioni relative vengono codificate in un vettore a dimensione fissa per poi essere utilizzate come input per l'encoder LSTM come segue:

$$m_i^t = M - LSTM(m_i^{t-1}, e_i^t, W_m) \tag{2.18}$$

dove M-LSTM è il nome dato all'encoder LSTM che codifica le informazioni spaziali delle traiettorie dei pedoni, m_i^t è lo stato nascosto al tempo t dell'encoder, e_i^t è il vettore delle posizioni relative codificato e W_m la matrice dei pesi per la cella M-LSTM che viene condivisa per tutti i pedoni della scena.

Purtroppo l'uso di una cella LSTM per persona non permette di catturare le iterazioni tra i diversi pedoni all'interno della scena. Per questo è stata creata una struttura a grafo dove i nodi rappresentano i pedoni e gli archi le loro iterazioni. Quindi il modulo Graph Attention Network consente di aggregare informazioni dai pedoni limitrofi assegnando una importanza differente a nodi diversi. Il modulo GAT opera su pile di grafi di attenzione e la figura 2.11 mostra un singolo grafo di attenzione.

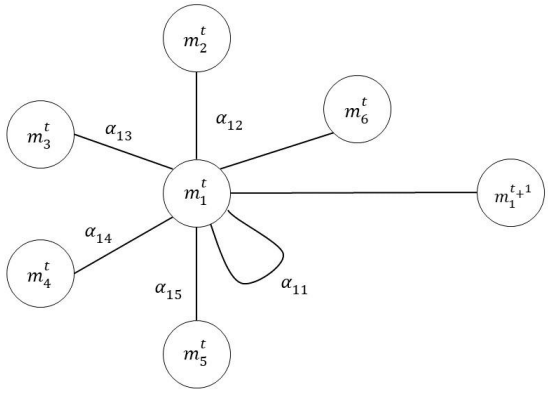


Figura 2.11: Esempio di un livello di attenzione del grafo

Nel periodo di osservazione l'encoder restituisce lo stato nascosto m_i^t dell'istante temporale corrente per il determinato pedone e verrà utilizzato per creare il grafo. I coefficienti del meccanismo di attenzione per una coppia di nodi (i, j) sono calcolati

come:

$$\alpha_{ij}^t = \frac{exp(LeakyReLU(\alpha[Wm_i^t||wm_j^t]))}{\sum_{k\in N} exp(LeakyReLU(\alpha[Wm_i^t||wm_k^t]))} \tag{2.19}$$

dove $||$ è l'operatore di concatenazione di due matrice, W è la matrice dei pesi di una trasformazione lineare condivisa che viene applicata a ciascun nodo e α è un vettore di pesi di un singolo strato di una rete neurale feedforward. I coefficienti di attenzione dei nodi servono per calcolare l'output di un singolo grafo di attenzione al tempo t per un pedone i. Il risultato di un singolo grafo viene espresso dalla seguente formula:

$$\hat{m}_i^t = \sigma(\sum_{j\in N} \alpha_{ij}^t W m_j^t) \tag{2.20}$$

dove σ è una funzione non lineare. Il risultato $\hat{m}_i^t$ esprime l'influenza spaziale di altri pedoni, rispetto al pedone i al tempo t, codificata negli stati nascosti.

Per quanto riguarda le interazioni temporali queste vengono catturata utilizzando un approccio simile usato in precedenza per quelle spaziali ovvero attraverso un encoder LSTM, denominato G-LSTM. Nello specifico le interazioni temporali vengono codificati come:

$$g_i^t = G - LSTM(g_i^{t-1}, \hat{m}_i^t, W_g) \tag{2.21}$$

dove g_i^{t-1} esprime lo stato nascosto della cella LSTM nell'istante temporale precedente e W_g è la matrice di pesi usati dalla cella.

Come visto in precedenza i due encoder sono utilizzati per modellare il movimento di ciascun pedone e le iterazioni nel tempo. Per sfruttare contemporaneamente queste informazioni nella fase di previsione, al tempo T_{obs} le variabili nascoste vengono fuse insieme aggiungendo del rumore, come mostrato dalla seguente formula:

$$d_i^{T_{obs}} = m_i^t||g_i^t||z \tag{2.22}$$

Il risultato ottenuto sarà utilizzato per inizializzare lo stato nascosto di un'altra cella LSTM, che ricevendo in input il vettore $e_i^{T_{obs}}$ delle posizioni del pedone i all'istante T_{obs}, sarà in grado di produrre le posizioni future della traiettorie del pedone.

Per migliorare il risultato della predizione il rumore introdotto in $d_i^{T_{obs}}$ viene

preso da una distribuzione Gaussiana Normale e prendendo k campioni di rumore si va a minimizza la funzione 2.4 in modo da ottenere la previsione più simile alla traiettoria effettivamente compiuta dal pedone.

Per determinare la bontà della nuova architettura proposta il modello STGAT è stato confrontato con diversi modelli presenti in letteratura tra cui:

- La versione standard di LSTM senza meccanismo di pooling dove ogni traiettoria è stata considerata indipendente rispetto alle altre.
- S-LSTM [5] dove ogni pedone viene modellato attraverso l'uso di una cella LSTM e l'influenza della folla viene determinata raggruppato gli stati nascosti dei pedoni vicini per ogni passaggio temporale.
- S-GAN presentato nel paper [24] e riportato nel paragrafo 2.2

Inoltre sia STGAT che i vari modelli di confronto sono stati allenati con i dataset ETH e UCY, descritti nei paragrafi precedenti, che contengono migliaia di traiettorie di pedoni in scenari veri e presentano ricche interazioni tra le persone presenti nella scena. Al fine di paragonare le traiettorie ottenute dai vari modelli sono state utilizzate le metriche ADE e FDE come metodo di confronto per determinare le prestazioni dei modelli.

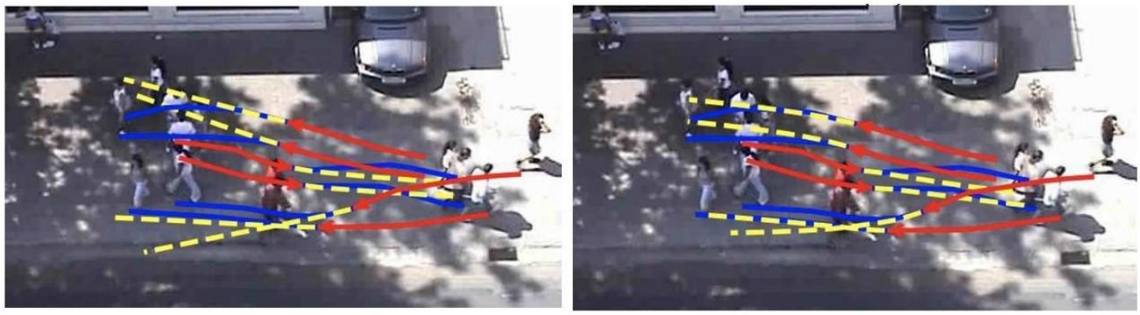


Figura 2.12: Traiettorie predette da S-GAN e STGAT. In rosso la traiettoria osservata fino a T_{obs}, in blue il percorso realmente compiuto dal pedone e in giallo la previsione.

In generale predire le traiettorie dei pedoni è un problema complesso perché si devono tenere in considerazione le proprietà spazio-temporali di ogni pedone presente nella scena. Infatti i pedoni, in scene affollate, possono avere interazioni complesse come formare gruppi, seguire altri pedoni, cambiare direzione per evitare collisioni, ecc. Tuttavia il modello STGAT che utilizza la Variety Loss con k = 20 ottiene buoni

risultati rispetto ad altri modelli. Se si confrontano i risultati delle metriche questo architettura arriva seconda solo a S-GAN che invece ottiene prestazioni migliori. Ciò nonostante, da un punto di vista qualitativo, le traiettorie predette da STGAT risultano essere migliori perchè più vicine a quelle realmente fatte dai pedoni, come mostrato in figura 2.12, confermando che il modello riesce a catturare le iterazioni spazio-temporali tra i soggetti presenti nella scena.

Purtroppo STGAT soffre di due criticità ovvero assegna spesso una importanza elevata a pedoni fermi, probabilmente dovuto al fatto che si utilizzano le posizioni relative come input per il modello, e quando la scena contiene molti soggetti i pesi assegnati ad ogni pedone diventano caotici portando risultati non corretti.

2.4 Trasformer per predizioni

Il Trasformer è un modello di Deep Learning, ideato per dati sequenziali, che adotta il meccanismo della *Self-Attention*, dando diversa importanza a ciascuna parte della sequenza di input. Viene utilizzato principalmente nel campo della Natural language processing e nella Computer Vision.

Come per le reti neurali ricorrenti, i Trasformer sono progettati per gestire dati di input sequenziali come il linguaggio umano e quindi si prestano molto bene per task di traduzione o di riepilogo del testo.

Tuttavia, a differenza delle RNN, i Trasformer non elaborano necessariamente i dati in ordine ma grazie al meccanismo di attenzione viene fornito il contesto per qualsiasi posizione nella sequenza di input. Ad esempio, se i dati di input compongono una frase, il Trasformer non deve elaborare l'inizio della sequenza prima di poter processare la fine ma identifica il contesto che conferisce significato a ciascuna parola nella frase. Questa funzione consente una maggiore parallelizzazione rispetto alle reti neurali ricorrenti e quindi riduce i tempi di allenamento.

I Trasformer sono stati introdotti nel 2017 da un team del Google Brain con lo studio [52] e stanno diventando il modello predefinito per i problemi di Natural language processing sostituendo modelli ricorrenti come LSTM [27]. Inoltre La parallelizzazione dell'allenamento consente l'addestramento su set di dati più grandi portando così allo sviluppo di sistemi pre-addestrati come BERT [18].

Sebbene i Trasformer nascono per task di Natural language processing possono essere utilizzati anche nell'ambito della predizione di traiettorie di pedoni come dimostrato nel paper [23]. Lo scopo di questo studio è di confrontare i risultati ottenuti dai Trasformer rispetto a quelli di un modello ampliamento utilizzato in

questo settore ovvero LSTM. Nello specifico vengono implementati due modelli di Trasformer ovvero la versione originale e Bidirectional Transformer (BERT)[18]. Il modello LSTM si basa sull'elaborazione sequenziale di dati, che appartengono ad una sequenza, e sulla memorizzazione di stati nascosti per rappresentare la conoscenza sui pedoni in una scena, come ad esempio la loro velocità, direzione e modello di movimento. Alcuni approcci più moderni realizzano strategie alternative, basate su LSTM, per catturare l'iterazione tra i vari pedoni come meccanismi di pooling o di attenzione. La fondamentale differenza tra i Trasformer e LSTM è che quest'ultimo modello elabora sequenzialmente le osservazioni prima di iniziare a prevedere, mentre i Trasformer guardano tutte le osservazioni disponibili, pesandole secondo un meccanismo di attenzione.

Le prestazioni dei modelli sono state calcolate sulla *TrajNet Challenge* ovvero una challenge che consiste nel prevedere 3161 traiettorie di pedoni, osservando per ogni traiettoria 8 posizioni consecutivi per un totale di 3,2 secondi e prevedendo le successive 12 ovvero 4,8 secondi. Le traiettorie appartengono ai famosi dataset UCY e ETH. Per paragonare i risultati ottenuti tra i vari modelli vengono calcolate, per ogni traiettoria, le metriche *Mean Average Displacement* e *Final Average Displacement* che sono un equivalente delle metriche ADE e FDE.

I risultati ottenuti dimostrano come I Transformers ottengono buone prestazioni anche nella previsione di traiettoria. Oltre a ottenere le migliori prestazioni sulle previsione nei dataset utilizzati, i Transfomer proposti hanno mostrato un migliore comportamento di previsione anche a lungo termine ovvero su una distanza maggiore di 4,8 secondi. Inoltre sono stati in grado di prevedere più traiettorie future sensate anche con osservazioni di input mancanti, che è un comportamento molto comune in dati prelevati direttamente da sensori nel mondo reale.

Capitolo 3

SCREEN Dataset

In questo capitolo verrà descritto il dataset utilizzano nel progetto di tesi ovvero il dataset sCREEN, quali sono le sue caratteristiche e come è stato creato. Successivamente sarà presentato il lavoro di estrazione delle traiettorie dei vari pedoni, il loro raffinamento e preparazione in modo da essere usate dai vari modelli, di rete neurale, per predire percorsi futuri di pedoni.

3.1 Il progetto sCREEN

Il progetto sCREEN [36] nasce all'Università Politecnica delle Marche e viene seguito in prima persona da Marina Paolanti, Daniele Liciotti, Rocco Pietrini, Adriano Mancini e Emanuele Frontoni. Lo studio ha come obbietto capire il comportamento dei consumatori all'interno di piccole realtà di rivendita al dettaglio e utilizzare queste informazioni per predire le loro azioni successive e suggerirle. In particolare si cerca di trovare quali sono gli scaffali che attirano maggiormente i consumatori e le loro reazioni davanti a determinate situazioni come un prodotto esaurito o modifiche al layout del negozio. Dopo aver stabilito il comportamento generale degli utenti si utilizzano le informazioni raccolte per prevedere quale sarà lo scaffale successivo di maggior interesse per un determinato consumatore viste le sue scelte precedenti. La previsione sarà suggerita all'utente attraverso una mappa del negozio che indica il percorso da percorrere a piedi dalla sua posizione attuale fino allo scaffale consigliato. Inoltre, l'azione proposta conterrà informazioni aggiuntive sui prodotti che sono presenti in quello scaffale e che quindi possono interessare al consumatore.

Negli ultimi anni, è aumentato l'interesse verso lo studio dell'ottimizzazione dello spazio di vendita e nell'assortimento di articoli in quello spazio. Le ragioni sono molteplici e di facile comprensione, infatti dal punto di vista del rivenditore, lo

spazio sugli scaffali è una risorsa importante ma la varietà dei prodotti in vendita aumenta i costi di inventario. Tuttavia avere un buon assortimento è fondamentale per aumentare il numero di consumatori e la loro soddisfazione. Dal punto di vista del produttore, sia la l'assortimento e l'allocazione dei prodotti negli scaffali sono ugualmente fondamentali, infatti questo non può generare un profitto adeguato se i suoi prodotti non sono posizionati correttamente nei vari negozio.

Oltre a ciò il concetto di esperienza di acquisto è cambiata, infatti i negozi oltre ad mantenere la funzione di luogo dove i consumatori si recano per acquistare prodotti sono diventati anche il posto in cui i consumatori trascorrono il loro tempo, testano i prodotti in tempo reale e cercano informazioni sulle ultime tendenze. Con l'avvento e il miglioramento di modelli di intelligenza artificiale, anche in ambienti di vendita al dettaglio, si cerca di dare ai venditori una conoscenza aggiuntiva sui propri consumatori in modo da migliorare la propria attività. Con l'Intelligenza Artificiale il negozio si evolve e permette agli ambienti di essere sensibili e adattabili alla presenza umana.

In letteratura sono già presenti progetti e studi in questo ambito che vanno a sfruttare metodi e modelli di Computer Vision ottenendo già buoni risultati come in [19] [30] [16] dove si è andato a creare un sistema integrato costituito da telecamere RGB-D e software in grado di monitorare gli acquirenti in ambienti di vendita al dettaglio. In questi casi l'obbiettivo dei vari progetti era di creare un innovativo sistema intelligente a basso costo in grado di valutare non solo il comportamento degli acquirenti, ma anche rilevare le loro interazioni con i prodotti negli scaffali attraverso tecniche RGB-D automatiche per l'analisi video.

Tuttavia l'approccio di sCREEN è stato diverso, infatti gli autori erano interessati a migliorare e personalizzare l'esperienza di acquisto attraverso l'istallazione di dispositivi digitali nei carrelli e nei cestini della spesa.

sCREEN è l'acronimo di *Consumer REtail ExperieNce* ed è un sistema meccatronico intelligente per l'assistenza alla navigazione indoor in negozi al dettaglio e non richiede l'utilizzo di mappe metriche ma solo topologiche. L'obbiettivo di sCREEN è di aiutare i consumatori a navigare all'interno di supermercati suggerendoli, in base al percorso fatto finora, dove si trova lo scaffale successivo più consone alle loro esigenze. La previsione delle posizioni future dei consumatori all'interno dei negozi presenta ovvi vantaggi sia per gli utenti stessi che per i rivenditori, infatti permette di risparmiare tempo, migliora l'esperienza dei clienti, semplifica le vendite e rende più facile la navigazione. Inoltre i consumatori possono fare affidamento su mappe interne personalizzate per ottenere indicazioni dettagliate all'interno del

negozio, trovando facilmente categorie di prodotti specifiche.

Gli ambienti di vendita al dettaglio, come i supermercati, hanno traiettorie già strutturate per i consumatori infatti, i negozi sono generalmente costituiti da corsie con prodotti collocati in scaffali. Inoltre, i prodotti possono essere messi in spazi diversi dagli scaffali come isole o lungo il perimetro del negozio. Il posizionamento dei prodotti limita la traiettoria del consumatore a percorsi approssimativamente unidirezionali all'interno dei corridoi o lungo il perimetro, questa caratteristica porta i nugozi e supermercati ad essere simili tra di loro. La figura 3.1 rappresenta il layout del supermercato in cui il sistema sCREEN opera. Nella figura vengono evidenziata in grigio gli scaffali mentre i cerchi rossi rappresentano le antenne posizionate nel controsoffitto del negozio che servono per rilevare la posizione dei carrelli o cestini della spesa degli utenti.

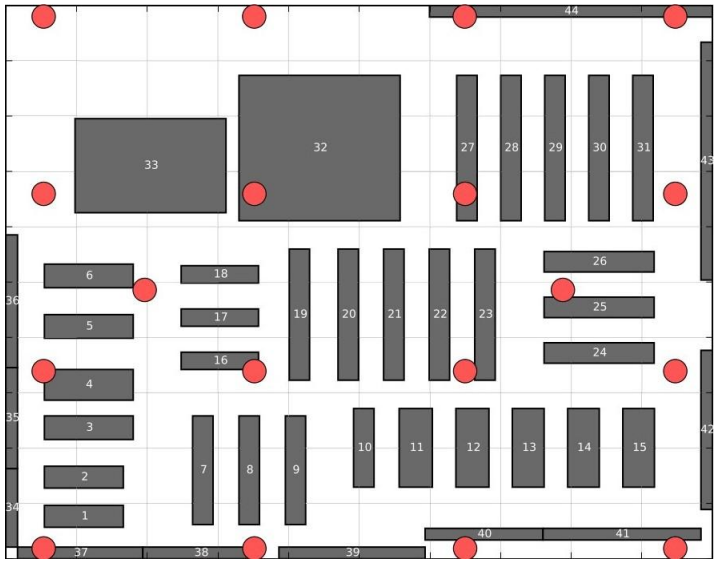


Figura 3.1: Layout del negozio e posizionamento degli scaffali. I 18 punti rossi rappresentano le antenne posizionati nel controsoffitto del negozio.

sCREEN è un sistema complesso formato da diverse parti, sia hardware che software, che cooperano tra loro:

- *Componente meccanica*: dispositivi elettronici installati direttamente nei carrelli della spesa e nei cestini, con la funzione di tracciare i clienti durante la loro esperienza di acquisto.

- *Dispositivi incorporati*: componenti elettronici montati sui carrelli della spesa e sul cestino per dare le informazioni all'acquirente. Questi dispositivi sono stati

sviluppati utilizzando un Raspberry Pi 3 Modello B al quale viene collegato un display LCD.

- *Hidden Markov model*: è il modello, che utilizzando i dati relativi alla posizione degli acquirenti nel negozio, permette di stabilire i punti di attrazione ovvero gli scaffali che saranno maggiormente visitati.

- *Componente automatico*: ha il compito di capire il tipo di cliente corrente e predire, grazie al modello, lo scaffale successivo di maggior interesse, dopodiché in base alla posizione del carrello è in grado di fornire le informazioni sul percorso ottimale dall'acquirente allo scaffale mostrandole sul display.

Il sistema sCREEN utilizza la tecnologia UWB per monitorare le posizioni dei consumatori nei negozi e inviare i dati di tracciamento raccolti a un server cloud. La tecnologia *Ultra Wideband* [48] conosciuto con l'acronimo UWB, è una tecnologia per la trasmissione wireless di dati. La sua caratteristica principale è che permette di sfruttare un'ampia banda di frequenze per trasmettere grandi pacchetti dati a corto raggio, infatti la distanza massima di trasmissione in condizioni ottimali è di circa 70 metri, con il vantaggio di non essere troppo energivora. A differenza di tecnologie simili, come quella Bluetooth, soffre molto meno la presenza di pareti o ostacoli lungo la traversata del segnale dalla sorgente alla destinazione. Questa qualità la rende una tecnologia ottima per essere usata all'interno di luoghi chiusi come abitazioni, uffici o centri commerciali. L'utilizzo di questa tecnologia ha permesso di raggiungere una precisione nella misurazione delle posizioni di 20 cm in termini di localizzazione indoor e inoltre, un'elevata autonomia per i tag collegati ai carrelli che sono alimentati a batteria. Questi risultati non sarebbero stati raggiunti da tecnologie simili come RFID o WLAN. Il sistema di tracciamento che è stato sviluppato per sCREEN prevedeva l'utilizzo di diversi dispositivi fisici:

- *Antenne*: dispositivi statici con posizione conosciuta posti sul controsoffitto per formare una griglia omogenea in modo da coprire l'intero negozio. I dati raccolti dalle antenne come tempo di arrivo del segnale, id del tag e il suo stato vengono direttamente indirizzati al server.

- *Tag*: sono i dispositivi mobili da tracciare e inviano dati alle antenne a specifici intervalli temporali. Al fine di prolungare la durata della batteria i vari tag sono stati dotati di accelerometro per stabilire se il carrello è in movimento. Da specificare che un determinato tag non trasmette ad una singola antenna ma invia i suoi dati a tutte le antenne in modo che il server riesca a stimare la sua posizione.

- *Server*: ha il computo di raccogliere dati dalle antenne e stimare la posizione del tag guardando il livello di potenza del segnale e il suo tempo di arrivo.

Grazie a questo sistema è stato possibile creare un dataset di posizioni denominato *Dataset sCREEN*. Il dataset è formato da 4622438 di punti che rappresentano le posizioni di carrelli o cestini rilevate nell'arco del mese di luglio 2017 in un supermercato tedesco. Per ogni posizione si raccolgono le coordinate cartesiane x, y e z all'interno del supermercato, il tempo della misurazione e il codice identificativo del carrello o cestino, in più viene dato un identificativo univoco alla stessa posizione. L'intero dataset è stato strutturato come un continuo flusso di punti ordinato temporalmente dove non viene specificato se un determinato tag è stato applicato ad un carrello o ad un cestino.

3.2 Dataset simili

Negli ultimi anni, in ambito di Computer Vision, si è affrontato molto il tema della *predizione di percorsi* non solo di robot ma anche di persone, questo perché stanno nascendo promettenti applicazioni come smart surveillance, auto a guida autonoma, navigazione robotica autonoma ecc. per le quali questo task è fondamentale.
A differenza di altri task come ad esempio rilevamento e tracciamento di pedoni dove le osservazioni dal passato al presente vengono utilizzate per individuare e tracciare il bersaglio nel fotogramma corrente del video, nel caso del task di predizione di traiettorie le osservazioni fatte fino a quel momento vengono usate per stimare la posizione futura del pedone.
Affinché un modello riesca a realizzare una buona predizione su un possibile percorso futuro di una pedone, deve avere a disposizione molte informazioni come dati sull'ambiente circostante e sulla direzione di movimento del bersaglio. Questa tipologia di informazione è molto diversa rispetto a quella richiesta per un normale compito di riconoscimento di immagini infatti, nel maggior parte dei casi, deve essere estratta dai dati. Per questo motivo i task di predizione spesso si basano su altri task di Computer Vision come rilevamento di pedoni, riconoscimento degli attributi dei soggetti e semantic segmentation ovvero la categorizzazione in classi degli oggetti presenti nella scena.

Fondamentalmente un modello di predizione di traiettorie ha bisogno di 2 categorie di informazione in particolare:

- *Caratteristiche dell'ambiente circostante.* Un pedone decide la direzione e percorre una specifica strada in base all'ambiente circostante. Ad esempio, nella maggioranza dei casi i pedoni camminano lungo il marciapiede ed evitano gli ostacoli lungo il loro percorso. Quindi il movimento del bersaglio è influenzato dinamicamente dall'ambiente e dalle sue caratteristiche, perciò tali informazioni devono essere estratte dal video. Per estrarre queste informazioni, tipicamente si utilizza un task di *Semantic segmentation* che permette di assegnare ad ogni pixel o gruppi di pixel nell'immagine, una classe di appartenenza. Questa suddivisione in categorie è essenziale per determinare dove esistono ostacoli nella scena e quali sono le aree dove un pedone può camminare.

- *Caratteristiche del soggetto.* Oltre alle caratteristiche dell'ambiente, che vanno a modificare il percorso del pedone, bisogno tenere in considerazione anche alcuni attributi del pedone come ad esempio sesso ed età. Infatti, quando molti pedoni camminano insieme nello stesso luogo questi tendono a compiere azioni che evitano scontri con altri pedoni. Tuttavia le persone non sono uguali e quindi una pedone giovane camminerà più velocemente e avrà dei riflessi più rapidi rispetto ad uno più anziano.
  Un altra caratteristica del bersaglio, che viene ricercata più comunemente, è l'orientamento questo perché, una volta stimato, può essere utilizzato per prevedere in quale direzione sta andando, diminuendo così l'errore di previsione.

Nel corso degli anni molti dataset sono stati usati per valutare modelli di predizione di traiettorie. La differenza sostanziale tra di loro è il numero di scene e come queste vengono riprese, infatti non esiste un dataset universale adatto per tutti i modelli ma un insieme di dati risulta essere più adatto per un determinato compito.

I dataset più usati sono quelli creati con riprese di grandi spazi aperti come stazioni o mercati, e vengono realizzati tipicamente da telecamere con lenti grandangolari come quelle di videosorveglianza. Questi dataset vengono solitamente utilizzati per valutare i metodi di tracciamento dei pedoni, tuttavia, vengono anche utilizzati per altri task come quelli di predizione del percorso. Esempi di dataset sono:

- *Grand Central Station Dataset* [41] contiene le traiettorie di 12684 pedoni che sono stati ripresi da un videocamera posizionata nella Grand Central Station di New York. I percorsi dei pedoni sono stati estratti da un video della durata di 33 minuti con un frame rate di 25fps ad una risoluzione di 720x480 pixel.

- *VIRAT Video Dataset* [39] è un grande insieme di dati creato per fornire un grande dataset per valutare le prestazioni di diversi algoritmi di riconosci-

mento di eventi visivi con particolare attenzione sul task di *Continuous visual event recognition (CVER)* in aree esterne ad ampia copertura. Nello specifico questo task consiste sia nel riconoscere un evento che localizzare nel piano spazio-temporale di un grande video continuo. Il dataset contiene diversi video ripresi da telecamere di sorveglianza posizionate in vari parcheggi con angoli di ripresa differenti. Per ciascun video vengono fornite le posizioni di ogni pedone, automobile e oggetto nella scena, inoltre le azione fatte dai pedoni vengono classificate con una etichetta per fornite più informazioni durante l'allenamento dei modelli. Il dataset è stato creato per essere il più realistico possibile in termini di risoluzione, disordine dello sfondo, diversità nelle scene e categorie di attività/eventi umani.

Un altra tipologia di dataset sono i *Top view* dove le traiettorie dei pedoni vengono prese da scene riprese dall'alto. Tipicamente le scene riprese contengono video di pedoni che camminano lungo una strada dove non ci sono altri oggetti in movimento. Questa caratteristica rende la scena molto semplice aiutando l'apprendimento dei modelli. Esempi di dataset sono:

- *UCY Dataset* [3] contiene due scene per un totale di 786 pedoni ed è diviso in tre parti: ZARA-01, ZARA-02 e UNIV. In origine è stato creato per uno studio sullo sviluppo di folle virtuali che potessero essere usate in vari ambiti come nell'industria cinematografica o per la creazione di videogiochi per computer. L'obbiettivo dello studio era quello di realizzare un modello di folla il più realistico possibile che prendesse i suoi comportamenti da osservazioni reali e non da regole imposte a priori.

- *ETH Dataset* [40] è diviso in due principali parti: ETH e HOTEL, ogni sezione contiene due scene con 750 pedoni ciascuna. L'insieme di dati è stato raccolto per una ricerca sul miglioramento del task di tracciamneto degli oggetti. In particolare lo scopo dello studio era quello di verificare se tenere in considerazione il comportamento umano dei pedoni in un ambiente affollato potesse migliorare le prestazioni di tracciamento.

- *Edinburgh Informatics Forum Pedestrian Database* [9] raccoglie più di 1000 traiettorie di pedoni estratte da riprese catturate da una telecamere fissa posizionata all'interno dell'università di Edimburgo.

- *Stanford Drone Dataset* [4] è composto da otto scene catturate da diversi droni posizionati attorno agli otto siti dell'università di Stanford. Inoltre sono

fornite annotazioni di oggetti in movimento, come ciclisti, skateboarder e automobili, nonché pedoni. A differenza dei precedenti dataset che sono stati ideati principalmente per task di monitoraggio di pedoni o analisi di comportamenti della folla questo è stato realizzato principalmente per predizione di traiettorie di pedoni.

Un categoria di dataset usata sono i *Car-mounted cameras* che prevede che i video dei dataset siano raccolti da telecamere poste nella parte anteriore dell'auto. Quindi l'obiettivo principale è prevedere i percorsi dei pedoni davanti all'auto. Questi dataset vengono molto usati per sviluppare e migliorare la guida autonoma. Alcuni dataset di questa tipologia sono:

- *Daimler Pedestrian Path Prediction Benchmark Dataset* [35] è formato da 68 sequenze di immagini di pedoni dove, 55 sequenze sono state registrate a velocità del veicolo comprese tra 20 e i 30 km/h, le altre invece dove il veicolo era fermo. Le varie sequenze sono formate da fotogrammi che contengono uno o più pedoni. Ogni sequenza e stata classificata in quattro distinte classi che vanno a rappresentare diverse azioni dei pedoni. In origine il dataset era stato creato per uno studio nell'ambito dei veicoli intelligenti per comparare diversi filtri bayesiani ricorsivi per la previsione del percorso pedonale a brevi orizzonti temporali.

- *KITTI Vision Benchmark Suite* [1] è stato costruito per applicazioni di *Intelligent Transport System* ovvero tutte quelle applicazione che mirano a fornire servizi innovativi relativi alle diverse modalità di trasporto e gestione del traffico. Consente, inoltre, agli utenti di essere meglio informati e rendere più sicuro e coordinato l'uso delle reti di trasporto. Viene utilizzato per varie valutazioni di modelli di rilevamento di pedoni, veicoli e linee bianche sulla strada. Contiene non solo immagini RGB ma anche immagini stereo, dati LIDER 3D, posizioni GPS e mappe stradali, ed è quindi utile per la previsione del percorso che utilizza informazioni ricche per comprendere l'ambiente.

Una particolare tipologia di dataset è quella *First-person view* dove a differenza dei dataset precedenti in cui i video delle varie scene erano riprese da telecamere esterne o montate su automobili, in questi i video vengono ripresi con una visuale in prima persona e vengono utilizzati per prevedere il percorso della persona che sta effettivamente registrando il video.

Come visto in precedenza esistono diverse categorie di dataset che si differenziano non solo per il numero di video che contengono ma anche come questi vengono

acquisiti e per la presenza o meno di informazioni aggiuntive come ad esempio coordinate GPS. Tutte queste informazioni verranno poi utilizzate dai vari modelli di predizione per predire una traiettoria di un possibile pedone.

3.3 Dati preliminari

Nel dataset sCREEN sono state raccolte 4622438 posizioni da 175 carrelli o cestini. Le misurazioni cominciano il 1 Luglio 2017 alle 3:03 del mattino e terminano il 30 Luglio 2017 alle 23:55. Le coordinate dei punti sono in valore assoluto ovvero senza unità di misura. Inoltre sebbene è presente la coordinata z questa rimane, per ogni punto presente nel dataset, nulla quindi si è provveduto a rimuoverla dato che non porta nessuna informazione utile. Da una analisi preliminare si è scoperto che le coordinate x variano tra -3,63 a 49,7 mentre le coordinate y variano da un minimo di -3,08 e ad un massimo di 36,97. Il fatto che ci siano coordinate negative fa pensare che il dataset sia abbastanza rumoroso e quindi che qualche misurazione sia fuori scala.

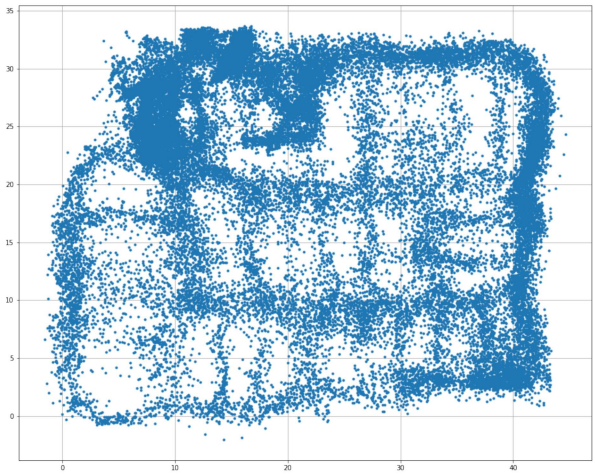


Figura 3.2: Plot dei primi 100000 punti del dataset

Nella figura 3.2 vengono rappresentati sul piano cartesiano le prime 100000 posizioni presenti nel daataset. Come si può vedere ci sono regioni dove è presente una più alta concentrazione di punti rispetto ad altre che sono vuote. Questo fa pensare che nelle parti con meno punti ci sono degli ostacoli che i consumatori non possono attraversare e che quindi sono costretti ad aggirare. Confrontando questa immagine con la figura 3.1 si nota che in queste regioni sono presenti gli scaffali.

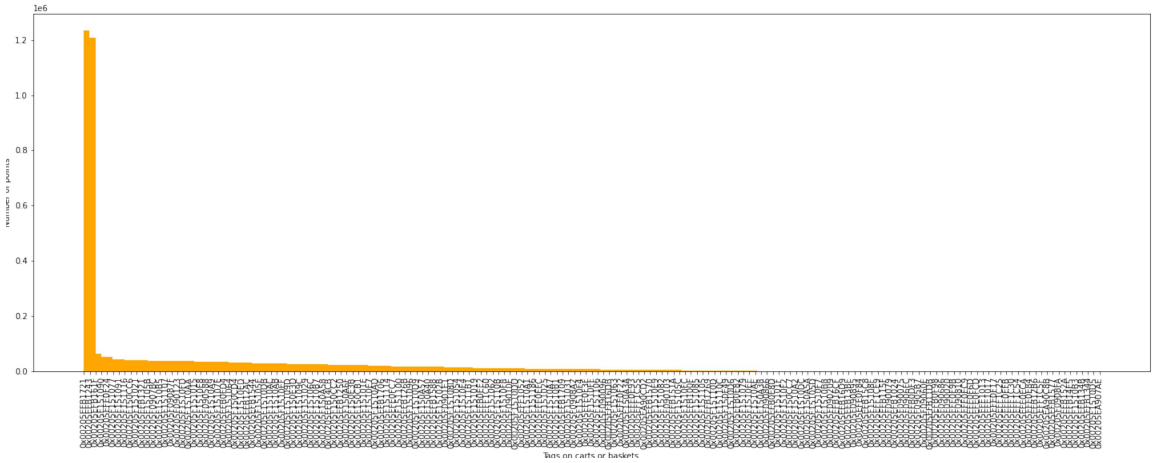


Figura 3.3: Distribuzione del numero di punti raccolti per ogni carrello o cestino. Nelle ascisse l'id dei tag mentre nelle ordinate il numero di punti raccolti espresso in 10^6

Dopo una prima analisi preliminare l'elevato numero di posizioni raccolte fu associato al grande numero di carrelli presenti nel supermercato e all'esteso periodo di campionamento utilizzato per raccogliere le posizioni. Tuttavia come mostrato nella figura 3.3 c'è una distribuzione completamente sbilanciata nell'uso dei carrelli o cestini. In particolare emergono due tag che registrano un numero decisamente superiore di posizioni raccolte, nell'arco del mese, rispetto a tutti gli altri. Infatti i tag *0x00205EFB1721 e 0x00205EFB1243* registrano rispettivamente 1234224 e 1208417 posizioni, un valore troppo elevato se paragonato al numero di posizioni raccolte dal terzo tag più numeroso ovvero il tag *0x00205EFB161F* che registra solo 62310 punti. Dopo uno studio più approfondito i due tag anomali si possono definire rumorosi, infatti come si può vedere dalla figura 3.4, che rappresenta il plot delle posizioni raccolte dal tag *0x00205EFB1243*, molti punti si trovano sopra a delle regioni dove i consumatori non possono andare, lo stesso accade per il tag *0x00205EFB1721*. In aggiunta, a differenze degli altri tag, il sistema sembra non essere in grado di capire quando questi due sensori sono fermi nella stessa posizione per evitare di registrare punti inutili. Questo fenomeno porta alla formazione di cluster di migliaia di punti nella stessa posizione.

Inoltre ci sono ben 40 tag che hanno registrato solo una posizione per tutta la durata del mese, questo fa presumere che questi carrelli o cestini non siano mai stati usati e che l'unico punto registrato sia una posizione rumorosa e quindi possono essere scartati. Infine sono presenti 5 tag che hanno riportato un numero di punti compreso tra 2 e 100. Valori da considerarsi anomali perchè o indicano che il carrello preso in esame è stato usato solo una volta oppure i punti raccolti sono malfunzionamenti dei sensori.

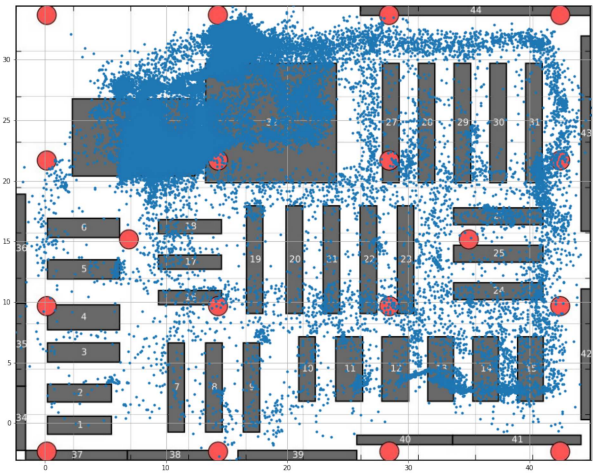


Figura 3.4: Plot dei punti del tag 0x00205EFB1243 sulla mappa del supermercato

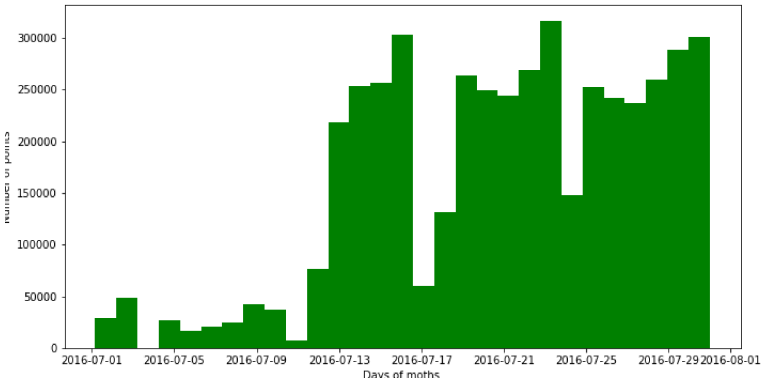


Figura 3.5: Distribuzione dei punti durante i giorni del mese

Un altra analisi condotta sul dataset è lo studio della distribuzione dei punti nei giorni di Luglio. Come si può vedere dalla figura 3.5 i punti non sono distribuiti equamente tra i vari giorni del mese. Infatti nei primi 10 giorni si registrano poche posizioni, addirittura solo 653 nel terzo giorno, mentre già dall'undicesimo giorno si sono raccolte più di 210000 posizioni. Il giorno con più misurazioni risulta essere il 23 Luglio con 251929 punti.

Per quanto riguarda la distribuzione dei punti durante le ore del giorno sembra meno anomala. Infatti come mostrato in figura 3.6 c'è una concertazione di punti nelle ore centrali della giornata, ovvero dalle 9 alle 17, facendo pensare ad un aumento di consumatori. Dal grafico, inoltre, emerge che il supermercato, usato per realizzare il dataset sCREEN, sia aperto 24 ore su 24 dato che anche durante le ore notturne si continua a registrare traiettorie dei consumatori.

Nel grafico in figura 3.7 si può notare come le misurazioni siano più o meno

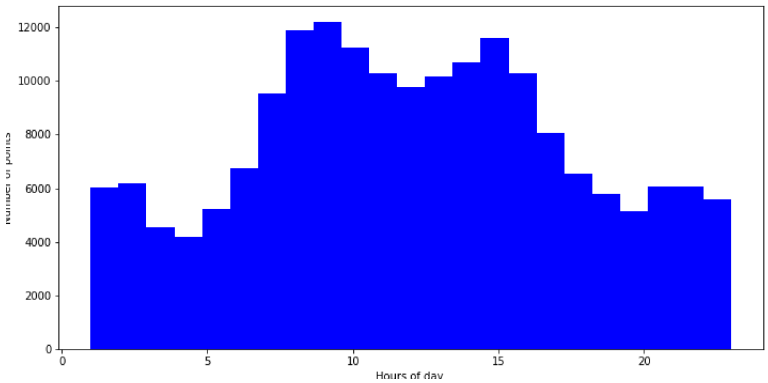


Figura 3.6: Distribuzione media dei punti durante le ore del giorno per tutto il mese

conformi rispetto ad una affluenza tipica in un supermercati ovvero con meno clienti durante la settimana e un maggiore afflusso di consumatori durante il weekend. Tuttavia, in questo caso, sembra che il giorno con meno posizioni raccolte sia la domenica.

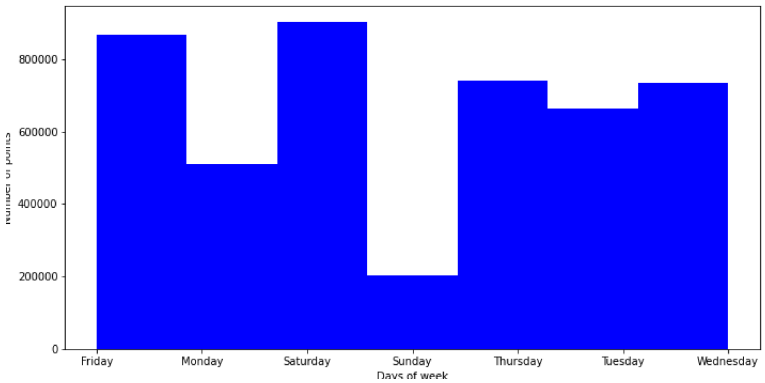


Figura 3.7: Distribuzione dei punti durante i giorni della settimana per tutta la durata del mese di Luglio

3.4 Estrazione delle traiettorie

Dopo una prima analisi emergono tre grosse criticità dal dataset sCREEN:

- E' un dataset molto rumoroso, quindi le posizione raccolte non sono precise;
- E' strutturato come un flusso continuo, a livello temporale, di punti quindi non c'è modo di sapere dove inizia la traiettoria di un determinato consumatore e dove finisce.
- Le rilevazioni delle posizioni degli utenti all'interno del supermercato non sono ad intervalli regoli sebbene il carrello o cestino associato è in movimento.

Il fatto che la traiettorie non siano già identificate nel dataset implica che devono essere estratte. Per questo sono stati sviluppati e testati una serie di metodi diversi per l'estrazione delle traiettorie.

3.4.1 Approccio Data missing

Questo approccio prende spunto da un suggerimento dagli autori del sistema sCREEN che consigliano di spezzare una traiettoria in due diverse se, all'interno di quest'ultima, c'è un intervallo temporale di almeno 5 minuti tra due punti consecutivi. Questo significa che il carrello o cestino è stato immobile per almeno 5 minuti e quindi il precedente cliente la lasciato e un altro consumatore può averlo preso. Il metodo, sebbene semplice, funziona infatti riesce ad estrarre ben 187 traiettorie dai 131 carrelli.

Purtroppo i dispositivi installati nei carrelli o cestini che comunicano al sistema sCREEN quando il carrello è in movimento, grazie ad un accelerometro, non sono molto precisi. Infatti quello che si è potuto notare è che i sensori non riconosco sempre che il carrello è fermo nella stessa posizione e quindi inviano un segnala al sistema in modo da rilevare la loro posizione. Un caso abbastanza frequente che si è potuto osservare, è quando il sensore inizialmente funziona correttamente rilevando che il carrello, dove è stato installato, è in movimento perché un utente sta navigando all'interno del negozio. Successivamente quando l'utente si ferma il tag smette di inviare segnali al sistema in modo da non raccogliere posizioni uguali. Questa fase dura solo per un certo intervallo di tempo, dopodiché il sensore ricomincia ad inviare segnali sebbene il carrello è fermo, quindi il sistema sCREEN va a registrare una serie di posizioni troppo ravvicinate tra di loro o addirittura sovrapposte. Dopo questo periodo di durata variabile il tag continua a funzionare correttamente.

Un altro caso notato è quando il tag è completamente difettoso ovvero non è in grado di capire quando il carrello è fermo e quindi continua a mandare segnali in modo che il sistema rilevi la sua posizione.

Questi comportamenti non sono associati solo ai tag *0x00205EFB1721* e *0x00205EFB1243*, che sono stati considerati i più rumorosi per il loro numero anormale di punti raccolti, ma anche ad altri sensori.

Questi fenomeni e l'utilizzo del metodo basato su data missing per estrarre le traiettorie genere tre tipi di percorsi: traiettorie realistiche, cluster di punti disordinati e traiettorie troppo lunghe per poter essere associate ad un singolo utente.

Per individuare i cluster di punti è stato creato un altro metodo che usa la *deviazione standard*. L'idea si basa sulla definizione di cluster ovvero un ammasso di oggetti ravvicinati tra di loro. Se i punti di una traiettoria sono equamente distribuita nel piano allora significa che può essere associata ad un percorso realistico che ha fatto un utente e quindi la sua deviazione standard, che indica il grado di dispersione dei punti, sarà elevata. Viceversa se la traiettoria analizzata è un cluster di punti allora questi dovranno essere vicini tra di loro o addirittura sovrapposti, perciò la sua deviazione standard sarà molto bassa. Il metodo usa un valore di deviazione standard per capire se la traiettoria analizzata è oppure no un cluster.

La threshold usata è stata ricavata empiricamente dai dati del dataset, in particolare è stata determinata dai punti del tag *0x00205EFB1721* che, dall'analisi preliminare, oltre a risultare il tag con più punti era anche estremamente rumoroso. Il procedimento ha previsto prima la separazione dei punti del tag in tante traiettorie con l'approccio data missing e poi, per ogni traiettoria, si è calcolata la deviazione standard per le coordinate cartesiane. Per ogni asse è stato scelto un valore di deviazione standard che permettesse di discriminare abbastanza bene tra un traiettoria possibile e un cluster di punti. Infine è stata selezionata un unica threshold che risulta essere la media di questi due valori. Lo studio ha provato che non tutte le traiettorie erano cluster infatti, da 187 traiettorie si sono scartate solo 68 percorsi che sono stati considerati come cluster, per un totale di 119 traiettorie effettive.

3.4.2 Approccio Euclidean distance

L'appoccio Data missing con il metodo di rimozione delle traiettorie cluster permette di ottenere buoni risultati, tuttavia non spezza le traiettorie che sono troppo lunghe per essere fatte da un solo utente. Inoltre ha rigidezza eccessiva, infatti una nuova traiettoria non per forza deve iniziare dopo un intervallo di 5 minuti tra due punti consecutivi, ma banalmente un consumatore può aver preso lo stesso carrello di un altro cliente appena questo lo ha lasciato. Sotto l'ipotesi che le traiettorie provenienti dall'approccio precedente si potessero suddividere nuovamente si è sviluppato e testato un ulteriore metodo per l'estrazione delle traiettorie per migliorare i precedenti risultati.

L'approccio che è stato sviluppato si basa sull'utilizzo della *Distanza euclidea* tra punti per identificare regioni della traiettoria dense e quindi candidabili come aree dove fare la suddivisione in sotto-traiettorie. La distanza euclidea viene definita

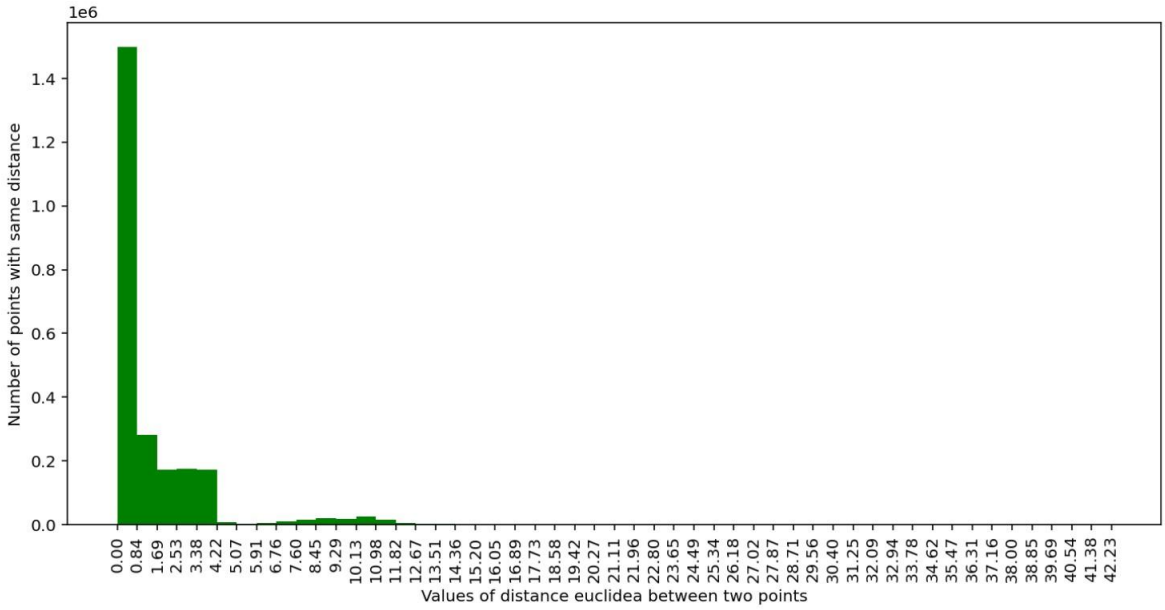


Figura 3.8: Distanze euclidea tra i punti del tag 0x00205EFB1721 e 0x00205EFB1243. Nell'asse x i valori delle distanze e nell'asse y il numero di punti con la stessa distanza

dall'equalizzazione:

$$dist(i,j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{3.1}$$

dove i, j indicano due punti all'interno del dataset.

L'idea è di determinare regioni della traiettoria dove i punti sono estremamente vicini tra di loro in un determinato intervallo di tempo. Preso un punto come centro della regione si vanno a identificare tutti i punti limitrofi, a livello temporale dal punto centrale, che non sono più distanti da questo rispetto ad una determinata soglia. Se questa regione si sviluppa in un intervallo temporale di 5 minuti allora si considerano i punti appartenenti a quest'area come punti rumorosi dove in realtà il carrello è rimasto fermo e quindi possono essere rimossi. Inoltre quest'area farà da divisore permettendo di dividere la traiettoria di partenza in due parti.

La distanza euclidea opportuna, per determinare queste zone, è stata scelta empiricamente, infatti se preso un valore troppo grande si sarebbero individuate regioni ampie che contenevano punti dove effettivamente il carrello era in movimento, viceversa con un valore troppo piccolo si sarebbero prese zone con solo punti sovrapposti. Per fare questo sono studiati i valori di distanza euclidea dei punti registrati dai tag *0x00205EFB1721* e *0x00205EFB1243* perchè si è potuto notare che avevano sia regioni estremamente dense di punti sovrapposti sia zone dove i punti sono molto vicini tra di loro.

Come si può notare dalla figura 3.8 i punti di questi tag, ma in generale anche di tutti gli altri, oltre a non essere stati registrati ad intervalli di tempo regolare, non è presente nessuna regolarità nella distanza. Paradossalmente ci sono dei casi dove la posizione di un utente è stata registrata da un lato di uno scaffale e la successiva, presa dopo un secondo, dall'altra. Come si può notare dal grafico questi tag hanno la maggioranza dei punti sovrapposti tra di loro, infatti la distanza euclidea, per questi punti, è uguale a 0. Inoltre, si può avere tra due punti consecutivi una distanza paragonabile alla grandezza del supermercato portando la varianza della distanza euclidea troppo variabile. Infine è stata scelta una sogli uguale a 1 perché si sono considerati i primi due valori di distanza ovvero 0 e 0.84 come valori anormali e quindi validi per definire aree dense di punti.

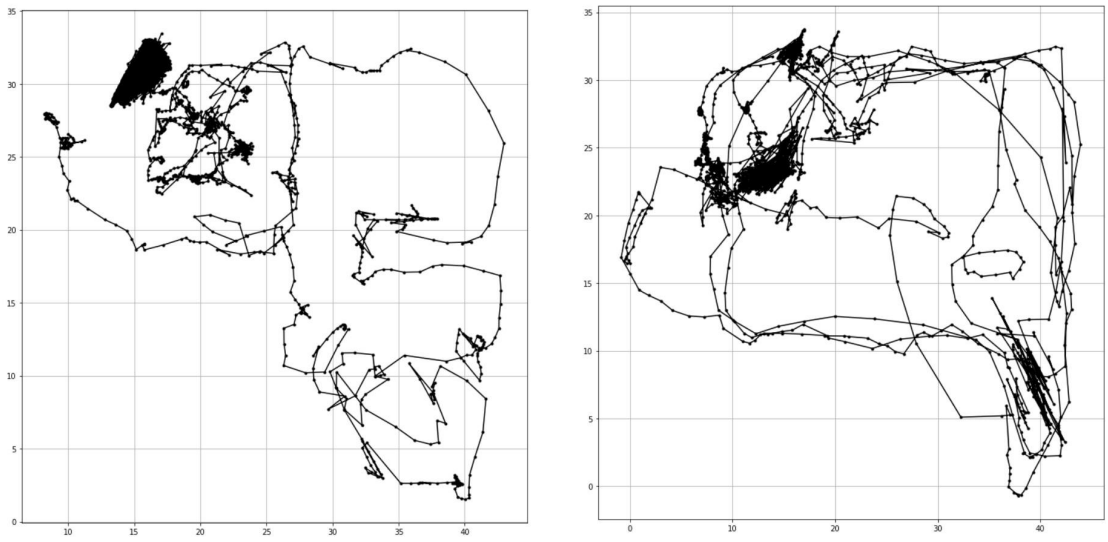
Valutando le traiettorie provenienti da questo metodo, si è notato che alcune di loro avevano un numero di punti troppo piccolo per essere considerate valide, pertanto si è deciso di rimuovere tutte quelle con un numero di punti inferiore o uguale a 150. Inoltre, dato che determinare la distanza euclidea tra punti consecutivi appartenenti ad un traiettoria è una operazione computazionalmente lunga, dato il numero di punti elevato, per aiutare il calcolo si sono rimossi tutti quei percorsi prevenienti dall'approccio data missing che avevano un numero di punti inferiore o uguale a 150.

La strategia per identificare i percorsi degli utenti in questa fase prevedeva: una estrazione preliminare delle traiettorie primordiali con l'approccio data missing, in seguito venivano rimosse tutte le traiettorie che venivano considerate sia come cluster di punti sia troppo piccole per essere valide. Successivamente le traiettorie rimaste si suddividevano ulteriormente andando ad individuare zone di punti dense. Dopo quest'ultima fase potevano ripresentarsi percorsi cluster o troppo corti che sono stati quindi eliminati. Questa strategia ha permesso di aumentare il numero di traiettorie da 119 a 4643.

3.4.2.1 Risultati ottenuti

La strategia adottata per estrarre le traiettoria permette di ottenere un gran numero di sotto percorsi, tuttavia non è da considerarsi un metodo ottimale. Infatti dopo un studio condotto sulle traiettorie estratte emergono delle problematicità che inizialmente questa strategia avrebbe dovuto togliere ovvero la rimozioni di zone cluster che non permettono un adeguato allenamento dei modelli e la suddivisione di traiettorie molto lunghe nel tempo in sotto percorsi.

La figura 3.9 mostra due esempi di traiettorie anomali emerse da questo approccio. Nella immagine 3.9a viene rappresentata la traiettorie n° 141 dove, si può vedere, nella sua parte iniziale, ovvero nell'angolo in alto a sinistra, una grande zona densa di punti. La traiettorie possiede quasi 27000 posizioni di cui poco più di 20000 sono in questa regione. Inoltre un altra caratteristica particola di questa traiettoria è la sua durata ovvero la prima posizione è stata registrata il 29 luglio 2016 alle 01:01:39 mentre l'ultimo punto è stato osservato nello stesso giorno ma alle 09:30:11, per una durata complessiva di 8 ore e 28 minuti. In particolare il tag associato a questo carrello o cestino è rimasto nella zona densa di punti per quasi 8 ore per poi muoversi. Invece, nella traiettoria n° 122, mostrata nella figura 3.9b sono presenti ben 3 aree dove i punti si concentrano ma a differenza della precedente dove c'era un unico cluster all'inizio della traiettoria, in questo caso sono posizionati anche più internamente.



(a) Traiettoria n° 141 **(b)** Traiettoria n° 122

Figura 3.9: Traiettorie dove sono presenti ancora zone dense di punti

Come riportava il grafico 3.5 che evidenziava giorni del mese dove si registravano più posizioni rispetto ad altri, con una tendenza ad avere più punti verso la fine del mese, questa caratteristica si è trovata, di conseguenza, anche nel numero di traiettorie estratte per ogni giorno. Il grafico 3.10 mostra come nei giorni 3, 10, 17, 24 di luglio non siano state estratte nessuna traiettoria sebbene nel dataset fossero presenti dei punti in alcuni di questi giorni. Inoltre, dato che il 23 luglio è stato il giorno con più posizione raccolte, risulta essere anche il giorno con il numero maggiore di traiettorie estratte, ovvero 356.

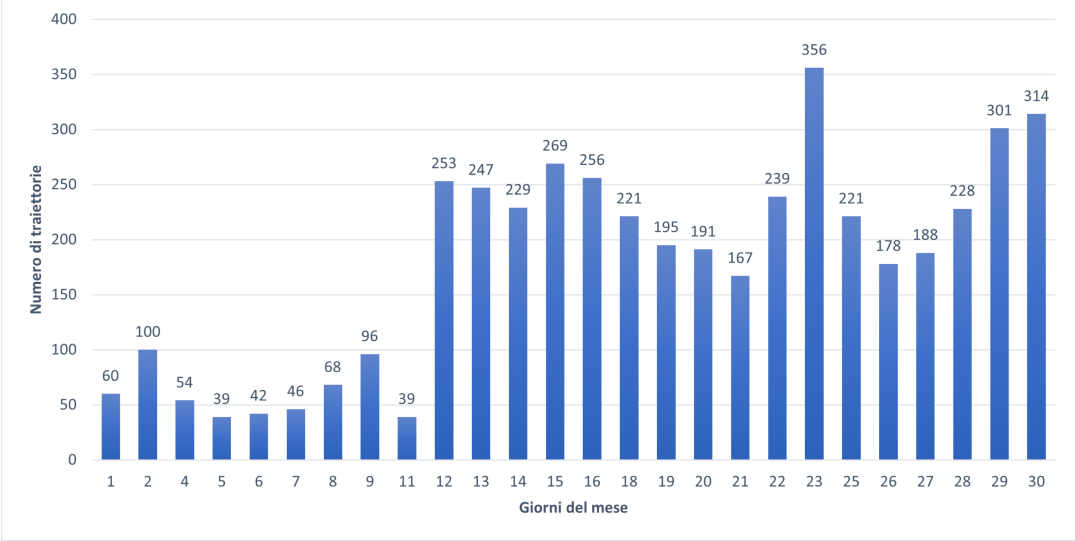


Figura 3.10: Numero di traiettorie estratte per i giorni del mese

3.4.3 Approccio ST-DBSCAN

I risultati provenienti dall'approccio precedente non sono ottimali e questo è dovuto all'utilizzo di metodi troppo semplici nella loro forma. La decisione di utilizzare metodi semplici e non computazionalmente onerosi è stata presa per evitare lunghi periodo di calcolo visto l'elevato numero di posizioni raccolte nel dataset sCREEN. Purtroppo visti i risultati non soddisfacenti si sono cercati approccio più complessi che potessero ottenere risultati migliori su un dataset così grande.

In particolare si sono cercati metodi che permettessero l'identificazione delle zone dense di punti all'interno delle traiettorie, estratte in precedenza, in modo da rimuoverle e rendere così i percorsi degli utenti più adatti per essere utilizzate da un modello.

L'analisi dei cluster è una fase molto importate in molti progetti e ricerche scientifiche e in diverse aree come data segmetation, discriminazione di attributi, noise filtering, rilevazione di valori anomali e image processing. Tipicamente, i metodi per la determinazione di cluster si basano su processi di apprendimento non supervisionato dato che non c'è nessuna conoscenza a priori sui dati ovvero non è a disposizione nessuna informazione su come raggrupparli. Molti metodi per individuare cluster funzionano solo per dati ordinari e ottengono risultati discutibili per dati con correlazione spaio-temporale come il dataset sCREEN, dove le posizioni raccolte oltre ad avere delle coordinate che le identificano nello spazio, hanno anche l'istante temporale di quando sono state misurate. La determinazione di cluster per questa tipologia di dati è più difficile perché gli algoritmi devono considerare, per la

creazione di cluster, sia l'aspetto spaziale che quello temporale dei dati.

Innanzitutto il problema di *Clustering* può essere definito come: dato un dataset D di n oggetti ovvero $D = \{a_1, a_2, a_3, ..., a_n\}$ allora il processo di partizione di D, basato su una certa misura di similarità, in $C = \{C_1, C_2, C_3, ..., C_k\}$ dove C_i viene chiamato cluster, ha la proprietà che $\bigcap_{i=1}^{k} C_i = \emptyset \, e \, \bigcup_{i=1}^{k} C_i = D \, con \, C_i \subseteq D \, dove \, i = 1, 2, ..., k$.

Esistono varie tipologie di algoritmi di clustering che si suddividono principalmente sulla metodologia di raggruppamento degli oggetti. Fondamentalmente gli algoritmi si dividono in quattro categorie:

- *Partitional algorithms*, i cluster vengono formati rispetto ad un valore di similarità tra gli oggetti e un valore medio considerato il centro di gravità del cluster, è il caso di K-Means [31], oppure in K-Medoid [25] dove il centroide del cluster viene rappresentato da un oggetto che è il più vicino al centro. Questi algoritmi hanno bisogno di un meta-parametro denominato K che rappresenta il numero di cluster desiderano con il quale si vuole dividere il dataset. Sfortunatamente questo parametro va conosciuto a priori e può cambiare radicalmente la partizione dei dati, quindi sono algoritmi che, per come sono definiti, non possono essere utilizzati in molti applicazioni.

- *Hierarchical algorithms*, producono un insieme di cluster nidificati organizzati come un albero gerarchico. Ogni nodo dell'albero rappresenta un cluster di un dataset. Un esempio di questa tipologia di algoritmi sono CURE [38] e BIRCH [51].

- *Grid-based algorithms*, si basano su una struttura a griglia a più livelli su cui vengono eseguite tutte le operazioni per identificare i cluster. In questa categoria si trovono algoritmi come STING [54] e WaveCluster [44].

- *Model-based algorithms*, ipotizzano che i dati siano generati da una combinazione di distribuzioni di probabilità quindi creano, per ogni cluster, un modello che cerca di rappresentarli. L'obbiettivo è di ottimizzare l'adattamento tra i dati presenti nel dataset e quelli generati dai modelli.

- *Density-based algorithms* è la classe di algoritmi più diffusa. L'idea di basa è che oggetti che formano una regione densa devono essere raggruppati in un unico cluster. Esempi di algoritmi famosi sono DBSCAN [34], OPTICS [33], DENCLUE [2] e CURD [45].

Per come è strutturato il dataset sCREEN e dato l'obbiettivo di individuare zone dense di punti all'interno delle traiettorie per eliminarle si è cercare un algoritmo di clustering basato su densità. Dopo alcuni test fatti con DBSCAN e OPTICS si è notato che il secondo algoritmo funzionava fin troppo bene per lo scopo, infatti va ad inserire punti all'interno di cluster anche se questi effettivamente non gli appartengono tagliando così parti di traiettoria. Per questo motivo e per un tempo di esecuzione minore si è optato per DBSCAN.

DBSCAN è progettato per identificare punti che appartengono a cluster di forma arbitraria da punti di rumore in qualsiasi dataset. Per creare i cluster l'algoritmo DBSCAN ha bisogno di un valore di raggio, basato su una misura di distanza definita dall'utente, per determinare la grandezza del cluster e un valore minimo di punti che ci devono appartenere.

L'algoritmo adotta un processo iterativo per determianare i cluster in particolare inizia con il primo punto del dataset e recupera tutti i suoi vicini all'interno del raggio definito. Se il numero totale dei punti vicini è maggiore del numero minimo di punti consentito affinché si possa definire un cluster, allora il punto preso è un *Core object* e viene creato il nuovo cluster. Un *Core object* è un punto tale che nell'intorno definito dal raggio scelto dall'utente contiene almeno il numero minimo di punti. Il processo viene ripetuto fino a quando tutti i punti sono stati elaborati.

Purtroppo, dopo una serie di analisi, neanche quest'algoritmo è risultato idoneo, infatti non tiene in considerazione, durante la formazione dei cluster, la temporalità dei punti raccolti. Quello che si è potuto notare è che l'algoritmo inseriva nella stesso cluster punti che, sebbene fossero vicini dal punto di vista spaziale, erano invece distanti per quanto riguarda il tempo. E' il caso di un consumatore che passa nella stessa zona per varie volte ma in istanti diversi.

La soluzione migliore proviene dall'utilizzo di una versione di DBSCAN sviluppata appositamente per dati con correlazione spazio-temporale ovvero *Spatial-Temporal DBSCAN* [13]. Quest'algoritmo risolve due grandi criticità di DBSCAN ovvero:

- Incapacità di ottenere risultati soddisfacenti per dati spazio-temporali;
- Assegnazione errata di punti di rumore a cluster con diversa densità.

Per poter supportare dati in due dimensioni l'algoritmo utilizza non più una sola misura di distanza ma due, ovvero rimane il raggio per definire la vicinanza di

due punti ma viene aggiunta un'altra soglia per stabilire la loro prossimità a livello temporale.

Gli algoritmi di clustering esistenti basati sulla densità producono risultati non soddisfacenti quando esistono cluster di densità diverse. Si prenda il caso nella figura 3.11 dove ci sono due diversi cluster C_1 e C_2 e due punti esterni o_1 *e* o_2. Come si può notare i due cluster hanno densità diversa ovvero i punti che appartengono a C_2 sono molto più vicini e densi al centro del cluster rispetto a quelli di C_1. L'algoritmo DBSCAN identifica solo un oggetto di rumore ovvero o_1. Questo è dovuto al valore di raggio scelto come iper-parametro dell'algoritmo, infatti al diminuire di questo alcuni punti che appartengono a C_1 potrebbero essere identificare come rumore mentre all'aumentare del valore di raggio, il punto o_1 non verrebbe identificato come rumore. Tipicamente un punto p si definisce *rumoroso* se data la suddivisione del dataset D in $C_1, C_2, C_3, ..., C_k$ allora p non appartiene a nessun cluster ovvero $p \in D\, e\, \forall i : p \notin C_i$.

Per risolvere questo problema ST-DBSCAN utilizza il concetto di *fattore di densità* ovvero assegna a ciascun cluster un fattore che esprime il grado di densità del cluster. Questo parametro viene definito come:

$$density\_factor(C) = \frac{1}{\left(\frac{\sum_{p \in C} density_distance(p)}{|C|}\right)} \tag{3.2}$$

dove *density_distance* è un valore determinato come:

$$density\_distance = \frac{density\_distance\_max(p)}{density\_distance\_min(p)} \tag{3.3}$$

con *density_distance_min(p)* la distanza minima tra un oggetto p e il suo vicino all'interno dell'area creata dal raggio scelto, mentre *density_distance_max(p)* la distanza massima.

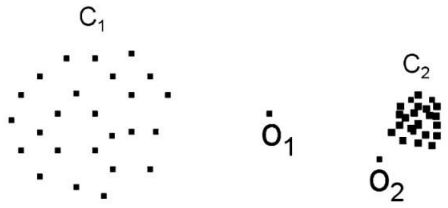


Figura 3.11: Esempio di dati dove ci sono due cluster con densità diversa

Il fattore di densità permette di capire il grado di densità del cluster. Se C è un cluster poco denso allora density_distance_min crescerà portando density_distance ad essere piuttosto piccola, costringendo così il fattore di densità ad essere prossimo

ad 1. Viceversa, se il cluster è denso allora density_distance_min diminuisce e di conseguenza density_distance aumenta portando il fattore di densità del cluster ad un valore quasi nullo.

Per l'estrazione delle traiettorie dei consumatori dal dataset sCREEN si è modificata la strategia per la loro identificazione. In primo luogo si vanno ad individuare delle traiettorie preliminari con l'approccio data missing. Successivamente si scartano le traiettorie con meno di 150 posizioni e con un deviazione standard inferiore o uguale a 1. Dopodiché, attraverso l'algoritmo ST-DBSCAN si vanno a cercare cluster di punti, all'interno dei percorsi, per individuarli ed eliminarli. Sono stati provati diversi valori per il parametro spaziale dell'algoritmo ma mantenendo sempre come funzione di distanza la distanza euclidea. Mentre per il parametro temporale si è voluto rimanere in linea con quanto scritto dagli autori del sistema sCREEN ovvero che una traiettoria di un cliente finisce quando il carrello o cestino della spesa rimane fermo per un minimo di 5 minuti o per lo meno, viste le considerazione precedentemente fatte sulla poca affidabilità dei tag installati sui carrelli, se rimane in una zona circoscritta in questo intervallo di tempo. Quindi ST-DBSCAN è stato impostato con soglia spaziale uguale ad 1 e parametro temporale uguale a 5 minuti. L'output dell'algoritmo fornisce una classe di appartenenza per ogni punto, in particolare se una posizione è classificata con un numero maggiore o uguale a zero allora il punto appartiene ad un specifico cluster mentre se la posizione viene classificata con -1 allora è un punto rumoroso che non appartiene a nessun cluster. Con questa classificazione è stato possibile eliminare le parti dense di punti dalle traiettorie, regioni che erano state create da errori di rilevazione dello spostamento dei tag. A questo punto le traiettorie risultavano frammentate ed è stato possibile suddividerle nuovamente con l'approccio data missing per poi rimuovere traiettorie troppo corte e poco dinamiche.

Dalle 119 traiettorie preliminari estratte in precedenza la suddivisione in sottotraiettorie a premesso di identificare 4283 percorsi.

3.4.3.1 Risultati ottenuti

Le traiettorie estratte con questo approccio, dopo un analisi qualitativa, risultano essere di maggior qualità rispetto alle precedente. Infatti come mostrato dalla figura 3.12, che rappresenta la traiettoria n° 2000, non sono più presenti zone dense di punti all'interno dei percorsi fatti dagli utenti.

Inoltre rispetto all'approccio precedente che aveva identificato quasi la metà delle posizioni come punti di rumore, il nuovo metodo va a rimuovere ancora più posizioni ritenute non utili, quindi le traiettorie identificate hanno in totale solo 1934381 e provengono da 105 tag distinti associati a diversi cestini o carrelli. Confrontando il numero delle nuove traiettorie estratte per i vari giorni del mese rispetto al vecchio metodo, come riportato dal grafico 3.10, si è potuto notare la stessa tendenza ovvero nei giorni con più posizioni raccolte si sono ottenute più traiettorie. La differenza con il metodo precedente è che per ogni giorno si ha un numero di percorsi estratti minore questo perchè il nuovo approccio riesce a riconoscere meglio cluster di punti che devono essere eliminati.

Figura 3.12: Esempio di traiettorie estratta

3.5 Interpolazione e preparazione dei dati

I modelli che verranno utilizzati per predire le traiettorie dei consumatori all'interno del supermercato hanno bisogno che i percorsi, estratti dall'approccio precedente, siano trascritte in una struttura specifica formata da frame. Ogni frame rappresenta cosa sta avvenendo nella scena in un determinato intervallo di tempo quindi in un unico frame possono esserci punti che provengono da diverse traiettorie ma che sono stati registrati nello stesso istante. Inoltre i frame hanno una grandezza specifica espressa in secondi che determina la durata della scena. Un frame può essere costi-

tuito quindi da più punti che devono essere scritti in forma adeguata, nello specifico ogni punto appartenente ad un frame ha le seguenti informazioni:

- Numero del frame;
- Numero della traiettoria di provenienza;
- Coordinate x e y.

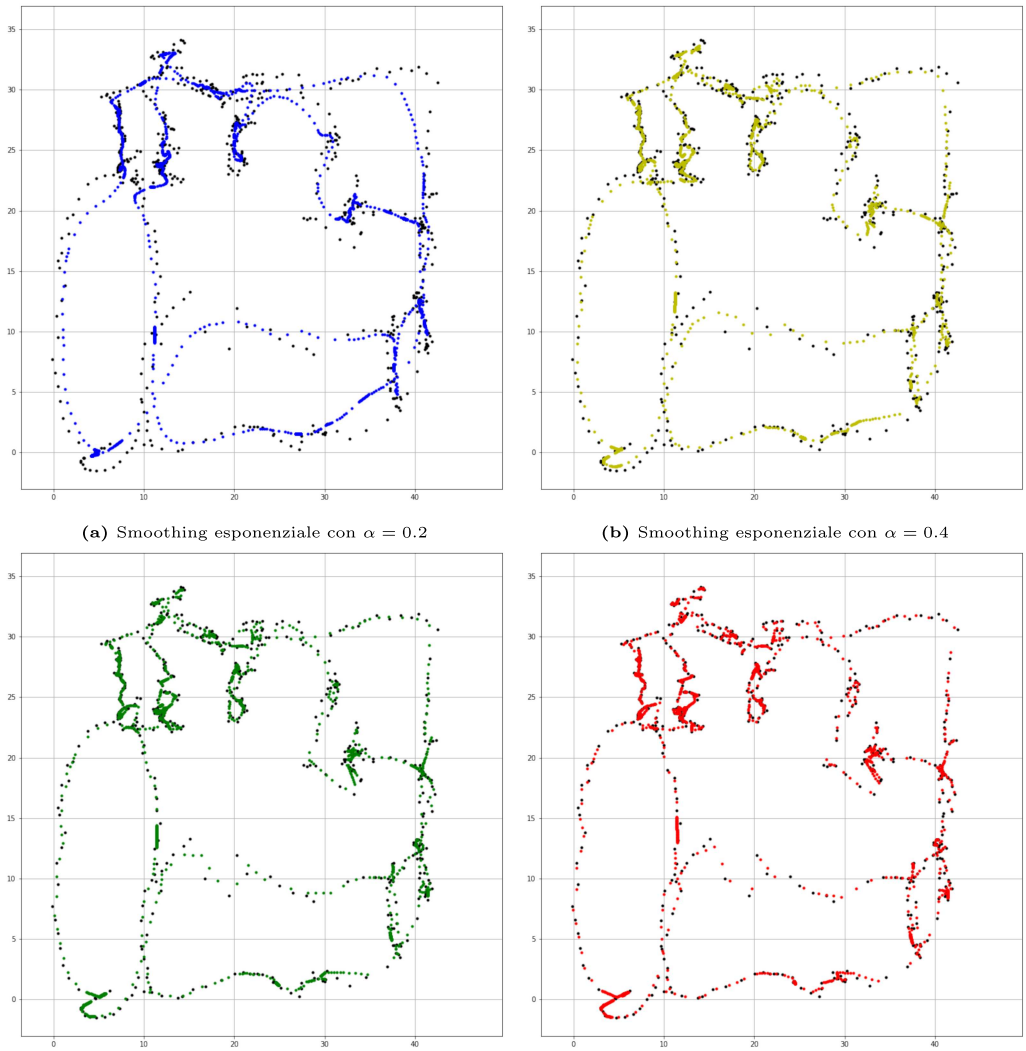
Come si può vedere nei frame non c'è nessun riferimento al tempo questo perché vengono creati in modo da rispettare una ordine temporale ovvero dati due frame consecutivi il primo conterrà solo punti che sono stati acquisiti prima rispetto alle posizioni del frame successivo. Quindi, ad esempio, se il primo punto in assoluto nel dataset sCREEN viene registrato in un determinato giorno e ora al secondo 0 e si sceglie una dimensione di frame uguale a 5 secondi allora questo frame conterrà tutti i punti registrati nello stesso giorno e ora con una variazione di 4 secondi. Mentre il frame successivo conterrà tutti i punti che saranno stati registrati dal quinto secondo in poi.

Purtroppo, come si è potuto analizzare, il dataset sCREEN non ha una registrazione costante nel tempo delle posizioni dei consumatori questo può portare al caso che per un determinato frame non ci siano punti e quindi la successione cronologiche verrebbe interrotta. Per ovviare a questo problema si è deciso di attuare un *resampling* sulle traiettorie estratte ovvero si è scelto un intervallo di tempo sul quale ci dovà essere per forza un punto. Nel caso in cui la traiettoria non preveda punti per quel momento allora verrà creato una nuova posizioni le cui coordinate verranno calcolate mediante interpolazione.

Sono stati testati diversi metodi di interpolazione, per capire quale funzionasse meglio su i dati, a diversi sampling rate. I metodi di interpolazioni esaminati sono:

- Lineare;
- Gaussiana;
- Smoothing esponenziale con successiva interpolazione lineare;
- Interpolazione lineare con consecutivo smoothing esponenziale.

Lo *Smoothing esponenziale* [29] è una tecnica empirica per livellare i dati delle serie temporali utilizzando una funzione a finestra esponenziale. Mentre nella media mobile semplice le osservazioni passate sono ponderate allo stesso modo, i risultati,



(a) Smoothing esponenziale con $\alpha = 0.2$

(b) Smoothing esponenziale con $\alpha = 0.4$

(c) Smoothing esponenziale con $\alpha = 0.6$

(d) Smoothing esponenziale con $\alpha = 0.8$

Figura 3.13: Traiettoria n° 4233 dove vengono testati diversi valori di α per lo smoothing esponenziale

ottenuti con funzioni esponenziali, sono medie ponderate delle osservazioni passate, con i pesi che decadono esponenzialmente man mano che le osservazioni invecchiano, ovvero più l'osservazione è recente, maggiore è il peso associato. Le coordinate dei punti vengono calcolate nel seguente modo:

$$s_o = x_o \; con \; t = 0$$

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1} \; con \; t > 0$$

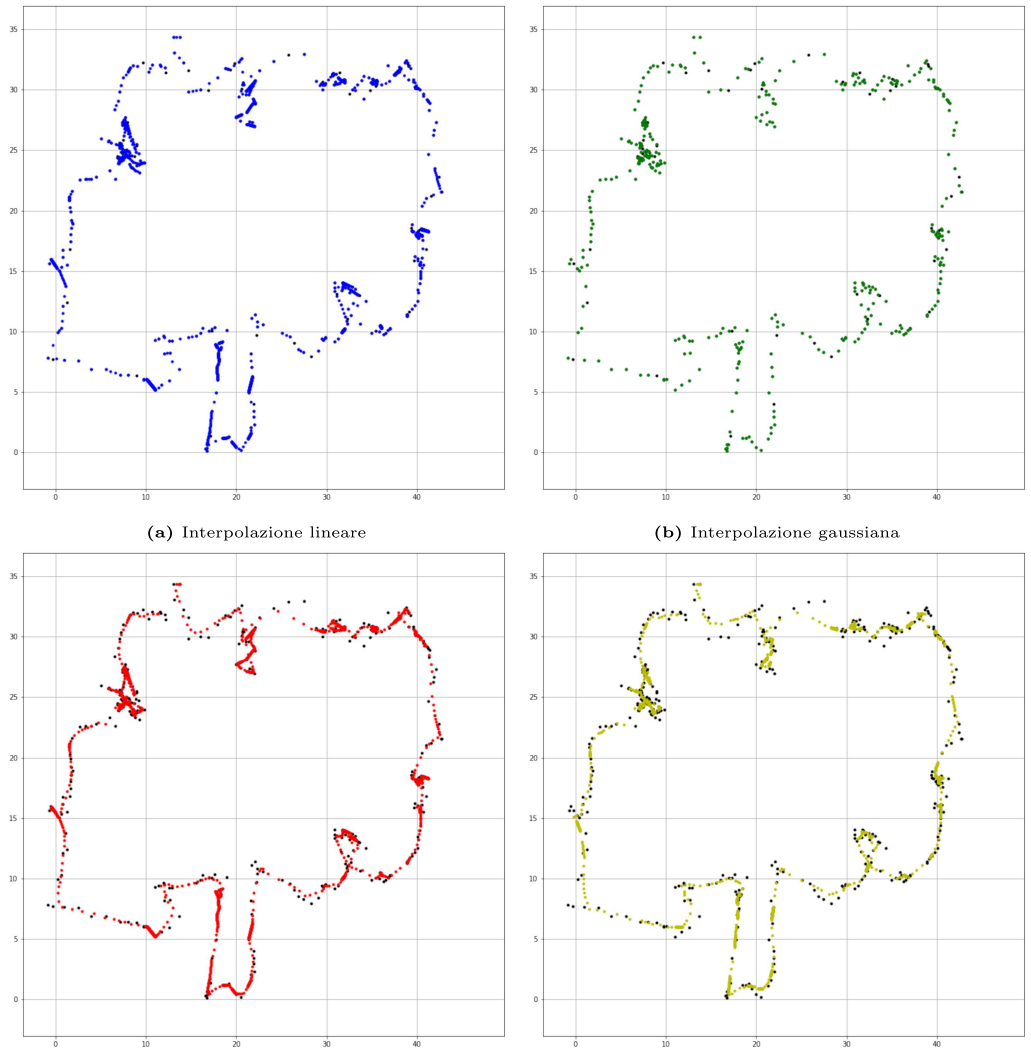
Sono stati testati diversi valori di α, come mostrato in figura 3.13, tuttavia dopo una valutazione qualitativa rispetto alla traiettoria di partenza si è deciso di usare un valore di $\alpha = 0.4$.

Per quanto riguardo il sampling rate sono stati provati diversi valori. Tuttavia si è potuto notare come a ssampling rate alti, ovvero compresi tra i 3 e 5 secondi, l'errore di interpolazione delle coordinate dei nuovi punti sia troppo elevato portando modifiche sostanziali alla traiettoria. Quello che si è notato è che fissato un intervallo di tempo dove, per ogni sua ripetizione, ci deve essere un punto si vanno ad eliminare anche posizioni reali che sono state raccolte da sistema sCREEN. Questo può portare che tra due punti reali ci sia un intervallo maggiore di tempo e quindi dovranno essere creati più posizioni artificiali per riempirlo. Di conseguenza le coordinate di questi punti dovranno essere approssimate rispetto alle prime posizioni reali e maggiore è la distanza temporale tra i punti reali maggiore sarà l'errore di approssimazione. Affinché si posso avere un buona interpolazione si è deciso di utilizzare un sampling rate uguale a 1 secondo.

Tra i diversi metodi di interpolazione valutati si è potuto vedere, come riportato dalla figura 3.14, che il metodo con smoothing esponenziale e successiva interpolazione lineare ottiene risultati migliori. Infatti la sola interpolazione lineare crea una serie di punti artificiali, tra due punti reali, che formano una retta e se l'intervallo temporale da riempire è troppo grande la retta formata va a rappresentare un comportamento di un utente troppo irrealistico. Mentre l'interpolazione gaussiana, dato che utilizza una distruzione gaussiana per correggere le coordinate dei punti reali e artificiali crea traiettorie con una concentrazione di punti alta nel centro perdendo posizioni nei lati. Invece con interpolazione lineare e successivo smoothing esponenziale si ottengono traiettorie con gli stessi problemi dall'interpolazione lineare, visti in precedenza, anche se vengono mitigati dallo smoothing.

Nonostante la strategia ottimale di estrazione e pulizia delle traiettorie dal dataset sCREEN abbia permesso di togliere molte posizioni rumorose, passando da 4622438 punti a 1934802 di posizioni, il numero di percorsi trovati è comunque elevato e impostando un sampling rate di 1 secondo il numero totale di posizioni sale raggiungendo quota 6219927.

Purtroppo un numero così alto di punti aumenta considerevolmente il tempo di allenamento dei modelli così date le traiettorie con resampling a 1 secondo si è provato ad aumentare il sampling rate nel tentativo di diminuire la mole di dati. Tuttavia come verrà discusso nei capitoli successivi aumentare il sampling rate e quindi anche la dimensione dei frame non porta a buoni risultati. Per questo al fine di ottenere risultati ammissibili i vari modelli sono stati allenati con dataset con sampling rate uguale a 1 secondo e per seguire il protocollo standard, utilizzato da



(a) Interpolazione lineare

(b) Interpolazione gaussiana

(c) Interpolazione lineare con successivo smoothing esponenziale con $\alpha = 0.4$

(d) Smoothing esponenziale con $\alpha = 0.4$ e successiva interpolazione lineare

Figura 3.14: Traiettoria n° 3878 sulla quale vengono applicati diversi metodi di interpolazione

diversi studi nel campo delle predizioni di traiettorie di pedoni, il sampling rate è stato ulteriormente abbassato a 0.4 secondi.

Dato che le posizioni raccolte dal sistema sCREEN hanno precisione fino al secondo è stato necessario introdurre nei dati originali, per avere un sampling rate così basso, anche i decimi di secondo. Per non rimuovere, durante il ressampling, un elevato numero di punti reali, i decimi di secondo sono stati impostati a multipli di 4. In pratica presi due punti di una traiettoria casuale, che distano l'uno dall'altro un secondo, il primo punto avrà decimi di secondo uguale a 0 mentre il secondo gli avrà uguali a 2, questo perché tra le due posizioni si andranno ad inserire altri due punti che avranno lo stesso tempo del primo punto ma decimi di secondo pari a 4 e 8 rispettivamente.

3.6 Correzione dei punti

Dopo l'estrazione delle traiettorie e aver fatto il resampling dei punti per ottenere un intervallo temporale regolare con relativa interpolazione per correggere le coordinate delle posizioni si è notato che queste non rispettavano i confini degli scaffali mostrati dagli autori del sistema sCREEN nella figura 3.1. Ovvero si è osservato che sebbene le traiettorie ricordino vagamente percorsi che possono essere associati alla mappa del supermercato preso in esame, i loro punti risultano essere acquisiti in zone dove un consumatore non potrebbe andare, ad esempio all'interno di un scaffale o addirittura fuori dal supermercato.

Nel tentativo di ottenere predizioni di traiettorie più realistiche si provveduto a fare delle correzioni alle coordinate dei punti in modo che, qualora un posizione si trovasse in una zona non permessa, come ad esempio all'interno di uno scaffale, allora questa viene modificata e spostata nella prima zona valida.

Dato che nel paper di presentazione del sistema sCREEN non c'è riferimento alle coordinate degli scaffali si è dovuto ricavarle empiricamente. Come primo passaggio si è trovato la migliore sovrapposizione tra la mappa, fornita dagli autori del paper, e le traiettorie estratte non andando a modificarne le proporzioni. Successivamente sono state misurate le coordinate degli angoli di ognuno dei 44 scaffali. Grazie alle coordinate ottenute è stato possibile definire delle regioni non attraversabili dalle traiettorie in modo da identificare tutti i punti che si trovassero all'interno di queste regioni per poterli spostare. Nella pratica, dopo aver trovato una serie di punti all'interno di un'area non consentita, si determina la deviazione standard delle coordinate per capire in quale direzione la traiettorie si sta muovendo in quel intervallo di tempo. Ad esempio se la deviazione standard della coordinata x è maggiore di quella della coordinata y, significa che la sezione di punti, che si trova dentro quel specifico scaffale, si muove principalmente di lato e quindi bisogna modificare le coordinate y dei punti in esame. Dopo aver determinato l'asse principale di movimento dei punti della sezione è stato possibile individuare in quale lato dello scaffale riposizionare le posizioni calcolando la media della coordinata presa in considerazione. Ovvero se quest'ultima risulta essere più vicina ad uno specifico bordo allora tutti i punti vengono spostati al di fuori di quest'ultimo. Quindi se per ipotesi si determina che l'asse principale di movimento della sezione di punti, all'interno dello scaffale, è l'asse x allora questi devono essere spostati rispetto ad uno dei due bordi orizzontali dello scaffale. Il bordo scelto sarà quello che avrà coordinate x più simili alla media delle coordinate x dei punti della sezione.

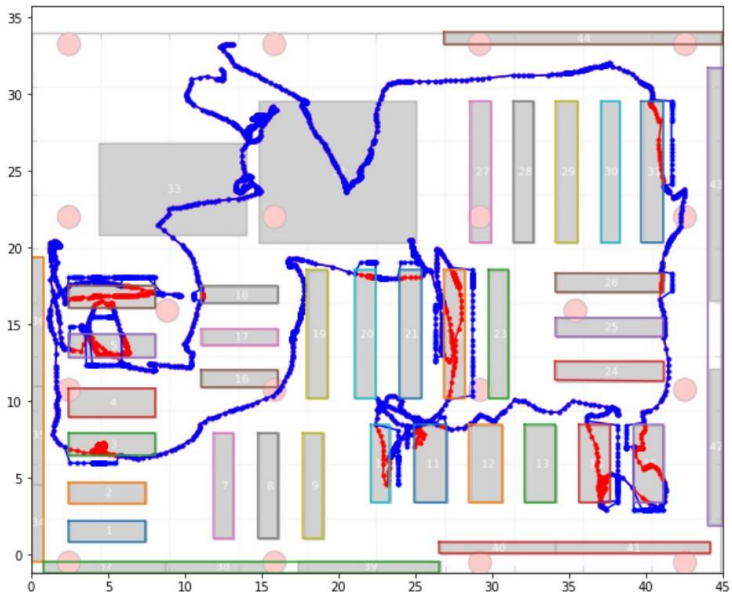


Figura 3.15: Esempio di riposizionamento dei punti all'interno degli scaffali della traiettoria n°5. In rosso la traiettoria originale mentre in blu quella modificata. I rettangoli colorati rappresentano le regione dove i clienti non possono attraversare

La figura 3.15 mostra un esempio di traiettoria prima e dopo la correzione dei punti. Come si può notare, sempre da questa immagine, gli scaffali 32 e 33 sono stati identificati come regioni dove ai consumatori è concesso il transito, questo perchè si è osservato che in questi due scaffali c'è un elevato numero di punti posizionati proprio al loro interno e visto che sono i due più grandi si è ipotizzato che fossero delle isole per l'esposizione di prodotti dove ai clienti è permesso camminare all'interno. Data questa ipotesi si è preferito evitare di definirli come regioni non pedonabili lasciando inalterati i punti al loro interno.

Capitolo 4

Modelli

In questo capitolo verranno descritti i modelli che sono stati utilizzati nel progetto di tesi per predire le traiettorie future dei pedoni a partire da un periodo di osservazione.

4.1 Modelli deterministici

Uno dei principali vantaggi del Deep Learning è la capacità di risolvere problemi complessi che richiedono la scoperta di schemi nascosti nei dati o di una profonda comprensione delle relazioni tra un gran numero di variabili interdipendenti. Gli algoritmi di Deep Learning sono in grado di apprendere questi pattern nascosti direttamente dai dati, combinarli insieme e creare regole decisionali molto più efficienti. Tuttavia, questi algoritmi al fine di ottenere delle buone performance, devono essere applicati solo in determinate circostanze, in particolare se è presente una grande quantità di dati per l'allenamento e se il problema è molto complesso come la classificazione delle immagini, l'elaborazione del linguaggio naturale, il riconoscimento vocale e la predizione.

Dato che il dataset sCREEN non è mai stato utilizzato per task di predizione di traiettorie di pedoni, al fine di determinare se l'utilizzo di modelli di Deep Learning possono ottenere risultati migliori, sono stati testati due diversi modelli deterministici per predire le traiettorie future in modo da confrontare i risultati ottenuti e individuare qual'è la soluzione ottimale tra i diversi modelli.

4.1.1 Costant velocity model

Negli ultimi anni sono state proposte diverse soluzioni per predizioni di traiettorie e molte delle quali si basano su reti neurali. Questi modelli vengono ampliamenti utilizzati perché le reti neurali sono potenti approssimatori di funzioni e si ritiene che siano in grado di tenere conto dei movimenti passati dei pedoni e di imparare come interagiscono tra di loro.

Tuttavia non è certo che reti complesse possono ottenere prestazioni migliori rispetto a modelli più semplici. Infatti come dimostrato nello studio [42] anche un semplice modello deterministico può ottenere risultati comparabili o addirittura migliori rispetto a modelli che utilizzano reti neurali.

Viene dimostrato come assunzioni preliminari sul movimento dei pedoni e problemi nell'allenamento delle reti neurali possono portare a prestazioni peggiori. In particolare viene dimostrato come l'aggiunta di vincoli fisici e semantica ambientale distorcono il movimento del pedone verso determinati schemi. L'aggiunta di informazione sull'ambite oltre a portare a risultati peggiori è anche un dato ridondante in quanto la rete neurale riesce ad apprendere una conoscenza approssimativa sull'ambiente. Inoltre fornire alla rete neurale un grande storico dei vari movimenti dei pedoni al fine di avere prestazioni migliori non permette di ottenere previsioni più accurate anche nel caso di traiettorie complesse.

Il *Constant Velocity Model* presentato in [42] è un modello che non richiede alcuna informazione oltre l'ultimo movimento relativo del pedone, infatti viene considerato quest'ultimo come il più significativo al fine della previsione della sua traiettoria futura. In particolare il modello presuppone che il pedone continuerà a camminare con la stessa velocità e direzione osservata negli ultimi due passaggi temporali.

4.1.2 Modello basato su Brownian Motion

Con il termine *Brownian motion o moto browniano* si fa riferimento al moto disordinato di particelle sufficientemente piccole da essere sottoposte a una forza di gravità trascurabile, che sono presenti all'interno di un fluido e sono osservabile al microscopio. Il fenomeno fu scoperto agli inizi dell'Ottocento dal botanico scozzese Robert Brown e poi fu modellizzato nel 1905 dal fisico teorico tedesco Albert Einstein. Con questo termine si indica sia il fenomeno naturale sia la sua rappresentazione matematica, la quale può descrivere l'andamento temporale di una classe molto ampia

di fenomeni casuali.

In particolare il moto browniano viene definito come un processo casuale $X = X_t : t \in [0, \infty)$ a valori reali che soddisfa le seguenti proprietà:

- $X_o = 0$.
- è un processo stazionario ovvero la sua distribuzione di probabilità congiunta non cambia nel tempo, di conseguenza, parametri quali come media e varianza rimangono inalterati. sia $s, t \in [0, \infty)$ con $s < t$ allora la distribuzione di $X_t - X_s$ è la stessa di X_{t-s}.
- X è un processo a incrementi indipendenti ovvero dato $t_1, t_2, t_3, ..., t_n \in [0, \infty)$ con $t_1 < t_2 < t_3 < ... < t_n$, allora le variabili aleatorie $X_{t_1}, X_{t_2}, X_{t_3}, ..., X_{t_n}$ sono indipendenti.
- X_t è normalmente distribuito con media 0 e varianza t per ogni $t \in [0, \infty)$.

Quindi il modello utilizzato prende in considerazione l'ultimo movimento fatto dal pedone come punto di partenza per la traiettorie predetta e i successivi punti vengono calcolati aggiungendo un valore casuale, estratta da una certa distribuzione di probabilità, all'ultimo punto predetto.

4.2 Long-Short-Term Memory

Le Reti Neurali Ricorrenti sono modelli matematici che negli anni hanno più volte dimostrato la loro superiorità in compiti difficili o impraticabili per altri tipi di reti neurali. Tuttavia il loro vero potenziale è limitato dall'efficacia della procedura di allenamento applicata, infatti algoritmi di discesa di gradiente come *Back-Propagation Through Time* o *Real-Time Recurrent Learning* [55] che propagano l'errore commesso dalla rete all'indietro al fine di modificare i pesi attraverso il gradiente, soffrono della criticità del *Vanishing/Exploding gradient problem* discusso nel capitolo 2.3. Questo problema non permette alle reti ricorrenti di apprendere dipendenze tra dati lontani in una sequenza mettendo in dubbio se le RNN standard possano mostrare vantaggi pratici significativi rispetto a reti feedforward basate su finestre temporali. Una soluzione al Vanishing/Exploding gradient problem che permette comunque alle rete neurale di apprendere dipendenze lontane nei dati è stata data nel 1997 dai ricercatori Hochreiter e Schmidhuber con il modello *Long Short-Term Memory*.

4.2.1 Vanishing/Exploding gradient problem

Per comprendere meglio questo problema di apprendimento si prende in considerazione l'esempio della semplice rete neurale ricorrente in figura 4.1, questa avrà in input un vettore x_t di dati per ogni istante temporale t e il vettore di contesto c_{t-1} che esprime le informazioni passate e la conoscenza acquisita, fino al momento precedente. Di conseguenza la rete restituirà, ad ogni istante t, il vettore output h_t che viene calcolato come:

$$h_t = \sigma(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t)$$

dove W_{rec} è la matrice di pesi della connessione ricorrente, W_{in} la matrice di pesi per i dati di input e σ è una funzione di attivazione non lineare come sigmoide o tangente iperbolica. In questo caso di rete neurale ricorrente il vettore di contesto c_t è uguale al vettore di output.

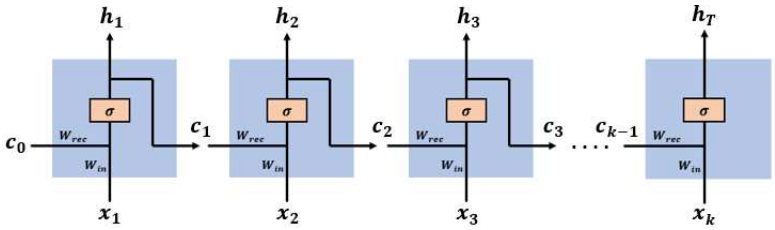


Figura 4.1: Esempio di un rete neurale ricorrente semplice che viene "srotolata" nel tempo creando una copia del modello per ogni fase temporale.

Al fine di migliorare le performance della rete e quindi minimizzare la funzione di errore E è necessario modificare i parametri del modello, ovvero i pesi, attraverso discesa di gradiente.

$$\Delta\theta = -\eta\frac{\partial E}{\partial \theta}$$

dove η è il learning rate e θ raccoglie le matrici dei pesi W_{rec}, W_{in} per un caso generale. Per calcolare i necessari gradienti vine utilizzato l'algoritmo Backpropagation Through Time che è molto simile al normale algoritmo di Backpropagation dato che si basa sul fatto, che per un certo intervallo di tempo, la rete neurale ricorrente può essere "srotolata" e paragonata ad una semplice rete completamente connessa. Infatti il calcolo del gradiente è quasi simile:

$$\frac{\partial E}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial \theta}$$

Per quanto riguarda il termine $\frac{\partial E_t}{\partial \theta}$, che esprime il gradiente da calcolare per l'istante t, si calcola come:

$$\begin{aligned} \frac{\partial E_t}{\partial \theta} &= \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}}\cdots\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial \theta} \\ &= \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\left(\prod_{k=2}^{t}\frac{\partial c_k}{\partial c_{k-1}}\right)\frac{\partial c_1}{\partial \theta} \end{aligned} \tag{4.1}$$

dove la derivata di c_t è:

$$\begin{aligned} \frac{\partial c_t}{\partial c_{t-1}} &= \sigma^{'}(W_{rec}\cdot c_{t-1} + W_{in}\cdot x_t)\cdot\frac{\partial}{\partial c_{t-1}}\cdot[(W_{rec}\cdot c_{t-1} + W_{in}\cdot x_t] \\ &= \sigma^{'}(W_{rec}\cdot c_{t-1} + W_{in}\cdot x_t)\cdot W_{rec} \end{aligned} \tag{4.2}$$

Sostituendo poi nella equazione 4.1 la derivata di c_t espressa nella precedente equazione si ottiene:

$$\frac{\partial E_t}{\partial \theta} = \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\left(\prod_{k=2}^{t}\sigma^{'}(W_{rec}\cdot c_{k-1} + W_{in}\cdot x_k)\cdot W_{rec}\right)\frac{\partial c_1}{\partial \theta} \tag{4.3}$$

Nell'equazione 4.3 si può notare dove si manifesta il vanishing/exploding gradient problem ovvero nella produttoria, nello specifico se:

$$\mid \sigma^{'}(W_{rec}\cdot c_{k-1} + W_{in}\cdot x_k)\cdot W_{rec} \mid > 1$$

per ogni k, allora il prodotto crescerà esponenzialmente, facendo esplodere l'errore portando a pesi delle connessioni ad oscillare di molto e di conseguenza ad avere un apprendimento instabile. Viceversa se:

$$\mid \sigma^{'}(W_{rec}\cdot c_{k-1} + W_{in}\cdot x_k)\cdot W_{rec} \mid < 1$$

per ogni k, allora il prodotto diminuisce esponenzialmente, facendo svanire l'errore, impedendo alla rete di apprendere entro un periodo di tempo accettabile.
Quindi come dimostrato anche nello studio [22] una normale rete neurale ricorrente non potrà apprendere dipendenze lontane nel tempo che sono distanti più di 5-10 istanti temporali.

4.2.2 Equazioni

La cella LSTM è esplicitamente progettata per evitare il *long-term dependency problem* infatti può imparare dipendenze tra dati che sono distanti più di 1000 istanti temporali. Nella versione originale presentata in [27], al fine di costruire una architettura che permettesse di propagare all'indietro l'errore commesso senza che questo svanisse o esplodesse utilizzando comunque una unità auto-connessa, fu introdotto il *Constant Error Carousel (CEC)* che permette di mantenere inalterato lo stato della cella, ovvero la sua memoria, in assenza di nuovo input o di segnale d'errore. Quindi lo stato della cella scorre direttamente lungo l'intera catena temporale con solo alcune interazioni lineari controllate.
Le modifiche alla memoria della cella avvengono grazie un *input gate* che permette di selezionare una parte dei dati di input, o escluderli del tutto, che possono essere aggiunti alla memoria. Allo stesso modo per difendere le altre celle da perturbazioni di contenuti irrilevanti, salvati nella memoria della cella stessa, vine inserito un *output gate* per selezionare i dati in uscita. L'unità base di questo modello è il *memory block* che può contenere una o più *memory cell* e una coppia di gate che fanno entrare l'input e uscire l'output da tutte le celle del blocco.

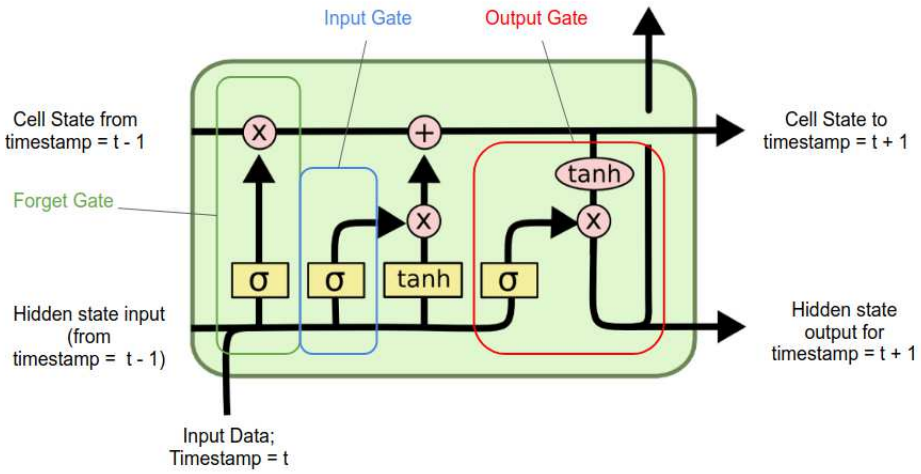


Figura 4.2: Cella LSTM

In particolare l'input gate, riferito all'istante temporale t, si definisce come:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{4.4}$$

dove σ è la funzione non lineare sigmoide, W_i la matrice dei pesi del gate al fine di apprendere quali dati possono entrare nella cella, h_{t-1} l'output della cella nell'istante precedente e x_t i dati di input. Similmente viene definito anche l'output gate come:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \tag{4.5}$$

mentre ad ogni istante temporale, la memoria della cella viene aggiornata nel seguente modo:

$$C_t = C_{t-1} + i_t \cdot \tanh(W_c \cdot [H_{t-1}, x_t] + b_c) \tag{4.6}$$

Infine l'output della cella sarà una versione filtrata della memoria della cella stessa:

$$h_t = o_t \cdot \tanh(C_t) \tag{4.7}$$

viene usata la funzione tanh per scalare lo stato della memoria in un range compreso tra -1 e 1.

Come detto in precedenza LSTM consente di archiviare le informazioni di intervalli di tempo arbitrari e di trasportare i segnali di errore molto indietro nel tempo. Purtroppo come dimostrato in [22] questa sua grande capacità è anche il suo punto debole dato che lo stato della cella tende a crescere linearmente durate l'allenamento del modello. Infatti se si presenta un flusso di input continuo, gli stati delle celle possono crescere in modo illimitato, causando la saturazione della funzione nell'output del modello. Questo accade anche se la natura del problema permette agli stati delle celle di essere ripristinati occasionalmente come ad esempio all'inizio di una nuova sequenza di input. Il principale effetto negativo è che l'intera cella di memoria perde la sua capacità di memorizzazione e inizia a funzionare come un normale neurone di rete RNN.

Per risolvere il problema in [22] viene introdotto un *forget gate* che impara a ripristinare lo stato interno della cella di memoria quando le informazioni memorizzate non sono più necessarie. Quindi adesso la memoria della cella viene aggiornato come:

$$\begin{aligned} C_t &= f_t \cdot C_{t-1} + i_t \cdot \tanh(W_c \cdot [H_{t-1}, x_t] + b_c) \\ f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \end{aligned} \tag{4.8}$$

Nel progetto di tesi viene usata la versione di LSTM con forget gate con l'aggiunta di uno strato di output per trasformare l'input, ad alta dimensionalità dell'ultima cella in due coordinate cartesiane.

4.3 Trasformers

Come visto nel paragrafo 2.3 le reti neurali ricorrenti sono un particolare tipo di rete neurale che permette di usare dati sequenziali ovvero dove un particolare dato non è slegato dal contesto ma è in funzione del precedente. Quindi sono reti in grado di connettere informazioni precedenti all'attività presente, ad esempio utilizzando informazioni da fotogrammi video passati per interpretare il fotogramma corrente.

Al fine di catturare la dipendenze nei dati è stato necessario modificate le normali rete neurali feed-forward aggiungendo una connessione retrograda, nello strato nascosto, che permette alla rete di "ricordare" cosa è accaduto negli istanti temporali precedenti e quindi consentire alle informazioni di persistere. Tuttavia se un dato dipende da un altro molto distante, all'interno della sequenza, sarà molto difficile per la rete imparare questa caratteristica per il problema del vanishing/exploding gradient che impedisce una corretta modifica ai pesi della rete. L'incapacità di una rete neurale ricorrente di non apprendere dipendenze molto lontane nel tempo viene comunemente chiamato *long-term dependencies problem*. Dato che le reti neurali ricorrenti sono afflitte da questo problema di apprendimento, negli anni sono state proposte diverse soluzioni per permette a queste reti di apprendere dipendenze tra dati distanti. Soluzioni proposti in letteratura sono ad esempio *Long-Short Term Memory* [27], *Gated Recurrent Units* [14] oppure *Bidirectional recurrent neural networks* [43].

Una delle soluzioni più usate è il modello LSTM che mantiene una memoria compatta, sotto forma di un vettore di numeri, a cui il modello può accedere per modificarla aggiungendo nuove informazioni o dimenticando quello ritenute meno importanti. Nonostante il modello riesca a risolvere il problema del vanishing/exploding gradient presenta un ulteriore criticità ovvero la capacità. LSTM è un modello pensato a celle dove ogni unità di memoria può influenzare la cella successiva con un parametro apprendibile. Questo comporta ad una serie di parametri apprendibili nel modello che crescono quadraticamente con la dimensione della memoria.

Inoltre non viene risolto un altro problema tipico delle reti neurali ricorrenti ovvero la parallelizzazione dei dati, infatti l'output al tempo t dipende da quello al tempo $t-1$ rendendo i tempi di training molto lunghi.

Al fine di ottenere una rete neurale in grado di catturare dipendenze in dati molto distanti all'interno di una sequenza con tempi di training veloci, nel 2017 è stato presentato una nuova tipologia di modello chiamata *Trasformer* [52]. La novità di questo modello è che non si basa su connessioni ricorrenti ma utilizza un particolare meccanismo di attenzione, chiamato *Self-attention*, per catturare le dipendenze tra

i dati.

I Trasformer sono stati introdotti per migliorare le presentazioni nel campo della traduzione automatica. Come per altri modelli, presenti in letteratura, i Trasformer fanno uso di una architettura encoder-decoder con l'aggiunta di elementi innovativi. Intuitivamente l'encoder mappa una sequenza di ingresso, formata da diversi simboli, $(x_1, ..., x_n)$ in una differente sequenza $z = (z_1, ..., z_n)$ che verrà utilizzata successivamente dal decoder per generare una sequenza di simboli in output $(y_1, ..., y_m)$.

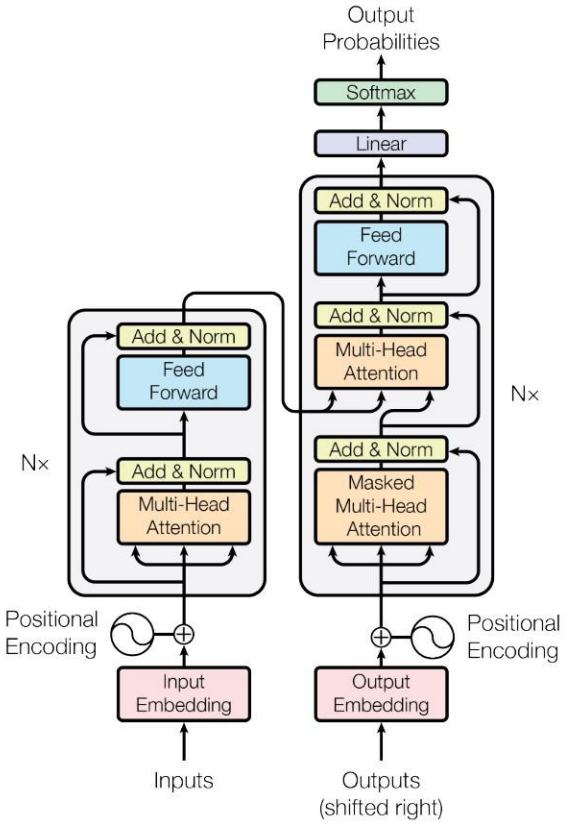


Figura 4.3: Trasformer. A sinistra l'encoder e a destra il deconder

Il modello è quindi formato da una serie concatenata di celle encoder seguite da altrettante celle decoder, la figura 4.3 mostra l'architettura base del modello.

L'encoder è formato da N strati uguali e ogni strato si suddivide in due sezioni. La prima sezione viene chiamata *Multi-head self-attention* mentre la seconda è una semplice rete feed-forward completamente connessa. Inoltre tra una sezione e la successiva è stata creata una connessione residua che porta l'input della sezione precedente, non alternato, alla sezione seguente.

Anche il deconder è formato da N strati uguali dove ogni strato si suddivide in tre sezioni, come l'encoder è presente la sezione Multi-head self-attention,

la rete feed-forward completamente connessa e le connessioni residue tra strati ma differentemente dagli strati precedenti viene aggiunta la sezione *Masked Multi-head self-attention*. Questa nuova sezione permette di oscurare parte dei dati di input per migliorare le predizioni.

Come si può vedere in figura 4.3 il modello non utilizza direttamente i dati grezzi su i quali apprendere ma una loro rappresentazione distribuita. Nello specifico si mappa una distribuzione di simboli come lettere, parole o numeri in uno spazio vettoriale a valori continui dove i vettori, associati alle parole, sono più vicini tra di loro, nello spazio vettoriale, se le parole occorrono negli stessi contesti linguistici, cioè se sono riconosciute come semanticamente più simili. Tuttavia durante la conversione degli elementi di input in vettori a valori continui viene persa la loro posizione all'interno della sequenza. Inoltre dato che il modello non fa uso di convoluzione e di connessioni ricorrenti, affinché riesca ad utilizzare l'ordine dei vari dati nella sequenza, viene aggiunta la posizione codificata nella rappresentazione vettoriale dell'input nella prima cella dell'encoder e del decoder. Quindi, intuitivamente, se lo stesso input appare in una posizione diversa, la sua rappresentazione effettiva sarà leggermente differente a seconda di dove appare.

Come accennato in precedenza i Trasformer introduco il concetto il *Self-Attention* che è leggermente diverso rispetto al normale meccanismo di attenzione trattato nei paragrafi precedenti. Ad esempio se si considera un task di traduzione per determinare il contesto o l'influenza di una parola rispetto alle altre, all'interno di una sequenza, il normale meccanismo di attenzione prevede di determinare un punteggio date le rappresentazioni vettoriali dei dati in input. Il punteggio di un vettore viene calcolato attraverso il prodotto scalare tra il vettore stesso e tutti gli altri, in questo modo si ottengono tanti valori quante le parole della sequenza. Successivamente i valori ottenuti vengono normalizzati utilizzando un funzione Softmax in modo che la loro somma sia uguale a 1. Infine vengono moltiplicati per le rappresentazioni vettoriali delle parole e ogni punteggio, così ottenuto, esprimerà quanto il vettore scelto, e quindi la parola rappresentata, è correlato rispetto a tutti gli altri. Il problema di questo approccio è che non ci sono parametri esterni che possono essere appresi.

Il meccanismo di Self-Attention non è altro che il normale meccanismo di attenzione ma con l'introduzione di parametri che possono essere appresi da un modello. Nello specifico vengono introdotte tre matrice di parametri che vanno a moltiplicare la rappresentazione vettoriale dei dati di input per ottenere tre matrici comunemente

chiamate Query, Key e Value da essere utilizzate nel meccanismo di attenzione.

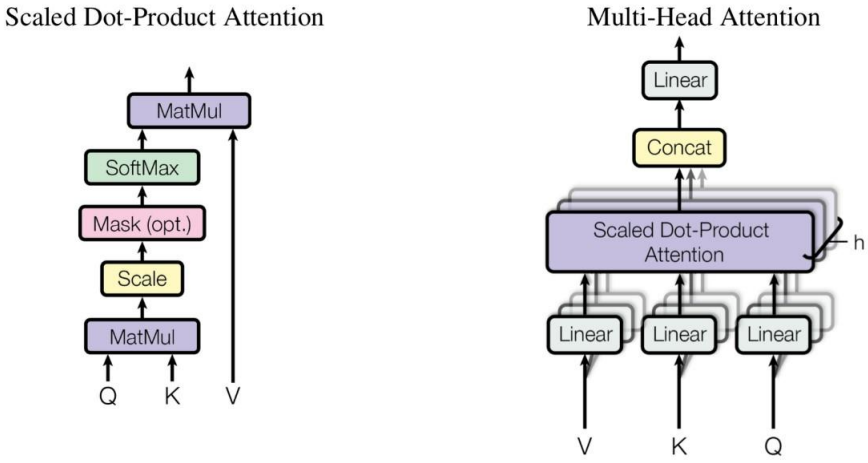


Figura 4.4: A sinistra il Scaled Dot-Product Attention, mentre a destra il Multi-Head Attention che consiste in una serie di strati di attenzione che calcolano in parallelo.

Nei Trasformer è il meccanismo di self-attention viene rappresentato dal modulo Multi-head self-attention. Come mostrato in figura 4.4 questo modulo si suddivide in due sotto-moduli denominati *Scaled Dot-Product Attention* e *Multi-Head Attention*. Il primo sotto-modulo si occupa di calcolare i coefficienti di attenzioni date le matrici Query, Key e Value come descritto in precedenza. Formalmente i coefficienti di attenzione vengono determinati mediante la seguente formula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4.9}$$

dove d_k è la dimensione della matrice Keys e Query. La divisione con $\sqrt{d_k}$ serve per ottenere un gradiente più stabile. Mentre il sotto-modulo Multi-Head Attention utilizza il primo per determinare i coefficienti di attenzione su diverse varianti delle matrici di input. In particolare si calcolano h trasformazioni lineari delle matrici Query, Key e Value con le quali determina diversi coefficienti di attenzioni al fine di migliorare le prestazioni di apprendimento. L'intero processo può essere rappresentato dalla seguente funzione:

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, head_2, ..., head_h)W^O \\ dove\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4.10}$$

dove W_i^Q, W_i^K, W_i^V, W^O sono matrici di trasformazione lineare. L'output del modulo Multi-head self-attention sarà una trasformazione lineare delle matrici che rappresenta la concatenazione dei coefficienti d'attenzione proveniente dalle diverse

varianti delle matrici Key, Query e Value. Quindi, in generale, questo sotto-modulo permette di osservare diverse parti della sequenza in modo differente ogni volta consentendo al modello di catturare meglio le informazioni sulle posizioni perché ogni *head* si occuperà di diversi segmenti dell'input. La loro combinazione darà, quindi, una rappresentazione più robusta. Inoltre ogni head catturerà anche diverse informazioni contestuali, correlando le parole in modo unico.

Dopo la sezione Multi-head self-attention, sia nell'encoder che nel decoder, il risultato ottenuto vine utilizzato da una rete feed-forward. Questa rete ha i parametri identici per ogni posizione nella sequenza ma diversi per ogni strato, in pratica può essere vista come un'altra trasformazione lineare separata e identica di ciascun elemento dalla sequenza data.

Il modello di Trasformer utilizzato per la predizione delle traiettorie del dataset sCREEN è più semplice rispetto alla versione tradizionale perchè viene utilizzato solo l'encoder e al posto del decoder viene inserito uno strato neurale di output che ridimensiona l'output dell'encoder ad alta dimensionalità in due coordinate cartesiane.

Capitolo 5

Esperimenti

In questo capitolo verranno descritti gli esperimenti condotti sui modelli, al fine di ottenere delle buone predizioni e qual'è stato il protocollo di valutazione utilizzato. Successivamente verrà presentata un'analisi quantitativa per mostrare l'efficacia dei modelli provati e infine un analisi qualitativa sulle traiettorie predette.

5.1 Dettagli sull'implementazione

Durante le prime fasi di test sui modelli sono state utilizzate 3 versioni del dataset sCREEN, ovvero una versione con resampling a 0.4 secondi e 2 con resampling a 1 secondo. Le versioni con lo stesso sampling rate differenziano dal fatto che in una le coordinate dei punti sono state modificate nel caso le posizioni di un consumatore siano state registrate erroneamente all'interno degli scaffali del negozio.

Sebbene lo standard, nella comunità scientifica nel campo delle predizioni di traiettorie, utilizzi un intervallo di 0.4 secondi tra una posizione e la successiva, questo protocollo è stato scartato nelle fasi più avanzate dello studio. Infatti si è osservato che nonostante i vari modelli ottenessero risultati migliori, a livello quantitativo, la lunghezza delle traiettorie predette era troppo piccola rispetto alle dimensioni del negozio e quindi non catturavano le dinamicità dei vari percorsi fatti dai clienti.

Per quanto riguarda la versione del dataset con i punti modificati non è mai stata utilizzata per ottenere dei dati concreti. Infatti era stata creata al fine di utilizzarla in altri modelli che andavano a implementare una semantica sugli ostacoli presenti nel negozio. Purtroppo a causa di molti ritardi, dovuti principalmente alla difficoltà di allenamento dei vari modelli descritti nel paragrafo precedente, non è stato possibile testare altre soluzioni. Tuttavia qualche test su questo dataset è stato fatto ma i risultati delle metriche risultavano peggiori rispetto al dataset originale,

questo significa che le modifiche fatte sulle coordinate dei punti hanno portato del rumore aggiuntivo influenzando negativamente i modelli.

Per questi motivi i vari modelli sono stati allenati con la versione del dataset con resampling a 1 secondo senza modifica delle coordinate dei punti. Le 4283 traiettorie del dataset sono state suddivise in due parti ovvero 3425 per il training, quindi 80%, di cui il 10% viene utilizzato per il validation set mentre le rimanenti 858 sono state impiegate per il test set.

Come spiegato nel paragrafo 3.5 i dati del datset sCREEN sono stati organizzati in forma di frame ovvero una struttura che rappresenta cosa sta avvenendo nella scena in un determinato intervallo di tempo. Prima però di essere utilizzati dai modelli i dati subiscono un ulteriore pre-processing al fine di rendere l'allenamento più veloce ed efficace.

In particolare, preso un intervallo di frame, si vanno a determinare quali siano le traiettorie dei pedoni presenti per poi andare ad estrarre delle sotto-traiettorie che sono contenute nell'intervallo selezionato. Successivamente le sotto-traiettorie vengono raggruppate in *batch* dove ogni batch contiene 32 sotto-percorsi. Inoltre viene fatta anche una selezione delle sotto-traiettorie che andranno a formare il batch scartando quelle che:

- terminano prima del periodo di osservazione;
- hanno un numero di frame minore o uguale a 5;
- hanno come distanza euclidea tra il primo e l'ultimo frame, sia del periodo di osservazione che di predizione, minore o uguale a 1. Quindi non si tengono in considerazione tutte le sotto-traiettorie che sono troppo statiche o che hanno andamenti molto complessi e quindi impossibili da prevedere.

E' stato considerato anche un approccio *sliding window* sugli intervalli ovvero dato un periodo di frame il successivo non iniziava dopo l'ultimo frame dell'intervallo precedente ma dopo una costante dal primo frame. Tuttavia questo approccio impediva un corretto allenamento dei vari modelli e quindi non è stato utilizzato.

Inoltre, sempre per migliorare l'allenamento dei modelli, tutte le coordinate della parte di osservazione delle sotto-traiettorie sono state scalate rispetto a quelle dell'ultimo frame.

Per valutare i modelli sono stati presi in considerazione due protocolli di predizione ovvero *Short range* dove si osservano 8 frame per una durata di 8 secondi e si

cerca di predire i successivi 12 per un totale di 12 secondi e *Long range* nel quale si osservano 5 frame per predire e 30 consecutivi.

Per poter paragonare le traiettorie predette e valutarne la bontà sono state utilizzate le metriche *Average Displacement Error (ADE)* e *Final Displacement Error (FDE)*. La metrica ADE si riferisce all'errore quadratico medio tra tutti i punti stimati di una traiettoria e quelli originali su tutti i pedoni.

$$ADE = \frac{\sum_{i=1}^{N} \sum_{T_{obs}}^{T_{pred}} [(\hat{x}_i^t - x_i^t)^2 + (\hat{y}_i^t - y_i^t)^2]}{n(T_{pred} - T_{obs})} \tag{5.1}$$

dove N è il numero di pedoni e le coordinate $(\hat{x}_i^t, \hat{y}_i^t)$ indicano le posizioni dei punti predette dal modello. Mentre FDE esprime la distanza tra la destinazione finale prevista e la destinazione reale come media su tutti i pedoni.

$$FDE = \frac{\sum_{i=1}^{N} \sqrt{(\hat{x}_i^{T_{pred}} - x_i^{T_{pred}})^2 + (\hat{y}_i^{T_{pred}} - y_i^{T_{pred}})^2}}{n} \tag{5.2}$$

Inoltre per il modello LSTM sono stati testati altri tre protocolli ibridi che puntano a migliorare i risultati del protocollo Long range. In particolare si considera lo stesso intervallo di frame, ovvero di 35, ma viene aumentata la parte di osservazione in questo modo il modello può osservare una parte più grande di traiettoria e predirne una porzione più piccola riducendo così l'errore. I protocolli testati prevedono di:

- osservare 15 frame e predirne 20
- osservare 20 frame e predirne 15
- osservare 25 frame e predirne 10

In generale i modelli LSTM, utilizzati per valutare i diversi protocolli di predizioni, sono stati allenati con optimizer Adam e learning rate fissato a 0.0001. Inoltre sono state utilizzate solo le sotto-traiettorie con lunghezza euclidea maggiore di 1 e successivamente codificate in vettori di dimensione 32. Per generare più previsioni equamente fattibili rispetto ad uno stesso periodo di osservazione vengono sommate due matrici di rumore rispettivamente all'input codificato e all'output della cella LSTM prima dello strato di output. Il rumore viene mitigato attraverso una costante moltiplicativa α. Naturalmente l'utilizzo di protocolli di previsione diversi ha richiesto modifiche ad hoc, in particolare per il protocollo:

- *Short range* è stato utilizzato un Cosine Annealing come learning schedule con l'aggiunta di regolarizzazione L2 con tasso 0.001 mentre la costante α di noise è stata impostata con valore 0.005. In questo caso viene utilizzata una dimensione di cella LSTM di 256.

- *Long range* viene impostato un valore di 0.005 come regolarizzazione L2 e costante α di noise, inoltre viene aumenta la dimensione di cella LSTM a 512.

- *15-20* vengono usati gli stessi parametri del protocollo Short range ma la costante α di noise viene abbassata a 0.0005 ed aumenta la dimensione di cella LSTM a 512.

- *20-15* è stato utilizzato un valore di α pari a 0.005 con valore di regolarizzazione L2 di 0.05 e dimensione di cella LSTM di 256.

- *25-10* sono stati usati gli stessi parametri del protocollo Short range.

Invece per i modelli deterministici, dato che non hanno bisogno di allenamento, vengono influenzati da molti meno parametri. Infatti il modello Costant Velocy model, abbreviato con Costant-model, crea una serie di traiettorie possibili che sono distanti l'una dall'altra da un determinato angolo. Al fine di avere un area circoscritta è stato utilizzato un valore di raggio di 25°. Mentre il modello basato su Brownian Motion, denominato con Brownian-model, usa una distribuzione normale di probabilità con media zero per estrarre dei valori con i quali generare le traiettorie. E' stato scelto un valore di devianza standard, per la distribuzione normale, di 0.3 che permette di creare traiettorie compatte e non troppo dispersive.

Per quanto riguarda il Trasformer encoder è stato allenato con Stochastic gradient descent (SGD) e learning rate fissato a 0.0001. Sono state utilizzate solo le sotto-traiettorie con lunghezza euclidea maggiore di 1 e successivamente codificate in vettori di dimensione 256. Inoltre il Trasformer è stato impostato come una catena di 4 encoder dove ognuno ha 8 head e una dimensione della rete completamente connessa di 2048. Per generare più previsioni, come per LSTM, è stata aggiunta all'output del modello una matrice di rumore che viene moltiplicata da una costante α che varia in base al protocollo, infatti per *Short range* viene impostata a 0.05 mentre per *Long range* a 0.01.

Per tutti i modelli di Deep Learning viene utilizzata la funzione *Mean Square Error* come funzione di loss.

5.2 Analisi quantitativa

Short range				
	Brownian-model	Costant-model	LSTM	Trasformer encoder
ADE	1,9209	1,2283	0,9477	**0,8534**
FDE	3,4255	2,7425	1,8312	**1,6516**

Tabella 5.1: Risultati delle metriche ADE e FDE ottenuti dai vari modelli con i protocollo Short Range

La tabella 5.1 mostra i risultati quantitativa ottenuti dai diversi modelli sul dataset sCREEN usando il protocollo Short range. Il modello Trasformer encoder ottiene risultati delle metriche ADE e FDE migliori rispetto agli altri modelli testati. Tuttavia il miglioramento non è così significativo per quanto riguarda la metrica FDE che esprime la distanza tra la destinazione finale prevista e la destinazione reale come media su tutti i pedoni. Questo significa che entrambi i modelli, mediamente, commettono circa lo stesso errore nella previsione della destinazione.

Long range				
	Brownian-model	Costant-model	LSTM	Trasformer encoder
ADE	3,8201	4,5461	2,5388	**2,3755**
FDE	6,1011	10,3605	4,2986	**4,0153**

Tabella 5.2: Risultati delle metriche ADE e FDE ottenuti dai vari modelli con i protocollo Long Range

Invece la tabella 5.2 mostra i risultati ottenuti dai modelli testati sul dataset sCREEN usando il protocollo Long range. In questo caso i valori delle metriche risultano essere, per tutti i modelli, peggiori rispetto a quelli ottenuti con il protocollo Short range. Questo esprime la difficoltà dei modelli nel fare predizioni a lungo termine con un periodo di osservazione limitato in un ambiente così variabile. Tuttavia il modello Trasformer encoder anche in questo caso riesce ad ottenere prestazioni migliori.

	Short range	Long range	15-20	20-15	25-10
ADE	0,9477	2,5388	1,8421	1,2830	**0,6520**
FDE	1,8312	4,2986	3,5938	2,4463	**1,3065**

Tabella 5.3: Risultati delle metriche ADE e FDE ottenute dal modello LSTM su diversi protocolli di previsione

La tabella 5.3 mette a confronto le prestazioni del modello LSTM su diversi protocolli di predizione. I risultati peggiori si ottengono nel protocollo Long range e questo è dovuto principalmente al fatto che le traiettorie dei pedoni, all'interno

del negozio, sono molto variabili con cambi di direzione improvvisi e velocità non costante. Casi comuni sono ad esempio un pedone che si dirige verso una corsia del negozio ma inaspettatamente cambia completamente destinazione o addirittura torna indietro, oppure quando rallenta o si ferma davanti uno scaffale per cercare un prodotto. Questi comportamenti non lineari sono del tutto impossibili da predire per il modello LSTM specialmente con un periodo di osservazione ridotto. Un modo per ottenere prestazioni migliori è ridurre notevolmente il periodo di previsione rispetto a quello di osservazione come nel protocollo 25-10 dove si riesce ad ottenere risultati migliori anche rispetto al protocollo Short range.

5.3 Analisi qualitativa

Le figure 5.1 e 5.2 mostrano un confronto qualitativo delle traiettorie predette dai diversi modelli rispetto al protocollo Short range. In entrambe le figure, come suggerito anche dall'analisi quantitativa, il modello Brownian-model ottiene i risultati peggiori, infatti la generazione casuale delle traiettorie non porta a predizioni fattibili. Mentre il modello Costant-model, sebbene nella sua semplicità, riesce almeno a predire correttamente la direzione del percorso ma non riesce a generare traiettorie articolate. Invece i modelli di Deep Learning ottengono risultati decisamente migliori andando a imitare pure l'andamento non lineare dei percorsi dei pedoni.

Tuttavia, sebbene il modello LSTM surclassi i due modelli deterministici per quanto riguarda le performance, il modello che riesce ad ottenere percorsi più corretti è il Trasformer encoder che predice, sebbene con una certa soglia di errore, anche traiettorie con un cambio importante di direzione come quella mostrata nella figura 5.1. Anche nella figura 5.2, nonostante il modello LSTM riesca a determinare sia la curva che la direzione della traiettoria, il modello Trasformer encoder genera una predizione più verosimile confermando i risultati dell'analisi quantitativa.

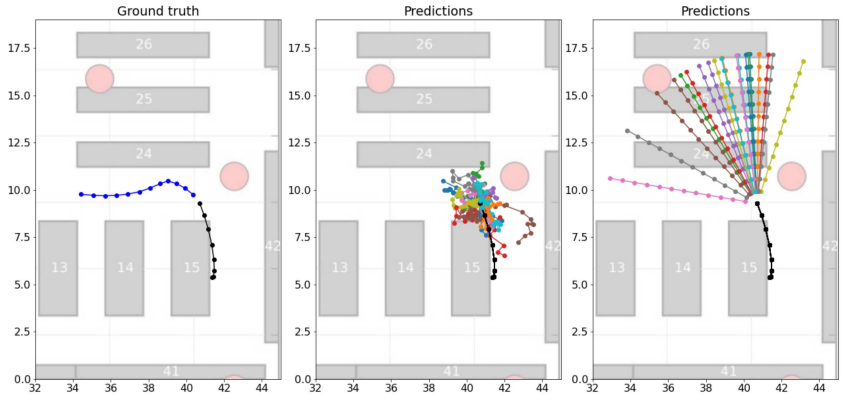
I percorsi dei clienti all'interno di un supermercato sono tipicamente complessi e questa caratteristica si ritrova anche nelle traiettorie estratte dal dataset sCREEN dato che un consumatore può cambiare direzione improvvisamente, fermarsi per cercare un prodotto o tornare indietro perché ha sbagliato corsia. Di conseguenza fare previsioni sempre più a lungo termine con un periodo di osservazione ristretto diventa un compito difficile perché il modello deve tenere in considerazione molte più variabili. La figura 5.3 mostra un esempio di traiettoria comune nel dataset con più cambi di direzione. Purtroppo queste caratteristiche rendono la traiettoria imprevedibile per i modelli, infatti sia LSTM e Trasformer encoder non sono in

grado di fornire una previsione verosimile. Diversamente se la traiettoria è più regolare, come quella mostrata in figura 5.4, allora si possono ottenere risultati più incoraggianti. In questi casi, come per il protocollo Short range, il Trasformer encoder riesce ad ottenere risultati migliori rispetto a LSTM raggiungendo quasi il punto di destinazione. Tuttavia fare previsioni così a lungo termine, come quelle del protocollo Long range, in un ambiente così complesso e ricco di ostacoli porta ad ottenere previsioni non sempre accurate.

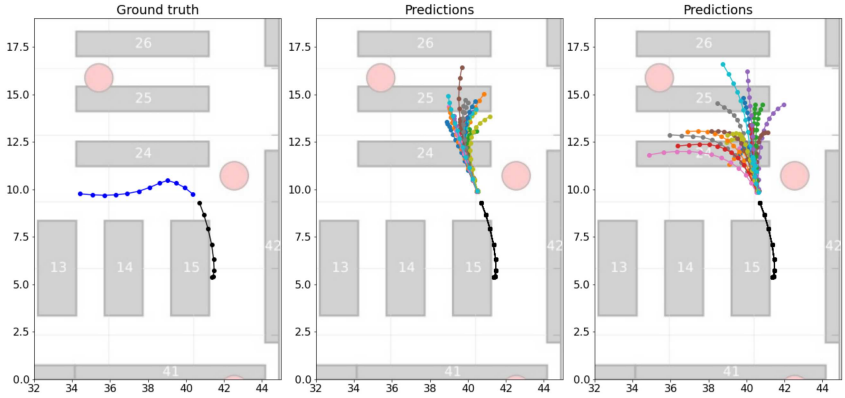
Per quanto riguarda i modelli deterministici non riescono a fare previsioni soddisfacenti in un intervallo temporale così lungo confermando la superiorità dei modelli di Deep learning in tali ambienti complessi.

La figura 5.5 mostra alcune traiettorie prodotte dal modello LSTM per i protocollo 15-20, 20-15 e 25-10. Nella maggioranza dei casi con il protocollo 25-10 si ottengono traiettorie più realistiche come mostrato nella figura 5.5b. Infatti con il protocollo 15-20, LSTM non riesce a predire traiettorie con un cambio di direzione troppo netto tuttavia, se il periodo di osservazione aumenta e quello di predizione diminuisce, in modo da avere previsioni per l'immediato futuro, il modello riesce a generare delle traiettorie accettabili. Comunque esistono casi, come quello mostrato in figura 5.5a, dove anche con un lungo periodo di predizione, ad esempio nel protocollo 15-20, si riesce ad avere buone previsioni.

In generale LSTM riesce a determinare cambi di direzione se questi vengono introdotti nel periodo di osservazione e sono graduali nel tempo oppure possono essere abbastanza improvvisi ma devono riguardare solo le ultime posizioni della traiettoria.

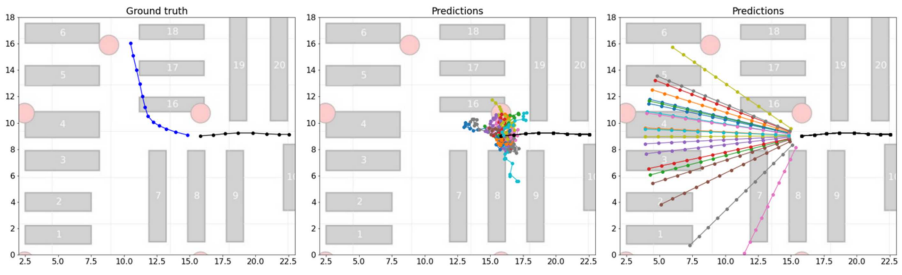


(a) Modelli deterministici

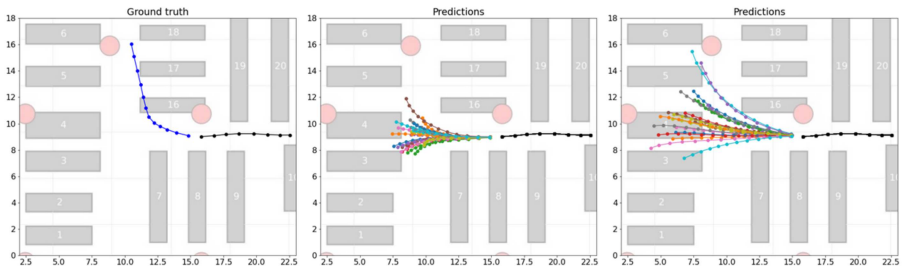


(b) Modelli di Deep Learning

Figura 5.1: Esempio di traiettorie predette per il protocollo Short range. Nelle figure 5.1a-A e 5.1b-A le traiettorie da predire. In 5.1a-B e 5.1a-C i risultati dei modelli deterministici Brownian-model e Costant-model mentre nelle figure 5.1b-B e 5.1b-C le traiettorie predette dai modelli LSTM e Trasformer encoder. In particolare, nei risultati dei vari modelli, i punti della traiettoria segnati in nero sono le posizioni osservate dal modello mentre gli altri rappresentano i punti predetti.

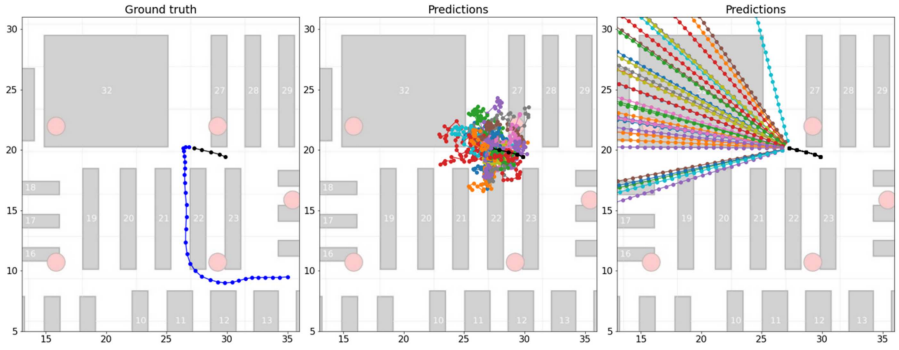


(a) Modelli deterministici

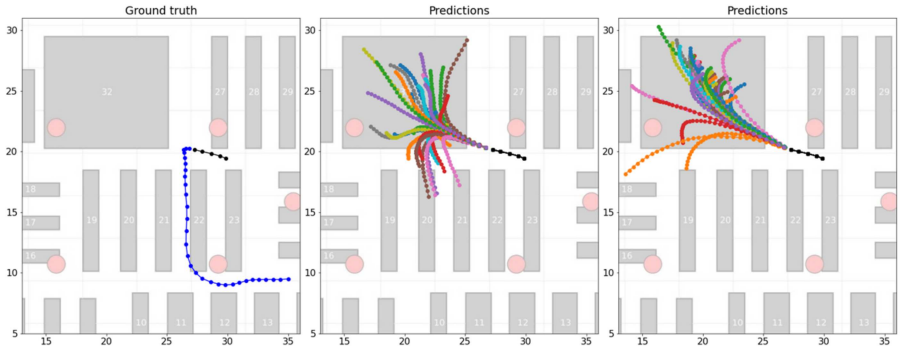


(b) Modelli di Deep Learning

Figura 5.2: Esempio di traiettorie predette per il protocollo Short range. Nelle figure 5.2a-A e 5.2b-A le traiettorie da predire. In 5.2a-B e 5.2a-C i risultati dei modelli deterministici Brownian-model e Costant-model mentre nelle figure 5.2b-B e 5.2b-C le traiettorie predette dai modelli LSTM e Trasformer encoder.

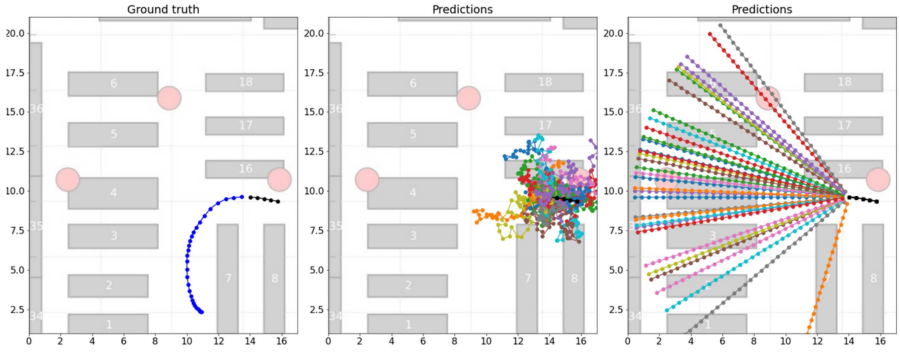


(a) Modelli deterministici

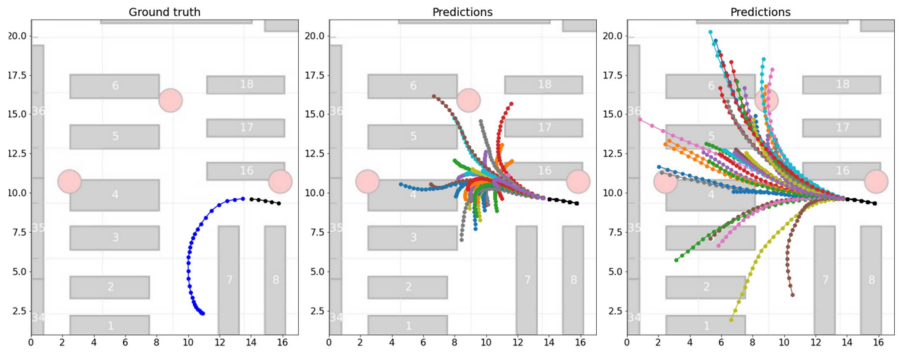


(b) Modelli di Deep Learning

Figura 5.3: Esempio di traiettorie predette per il protocollo Long range. Nelle figure 5.3a-A e 5.3b-A le traiettorie da predire. In 5.3a-B e 5.3a-C i risultati dei modelli deterministici Brownian-model e Costant-model mentre nelle figure 5.3b-B e 5.3b-C le traiettorie predette dai modelli LSTM e Trasformer encoder.

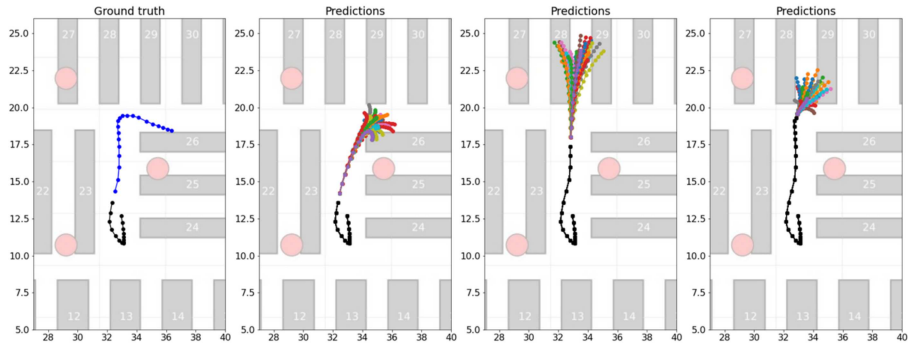


(a) Modelli deterministici

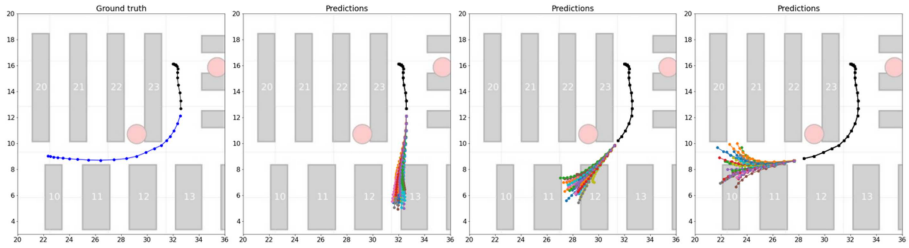


(b) Modelli di Deep Learning

Figura 5.4: Esempio di traiettorie predette per il protocollo Long range. Nelle figure 5.4a-A e 5.4b-A le traiettorie da predire. In 5.4a-B e 5.4a-C i risultati dei modelli deterministici Brownian-model e Costant-model mentre nelle figure 5.4b-B e 5.4b-C le traiettorie predette dai modelli LSTM e Trasformer encoder



(a) Esempio 1 di traiettoria



(b) Esempio 2 di traiettoria

Figura 5.5: Esempio di traiettorie predette dal modello LSTM su diversi protocolli di previsione. Nella figure 5.5a-A e 5.5b-A le traiettorie da predire, mentre le successive immagine rappresentato le traiettorie prodotte con il protocollo 15-20, 20-15 e 25-10.

Capitolo 6

Conclusioni

Il progetto di tesi sviluppato è nato con l'obbiettivo di verificare se era possibile utilizzare il dataset sCREEN, creato per il task di predizione di punti di attrazione all'interno di un negozio di vendita al dettaglio, per l'estrazione dei percorsi dei clienti al fine di utilizzarli per il task di predizione di traiettorie.

L'utilizzo di questo dataset, creato per un contesto diverso, ha portato numerose sfide sia nella fase di estrazione e pre-processing delle traiettorie sia nell'allenamento dei vari modelli utilizzati.

Una delle difficoltà più grandi affrontante è stata l'estrazione delle traiettorie. Infatti il dataset è stato strutturato come un flusso continuo di posizioni ordinate temporalmente quindi non è presente nessuna indicazione su quando inizia e finisce una specifica traiettoria. Inoltre non vengono fornite zone particolari dove tipicamente inzia e finisce il percorso di un cliente come ad esempio l'area d'entrata del supermercato dove prendere i carrelli o la zona delle casse dove di solito un utente termina la propria spesa. La mancanza di queste regioni che potevano essere utilizzate per determinare in modo abbastanza preciso dove iniziano e finiscono le traiettorie con l'aggiunta del fatto che le prime posizioni registrate da alcuni tag sono già all'interno del negozio non ha reso facile il compito di estrazione dei percorsi. Quindi un primo approccio che è stato utilizzato per determinare le traiettorie si basava su una indicazione lasciata dagli autori del sistema ovvero una traiettoria termina quando il suo tag rimane fermo per un minimo di 5 minuti. Purtroppo si è scoperto successivamente che le posizioni raccolte non sono precise ma contengo del rumore che varia in base ai tag rendendo difficile determinare quando effettivamente un carrello o cestino è fermo. In aggiunta le posizioni raccolte non sono fornite di unità di misura impedendo di capire quanto sia importante il rumore presente nelle varie misurazioni.

I dispositivi utilizzati per tracciare i clienti contengono un accelerometro per capire se il carrello o cestino è in movimento tuttavia alcuni tag sono difettosi e continuano ad inviare il loro stato anche se restano immobili per un lungo periodo di tempo. Inoltre nel dataset non viene neanche specificato se una posizione è stata raccolta da un cestino o carrello, tale informazione sarebbe stata utile per determinare se i tag difettosi erano principalmente nei carrelli o nei cestini in modo da isolare una categoria specifica. Nel dataset mancano anche indicazione sulla struttura del negozio dove sono state registrate le posizioni dei clienti, infatti è presente solo una cartina topologica che da una idea vaga di come è fatto il punto di vendita non specificando ne la dimensione ne come sono strutturati i vari scaffali. Ad esempio è stato supposto che gli scaffali 32 e 33, mostrati nella figura 3.1, fossero delle isole dove i clienti potessero entrare ma non è una teoria confermata.

Tuttavia, nonostante numerose difficoltà e diversi metodi testati, è stato possibile estrarre dal dataset ben 4823 traiettorie diverse.

Un ulteriore sfida affrontata, dopo l'estrazione delle traiettorie, è stata trovare il pre-processing migliore per elaborare i dati prima di essere utilizzati dai modelli. Infatti sono stati condotti numerosi esperimenti sul modello LSTM per creare batch che portassero ad un buon allenamento. Si è notato, a differenza delle traiettorie di altri dataset raccolte in ambienti aperti, come quelle del dataset sCREEN siano molto più complesse a causa di uno scenario indoor. Infatti sono comuni i casi di cambio di direzione improvvisi o di soste davanti uno scaffale per la ricerca di un prodotto. Questi comportamenti rendono difficile l'allenamento specialmente il secondo. Nella pratica si è osservato che avere traiettorie estremamente corte o immobile non permette ai modelli di generare una adeguata predizione portando un valore non corretto delle metriche facendo credere che il modello utilizzato ottenga delle buone prestazioni a livello quantitativo.

In conclusione è stato possibile allenare i modelli con il dataset sCREEN dimostrando che il modello Trasformer encoder bastaso su self-attention riesce ad ottenere risultati migliori di LSTM. Inoltre i numerosi esperenti condotti hanno provato che il dataset sCREEN è ideale per predizioni di traiettorie di pedoni su un raggio temporale futuro breve dato lo scenario indoor molto complesso a causa sia del contesto sia dalla presenza di numerosi ostacoli. Inoltre anche in ambiente indoor viene confermata la superiorità di modelli di Deep Learning rispetto a modelli deterministici di base.

6.1 Sviluppi futuri

I risultati raggiunti indicano come sia possibile produrre predizioni di traiettorie di pedoni in un ambiente indoor. La riuscita di questo progetto rappresenta una prima esplorazione nello studio di questo dataset e quali sono le sue reali potenzialità. Inoltre i risultati ottenuti invitano a procedere nella ricerca per trovare e sperimentare nuovi modelli al fine di ottenere prestazioni migliori, ad esempio sarebbe interessante:

- verificare se i risultati ottenuti dal Trasformer encoder possono essere migliorati introducendo anche la parte decoder del modello;
- capire quali sono le vere potenzialità dei Trasformer in questo campo utilizzando modelli più complessi;
- utilizzare modelli che tengono in considerazione le caratteristiche dell'ambiente per la generazione delle predizioni in modo da creare percorsi più realistici;
- provare a considerare le iterazioni sociali tra i clienti nel negozio per ottenere traiettorie migliori.

Gli spunti espressi in precedenza potrebbero essere gli argomenti chiave alla base di altre ricerche future.

Bibliografia

[1] Geiger A., P. Len e R Urtasun. «Are we ready for autonomous driving? the kitti vision benchmark suite». In: (2012). URL: 'https://homepages.inf.ed.ac.uk/rbf/FORUMTRACKING/'.

[2] Hinneburg A. e Keim D. «An Efficient Approach to Clustering in Large Multimedia Databases with Noise». In: (1998).

[3] Lerner A., Chrysanthou Y. e Lischinski D. «Crowds by example». In: (2007).

[4] Robicquet A.and Sadeghian A., Alahi A. e Savarese S. «Learning social etiquette: Human trajectory understanding in crowded scenes». In: (2016). URL: https://cvgl.stanford.edu/projects/uav_data/.

[5] Alexandre Alahi et al. «Social LSTM: Human Trajectory Prediction in Crowded Spaces». In: ().

[6] Saad Albawi, Tareq Abed Mohammed e Saad Al-Zawi. «Understanding of a Convolutional Neural Network». In: (2017).

[7] E. Alpaydin. «Introduction to MAchine Learning». In: (2014).

[8] J. Aneja, A. Deshpande e A. G. Schwing. «PConvolutional image captioning». In: (2018).

[9] Majecka B. «Statistical models of pedestrian behaviour in the forum». In: (2009). URL: https://homepages.inf.ed.ac.uk/rbf/FORUMTRACKING/.

[10] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. «Neural Machine Translation by Jointly Learning to Align and Translate». In: (2015).

[11] Pierre Baldi. «Gradient Descent Learning Algorithm Overview: A General Dynamical Systems Perspective». In: (1995).

[12] Y. Bengio, P. Frasconi e P. Simard. «The problem of learning long-term dependencies in recurrent networks». In: (1993).

[13] Derya Birant e Alp Kut. «ST-DBSCAN: An algorithm for clustering spatial–temporal data». In: (2006).

[14] Kyunghyun Cho et al. «Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation». In: (2014).

[15] Antonia Creswell et al. «Generative Adversarial Networks: An Overview». In: (2018).

[16] Liciotti D., Zingaretti P. e Placidi V. «An automatic analysis of shoppers behaviour using a distributed rgb-d cameras system». In: (2014).

[17] Li Deng e Dong Yu. «Deep Learning, Methods and Applications». In: (2014).

[18] J. Devlin et al. «Bert pre-training of deep bidirectional transformers for language understanding». In: (2019).

[19] Frontoni E. et al. «Information management for intelligent retail environment: the shelf detector system». In: (2014).

[20] Tharindu Fernando et al. «Soft + Hardwired Attention: An LSTM Framework for Human Trajectory Prediction and Abnormal Event Detection». In: (1995).

[21] J. Gehring et al. «Convolutional sequence to sequence learning». In: (2017).

[22] Felix A. Gers, J¨urgen Schmidhuber e Fred Cummins. «Learning to Forget: Continual Prediction with LSTM. Neural Computation». In: (2000).

[23] Francesco Giuliari et al. «Transformer Networks for Trajectory Forecasting». In: ().

[24] Agrim Gupta et al. «Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks». In: ().

[25] Sandhya Harikumar. «K-Medoid Clustering for Heterogeneous DataSets». In: (2015).

[26] Irtiza Hasan et al. «Mx-lstm: mixing tracklets and vislets to jointly forecast trajectories and head poses». In: (2018).

[27] Sepp Hochreiter, Fakultat fur Informatik e Jurgen Schmidhuber. «Long-short term memory». In: (1997).

[28] Yingfan Huang et al. «STGAT: Modeling Spatial-Temporal Interactions for Human Trajectory Prediction». In: ().

[29] Rob J Hyndman e George Athanasopoulos. *Forecasting: Principles and Practice - Capitolo 7*. URL: `https://otexts.com/fpp2/`.

[30] Daniele Liciotti et al. «Shopper analytics: A customer activity recognition system using a distributed rgb-d camera network.» In: (2014).

[31] Aristidis Likas, Nikos Vlassis e Jakob Verbeek. «The global k-means clustering algorithm». In: (2001).

[32] Matteo Lisotto, Pasquale Coscia e Lamberto Ballan. «Social and Scene-Aware Trajectory Prediction in Crowded Spaces». In: (2019).

[33] Ankerst M. et al. «PTICS: Ordering points to identify the clustering structure». In: (1999).

[34] Ester M. et al. «A density-based algorithm for discovering clusters in large spatial databases with noise». In: (1996).

[35] Schneider N. e Gavrila D.M. «Pedestrian path prediction with recursive bayesian filters: A comparative study». In: (2013). URL: http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/Pedestrian_Path_Predict_GCPR_1/pedestrian_path_predict_gcpr_1.html.

[36] Marina Paolanti et al. «Modelling and Forecasting Customer Navigation in Intelligent Retail Environments». In: (2017).

[37] Amir Rasouli. «Deep Learning for Vision-based Prediction: A Survey». In: (2020).

[38] Guha S., Rastogi R. e Shim K. «URE: An efficient clustering algorithm for large databases». In: (1998).

[39] Oh S. et al. «A large-scale benchmark dataset for event recognition in surveillance video.» In: (2011). URL: https://viratdata.org/.

[40] Pellegrini S. et al. «You'll never walk alone: Modeling social behavior for multi-target tracking». In: (2009).

[41] Yi S., H. Li e X Wang. «Understanding pedestrian behaviors from stationary crowd groups.» In: (2015).

[42] Christoph Scholler1 et al. «What the Constant Velocity Model Can Teach Us About Pedestrian Motion Prediction». In: (2020).

[43] M. Schuster e K.K. Paliwal. «Bidirectional recurrent neural networks». In: (1997).

[44] Gholamhosein Sheikholeslami, Suro jit Chatterjee e Aidong Zhang. «WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases». In: (1998). URL: http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput695-00/papers/vldb98.pdf.

[45] Gholamhosein Sheikholeslami, Suro jit Chatterjee e Aidong Zhang. «WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases». In: (1998). URL: http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput695-00/papers/vldb98.pdf.

[46] M. Somalvico. «Intelligenza Artificiale». In: (1987).

[47] A. Sperduti. *Apprendimento Automatico*. URL: https://www.math.unipd.it/~sperduti/ML10/ml_intro_nozioni_base.pdf.

[48] Mark Stone. *What is ultra-wideband, and how does it work?* 2021.

[49] Hang Su et al. «Forecast the plausible paths in crowd scenes». In: ().

[50] Ilya Sutskever, Oriol Vinyals e Quoc V Le. «Sequence to sequence learning with neural networks». In: (2014).

[51] Zhang T., Ramakrishnan R. e Livny M. «BIRCH: An efficient data clustering method for very large databases». In: (1996).

[52] Ashish Vaswani et al. «Attention Is All You Need». In: (2017).

[53] Anirudh Vemula, Katharina Muelling e Jean Oh. «Social attention: Modeling attention in human crowds». In: (2018).

[54] Wei Wang, Jiong Yang e Richard Muntz. «STING : A Statistical Information Grid Approach to Spatial Data Mining». In: (1997). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.3370&rep=rep1&type=pdf.

[55] Ronald J. Williams e David Zipser. «A learning algorithm for continually running fully recurrent neural networks». In: (1989).

[56] www.oracle.com. *What is machine learning*. URL: https://www.%20oracle.com/it/data-science/machine-learning/what-is-machinelearning.

[57] Simone Zambonia et al. «Pedestrian Trajectory Prediction with Convolutional Neural Networks». In: (2021).