Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Knowledge Transfer for Human Trajectory Prediction

Master Candidate

Luca Scattolaro

Student ID 2019157

Supervisor

Prof. Lamberto Ballan

University of Padova

Co-supervisor

Sourav Das

University of Padova

Date: 3 October 2022
Academic Year: 2021/2022

Abstract

Human trajectory prediction aims to analyze human future movements given their past positions and is an important topic in several application domains such as socially-aware robots, intelligent tracking systems and self-driving cars. The goal of the current work is to learn representations from already seen scenes and use transfer learning to transfer this knowledge from the training dataset to a new test set. In order to do this, we create a module that, for each human in the scene, extracts local features from a patch around the agents current position. To test its predictive capability, the proposed module was embedded into two model architectures, namely SAR and GoalSAR. Specifically, the resulting feature descriptors coming from our approach are concatenated to the lightweight attention-based recurrent backbone that acts solely on past observed positions for both the aforementioned architectures. We conducted extensive experiments using different features extractors and training approaches, such as training on the SDD dataset and test on the ETH/UCY dataset.

Sommario

La previsione delle traiettorie umane mira ad analizzare i movimenti futuri degli esseri umani in base alle loro posizioni passate ed è un argomento importante in diversi domini applicativi, come i robot sociali, i sistemi di tracciamento intelligenti e le auto a guida autonoma.
L'obiettivo di questa tesi è quello di apprendere rappresentazioni semantiche da scene già viste e di utilizzare la tecnica di transfer-learning per trasferire questa conoscenza dal training set al test set.
A tal fine, abbiamo creato un modulo che, per ogni essere umano presente nella scena, estrae le caratteristiche semantiche locali da un patch attorno alla posizione corrente dell'agente. Per testare la sua capacità predittiva, il modulo proposto, è stato incorporato in due architetture chiamate SAR e GoalSAR. In particolare, i features descriptors risultanti dal nostro approccio sono concatenati all'attention-based recurrent backbone, che agisce esclusivamente sulle posizioni osservate in passato per entrambe le architetture citate. Abbiamo condotto esperimenti approfonditi utilizzando diversi features extractors e diversi approcci di addestramento, come l'addestramento sul dataset SDD e il test sul dataset ETH/UCY.

Contents

List of Figures

List of Tables

List of Acronyms

FAST Features from Accelerated Segment Test

BRIEF Binary Robust Independent Elementary Features

SIFT Scale-Invariant Feature Transform

ORB Oriented FAST and Rotated BRIEF

PECNet Predicted Endpoint Conditioned Network

VAE Variational Autoencoder

MUSE VAE Multi-Scale Variational Autoencoder

CVAE Conditional Variational Autoencoder

GAN Generative Adversarial Networks

MG-GAN Modified Generator - Generative Adversarial Networks

LSTM Long Short-Term Memory

GRU Gated Recurrent Units

SAR Self-Attentive Recurrent backbone

Goal-SAR Goal-driven Self-Attentive Recurrent Networks

1 Introduction

Modeling human motion is essential for autonomous systems that operate in public spaces.
For example autonomous vehicles need to foresee future positions to avoid collisions, social robots require to move naturally alongside humans and it can also be used for video surveillance tasks to support human operators in order to intervene promptly.
Human trajectory forecasting is an extremely difficult problem, due to physical, social and mental factors that collectively influence peoples trajectories.

1.1 Problem Formulation

The goal of pedestrian trajectory prediction is to predict the future positions of a pedestrian given its previous position.
Concretely, given a scene where pedestrians are present, their coordinates are observed for a certain amount of time, called T_{obs}, and the task is to predict the future coordinates of each pedestrian from T_{obs} to T_{pred}.
There are two different types of trajectory prediction: the short-term and the long-term one. In the short term case we are dealing with 3-5 seconds while on long-term case with 5-30 seconds, all the experiments done for this thesis considered the short-term trajectory prediction problem. The position of each pedestrian is characterized by its (x, y) coordinates with respect to a fixed point. Moreover, thanks to a discretization of time, the difference between time t and time $t + 1$ is the same as the difference between time $t + 1$ and time $t + 2$.

1.2 Applications

Modelling how people move in their environment and in the presence of other people can benefit different areas:

- Crowd surveillance: systems able to predict pedestrian behaviour are also capable of detecting anomalies and unexpected behaviours to support human operators in order to intervene promptly.
- Social and autonomous robots: robots that operate in public environments need to move naturally alongside with humans and to avoid accidentally hitting them.
- Autonomous driving: self-driving vehicles have to understand the intentions of pedestrians and predict their future trajectories to safely operate.

1.3 Purpose

The literature on pedestrian trajectory prediction is extensive, however, there are still some topics that can be investigated further. One of these topics is Knowledge Transfer. There are numerous datasets for trajectory prediction, as described in Section 3.1, nonetheless, the total quantity of data is still limited. Thus, to address this issue would help being able to transfer knowledge from the training to the test set meaning to take advantage of the interplay between the functional properties of the scene and the prior knowledge of moving agents to forecast plausible paths from a new video scene (test set). The purpose of this thesis is to find a technique in order to obtain knowledge transfer that could be applied to get better results when dealing with transfer learning.

1.4 Methodology

To correctly analyze the results from different experiments the metrics used must be adequate and must be the same on all the experiments.
The metrics chosen are the Average Displacement Error (ADE) and the Final Displacement Error (FDE) explained in detail in Section 3.2. The chosen dataset are the SDD [24], the ETH and UCY [23, 18], datasets which are described in detail in section 3.1. We follow the well-established experimental protocol used

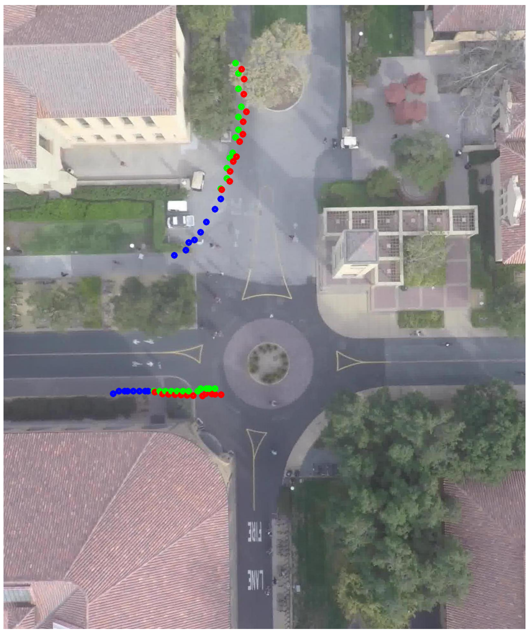

Figure 1.1: Example of trajectories on deathcircle scene of SDD dataset
Blue: observed agent positions.
Red: predicted future agent positions.
Green: real future agent positions.

in human trajectory prediction [1, 13], that is to observe 3.2 s and to predict the next 4.8 s.
This means that we consider $T_{obs} = 8$ and $T_{pred} = 12$ time steps, respectively. The results reported will be related to the models presented trained and tested on SDD and ETH/UCY datsets and then trained on SDD and tested on ETH/UCY (applying transfer learning). The baseline architectures used for this research are the SAR and the Goal-SAR [7] presented in detail in Chapter 4.1.

- SAR: recurrent backbone based on a multi-head attention mechanism.
- Goal-SAR: SAR with a scene-aware goal-estimation module.

An example of trajectories is reported in figure 1.1.:

– Blue: observed agent positions $T_{obs} = 8$.

– Red: predicted future agent positions $T_{pred} = 12$.

– Green: real future agent positions.

2 Related Works

This chapter is divided into two sections.
In the first section we reported the state of the art techniques over years used to solve the general trajectory prediction problem.
In the second section we focus more on the papers related to knowledge transfer applied for trajectory prediction problem in order to make an overview of the ideas from which we take inspiration from.

2.1 Trajectory Prediction

There have been different methods to solve the human agents trajectory prediction problem over time.
They evolved starting from physics-based models to data-driven models that use deep learning.
In this section I will analyze some state of the art architectures used to solve trajectory prediction problem over the years.

2.1.1 Social Forces

Helbing and Molnar in 1995 in [14] proposed a method that represents one of the most historically influential works.
The architecture considers each pedestrian as a particle and assumes that the motion of these particles can be described with three forces representing:

1. The acceleration towards the preferred velocity for each pedestrian;

2. The sum of all the repulsion forces: these forces simulate the desire to keep a certain distance from other people and obstacles;
3. The sum of all the attractive forces: these forces simulate the desire of pedestrians to stay close to other people (friends and couples walk closely together).

The sum of all the forces affecting a pedestrian will represent its final acceleration.
The social forces model takes a physics-based approach to trajectory prediction, it has a clear formalization but needs precise information on the obstacles and positions of other pedestrians, therefore can be applied only in scenarios where this data is available.

2.1.2 Social LSTM

Alahi et al. [1] uses a model called Social LSTM (Fig 2.1) which can account for the behavior of other people within a large neighborhood, while predicting a persons path.
They use a separate LSTM network for each trajectory in a scene.
The LSTMs are then connected to each other through a Social pooling (S-pooling) layer. Unlike the traditional LSTM, this pooling layer allows spatially proximal LSTMs to share information with each other. The hidden-states of all LSTMs within a certain radius are pooled together and used as an input at the next time-step.

2.1.3 MUSE-VAE

Mihee Lee at al. [17] proposed an architecture composed by 2 stages:

1. **Macro-stage (Coarse Prediction Stage):** this macro stage is composed by 2 different models
   - LG-CVAE: this is the long-term goal prediction model.
     The input consists of concatenated local semantic map and past trajectory heatmap while the output is a one long-term goal heatmap.
   - SG-net: this is the short-term goal prediction model.
     The input consists of concatenated local semantic map, past trajectory heatmap and long-term goal heatmap while in output it gives $NSG+1$ heatmaps, where NSG is the number of short-term goals.

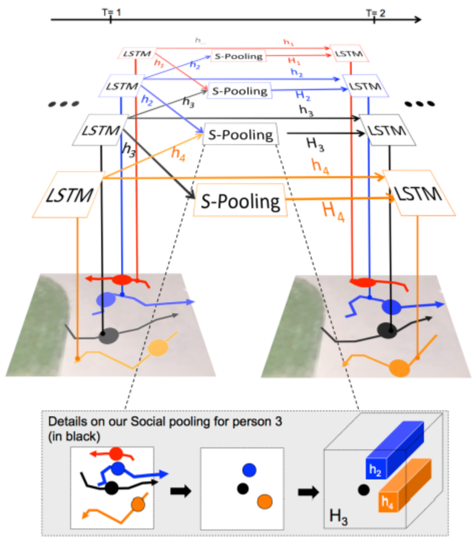

Figure 2.1: Overview of Social-LSTM method

2. **Micro-stage (Fine Prediction Stage):** they use this final stage to predict complete future trajectories at the micro level.
   To deal with uncertainty they leverage CVAE in this step as well.
   While decoding future steps, their model use the long-term and short-term goal information from SG-net in the form of LSTM-encoded feature and they apply the Teacher Forcing technique to correct the prediction by feeding the GT/predicted long-term and short-term goals during training/test time respectively.
   They also feed the U-net features from LG-CVAE to the prior network of micro-stage so that the Micro-stage also recognizes the environment.

The MUSE-VAE architecture is represented in the figure 2.2

2.1.4 GAN

A generative adversarial network (GAN), designed by Ian Goodfellow and his colleagues in [12], consists of two adversarially trained models:

- **Generative model G**: captures the data distribution.
  It takes a latent variable z as input, and outputs sample $G(z)$.

- **Discriminative model D**: estimates the probability that a sample came from the training data rather than G.
  It takes a sample x as input and outputs D(x) which represents the probability that it is real.

The training procedure is similar to a two-player min-max game with the

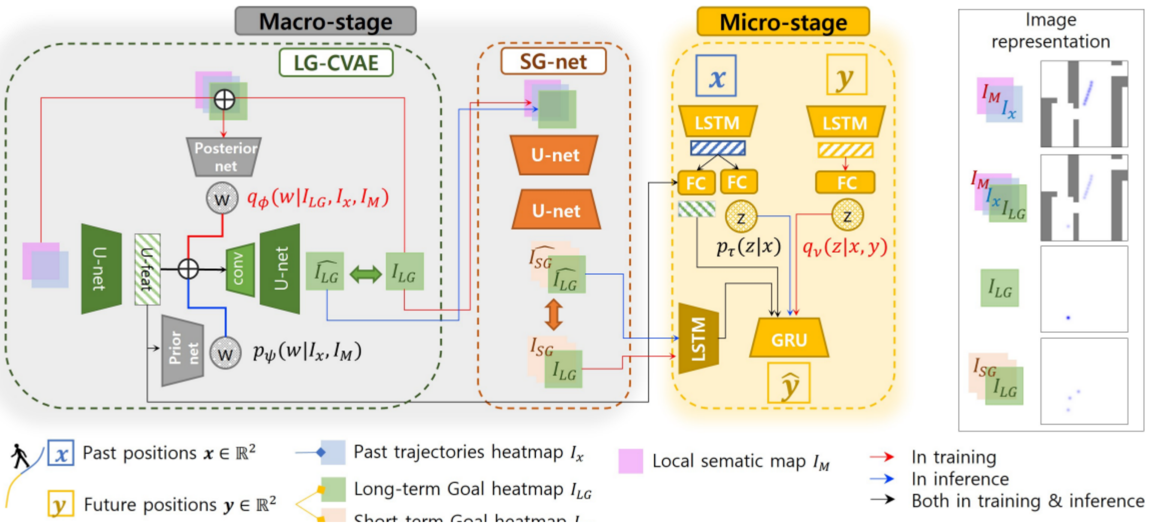

Figure 2.2: Overview of MUSE-VAE architecture

following objective function:

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

Goal GAN

Patrick Dendorfer et al. [10] proposed an architecture composed of three key components

- Motion Encoder (ME): extracts the pedestrians dynamic features recursively with a long short-term memory (LSTM) unit capturing the speed and direction of motion of the past trajectory.
- Goal Module (GM): combines visual scene information and dynamic pedestrian features to predict the goal position for a given pedestrian. This module estimates the probability distribution over possible goal (target) positions, which is in turn used to sample goal positions.
- Routing Module (RM): generates the trajectory to the goal position sampled from the GM. While the goal position of the prediction is determined by the GM, the RM generates feasible paths to the predetermined goal and reacts to obstacles along the way by using visual attention.

The Goal-GAN architecture is represented in the figure 2.3

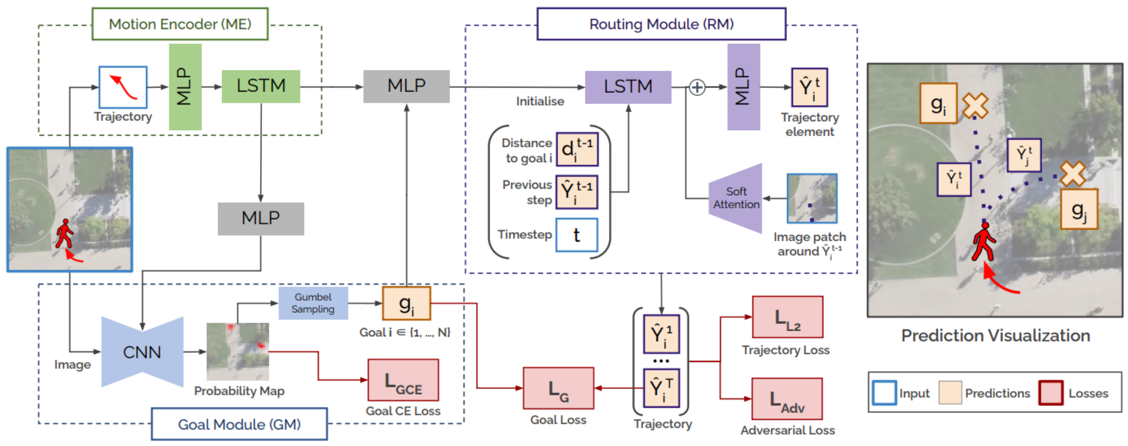

Figure 2.3: Overview of Goal GAN architecture

Social GAN

Agrim Gupta et. al [13] proposed a Socially-Aware GAN leveraging on the fact that generative models can be used with time series data to simulate possible futures.
Their model (Fig 2.4) is composed by three components:

- **Generator G**: Starting from the positions of the people inside the scene they create their fixed size embeddings e_i^t using a single layer MLP. Then they feed this embeddings to the LSTM cell of the encoder at time t:

$$e_i^t = \phi(x_i^t, y_i^t; W_{ee})$$
$$h_{ei}^t = LSTM(h_{ei}^{t-1}, e_i^t; W_{encoder})$$

  where $\phi(\cdot)$ is an embedding function, W_{ee} is the embedding weight and $W_{encoder}$ are LSTM wights which are shared between people in a scene. Encoder learns the state of a person and stores their history of motion and they model the human-human interaction via a Pooling Module (PM). After t_{obs} they pool hidden states of all the people present in the scene to get a pooled tensor P_i for each person. To get a future scenario that is consistent with the past they condition the generation of output trajectories by initializing the hidden state of the decoder such as:

$$c_i^t = \gamma(P_i, h_{ei}^t; W_c)$$
$$h_{di}^t = [c_i^t, z]$$

  where $\gamma(\cdot)$ is an MLP and W_c the embedding weight.

- **Pooling Module PM**: They passed the input coordinates through a MLP followed by a symmetric function using max pooling. The resulting vector P_i needs to summarize all the information a person needs to make a decision.

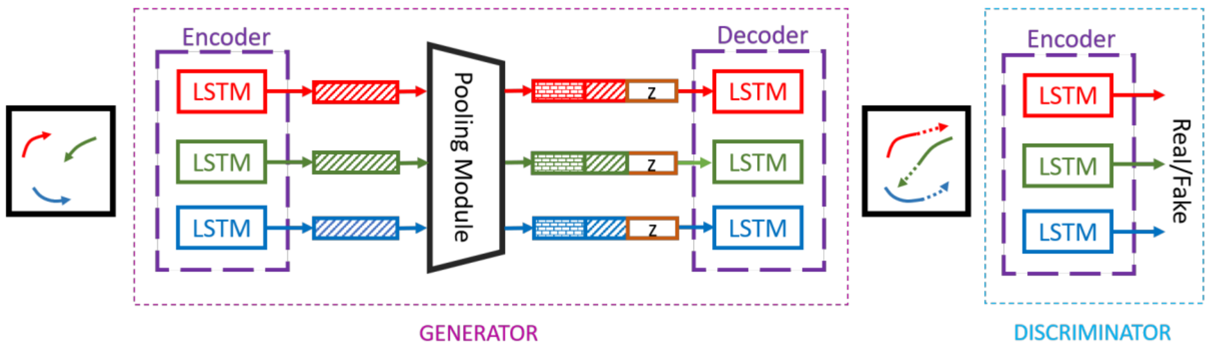

Figure 2.4: Overview of Social GAN architecture

- **Discriminator D**: It's a separate encoder that takes as input $T_{real} = [X_i, Y_i]$ or $T_{fake} = [X_i, \hat{Y}_i]$ and classifies them as real or fake. Moreover they used an MLP at the end of the encoder result to get a classification score.
  The goal of the discriminator is to learn social interaction rules and find which trajectories are fake (meaning not socially acceptable).

MG-GAN

Patrick Dendorfer et. al. [9] proposed a multi-generator framework (shown in Fig 2.5) for pedestrian trajectory prediction.
The model learns a discontinuous function as a mixture of distributions modeled by multiple generators.
Main parts of the model are:

- **Trajectory and Visual Encoders**: feature encoders extract visual and dynamic features d_i from the input sequences X_i and scene image patches I_i of each pedestrian i.
  The attention modules compute the physical attention [27] features v_i and social attention [2] features s_i.
  Then they concatenate the features obtained in $c_i = [d_i, v_i, s_i]$.

- **Multi-generator Model**: they used n_G different generators G_g which are LSTM decoders initialized with the features c and a random noise vector $z \sim N(0,1)$ as the initial hidden state h^0. Each generator specializes in learning a different trajectory distribution conditioned on the input c. The final trajectory $\hat{Y}$ is then predicted recurrently as:

$$\Delta\hat{Y}^t = LSTM_g(\Delta X^{t-1}, h^{t-1})$$

  They train a module that adapts to the scene by activating specific generators, conditioned on the observations and interactions c.

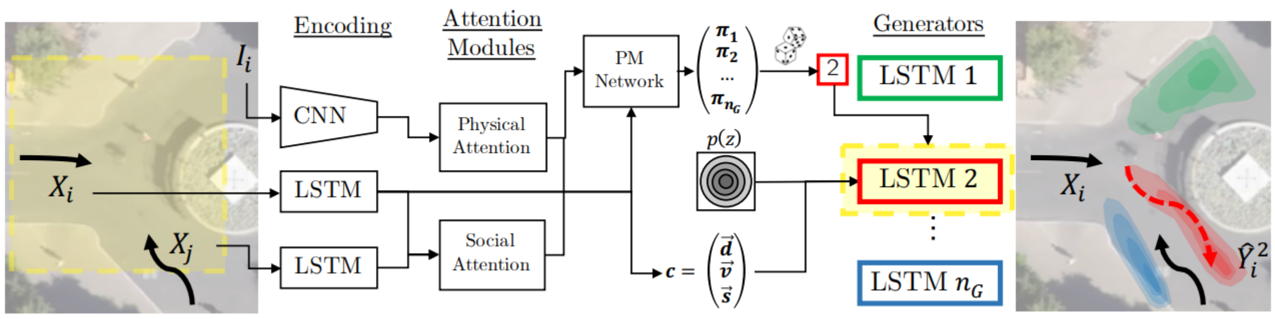

Figure 2.5: Overview of MG-GAN architecture

2.1.5 PECNet

Karttikeya Mangalam et. al. in [22] proposed a Predicted Endpoint Conditioned Network for flexible human trajectory prediction (Fig 2.6).
The network is composed by different modules:

- **Past Trajectory Encoder**: for all pedestrian p_k in the scene they extract the previous history T_i^k and the ground truth endpoint G^k. They use an encoder E_{past} to encode past trajectory T_i^k for all p_k independently.
- **Endpoint Encoder**: future endpoint G^k is encoded with and Endpoint encoder E_{end} to produce $E_{end}(G^k)$ independently for all k.
- **Latent Encoder**: both past and future destination representations coming from the aforementioned encoders are concatenated together and passed into the latent encoder E_{latent} which produces parameters (μ, σ) for the encoding latent variable $z = N(\mu, \sigma)$ of the VAE.
- **Endpoint VAE**: using the Endpoint VAE they infer a distribution on G (sub-goal endpoint) based on the previous location history T_i of p^k.

2.1.6 Y-net

Karttikeya Mangalam et. al. in [21] proposed an architecture called Y-net (Fig 2.7).
The RGB image I is processed with U-net [25] in order to produce a semantic segmentation map S with N_c classes. In parallel the past motion history $\{u_n\}_{n=1}^{n-p}$ of an agent p is converted to a trajectory heatmap H of spatial sizes of I and n_p

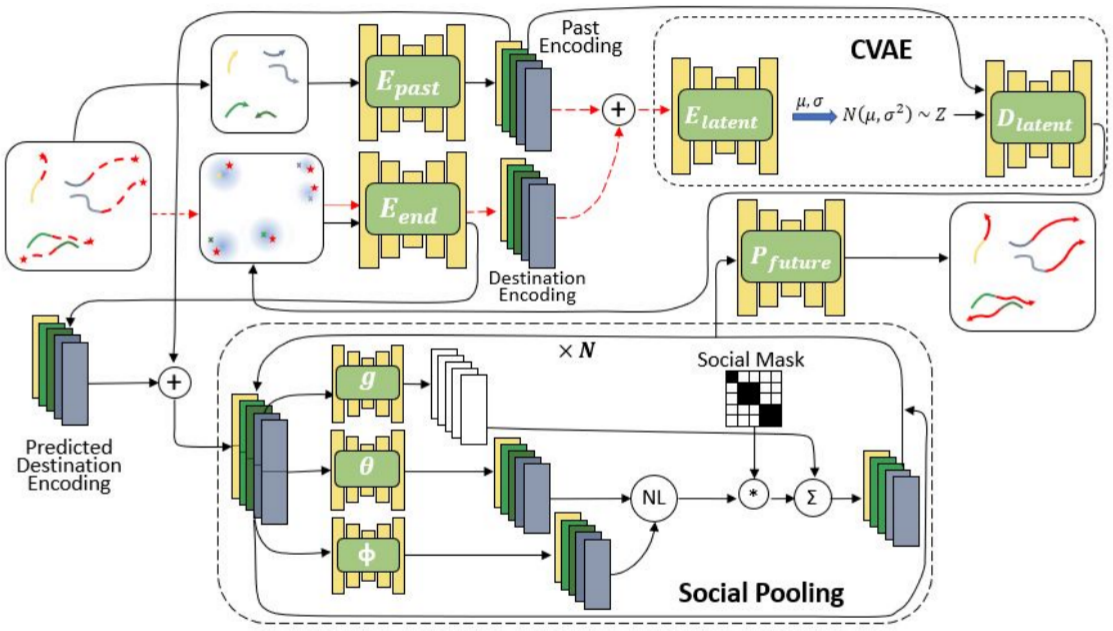

Figure 2.6: Overview of PECNet architecture

channels corresponding to the past t_p seconds.

$$H(n,i,j) = 2\frac{||(i,j) - u_n||}{\max_{(x,y)\in I} ||(x,y) - u_n||}$$

Then they concatenate the trajectory representation with the semantic map S producing the heatmap tensor H_S.
H_s is then processed by the U_e encoder (designed as a U-net encoder [25]) in order to obtain a compact and deep representation H_{U_e} that is passed onto the U_g goal decoder and the U_t trajectory decoder.
The U_g goal and waypoint heatmap decoder uses bilinear up-sampling and convolution to expand the feature map, moreover intermediate representations from U_e are merged after every deconvolution to not limit the final resolution of the goal heatmap.
The estimated goal and waypoint distributions are sampled and the obtained samples are converted to a heatmap representation. The obtained conditioning tensor H_{U_g} is spatially downsampled to match the corresponding blocks size and they are passed along with the past motion and scene representation H_{U_e} to the trajectory heatmap decoder network U_t.
They achieved multimodality in paths through estimated probability distribu-

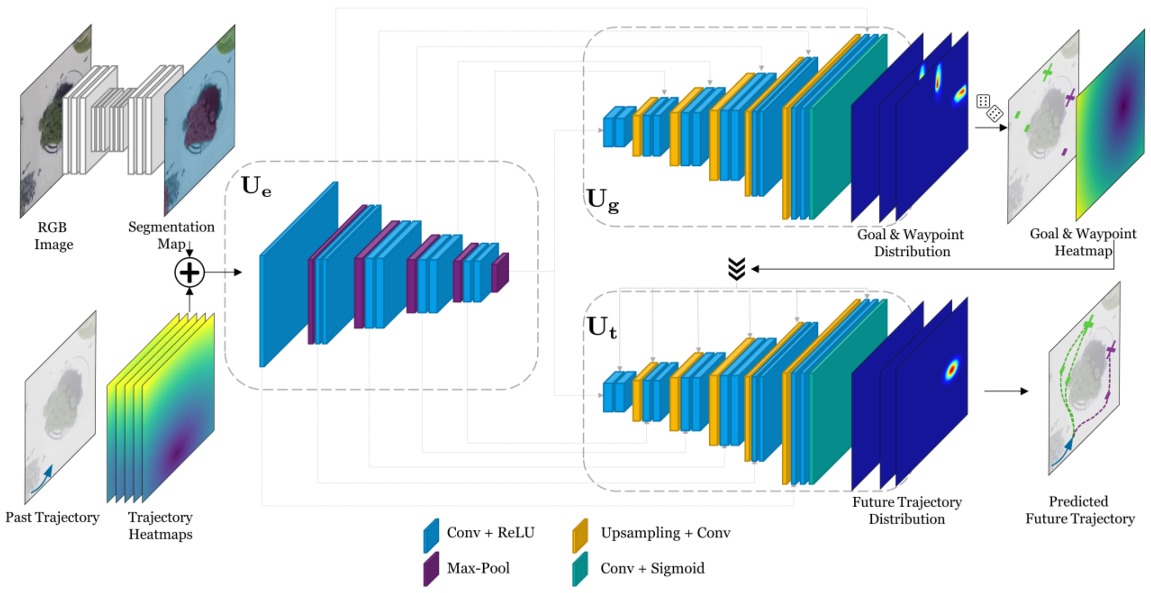

Figure 2.7: Overview of Y-net architecture

tions obtained by U_t conditioned on samples from U_g for predicting diverse multimodal scene-compliant futures.

2.1.7 Trajectron++

Tim Salzmann et. al. designed in [28] the Trajectron++, a modular, graph-structured recurrent model that forecasts the trajectories of diverse agents (Fig 2.8).

To understand the proposed framework idea we have to highlight different concepts:

- **Scene Representation**: starting from the scene they abstracted a spatio-temporal graph $G = (V, E)$ where:
  - Nodes: represent agents and they have also a semantic class representing the type of the agent (e.g. Pedestrian, bus, car)
  - Edges: represent interactions between agents meaning that an edge (A_i, A_j) is present in E if A_i influences A_j.

- **Modeling Agent History**: once the Scene representation phase is done the model needs to encode a node's current state, its history and how it is

influenced by its neighboring nodes.
To encode the observed history of the modeled agent: they fed current and previous D-dimensional states $x = s^{t-H,t}_{1,\dots,N(t)} \in \Re^{(H+1)\times N(t)\times D}$ into a Long Short-Term Memory (LSTM) network [15] with 32 hidden dimensions.

- **Encoding Agent Interactions**: to model neighboring agents influence the proposed network encodes graph edges in two steps:

  1. Edge information is aggregated using element-wise sum from neighboring agents of the same semantic class.
  2. Fed the previous computed aggregated states into an LSTM with 8 hidden dimension whose weights are shared across all edge instances of the same type.

  Then the encodings from all edge types that connect to the modeled node are aggregated to obtain one influence representation vector, representing the effect that all neighboring nodes have. For this, an additive attention module is used [3]. Finally, the node history and edge influence encodings are concatenated to produce a single node representation vector, e_x.

- **Incorporating Heterogeneous Data**: It also possible to include further additional information (e.g., raw LIDAR data, camera images, pedestrian skeleton or gaze direction estimates) in the proposed framework by encoding it as a vector and adding it to this backbone of representation vectors, e_x. In order to do that Trajectron++ encodes a local map, rotated to match the agents heading, with a Convolutional Neural Network (CNN) which output is concatenated with the node history and edge influence representation vectors.

- **Encoding Future Ego-Agent Motion Plans**: Trajectron++ produce predictions which take into account future ego-agent motion in order to evaluate a set of motion primitives with respect to possible responses from other agents.
  In order to do that the proposed framework is able to encode the future T timesteps of the ego-agents motion plan y_R using a bi-directional LSTM due to its strong performance on other sequence summarization tasks [6]. The final hidden states are then concatenated into the backbone of representation vectors, e_x.

- **Explicitly Accounting for Multimodality**: Trajectron++ explicitly handles multimodality by leveraging the CVAE latent variable framework [29].
  It produces the target $p(y|x)$ distribution by introducing a discrete Categorical latent variable $z \in Z$ which encodes high-level latent behavior and allows for $p(y|x)$ to be expressed as $p(y|x) = \sum_{z\in Z} p_\psi(y|x,z)p_\theta(z|x)$ where $|Z| = 25$ and ψ, θ are deep neural network weights that parameterize their respective distribution.

- **Producing Dynamically-Feasible Trajectories**: After obtaining a latent variable z, it and the backbone representation vector e_x are fed into the decoder, a 128-dimensional Gated Recurrent Unit (GRU) [8]. Each GRU cell

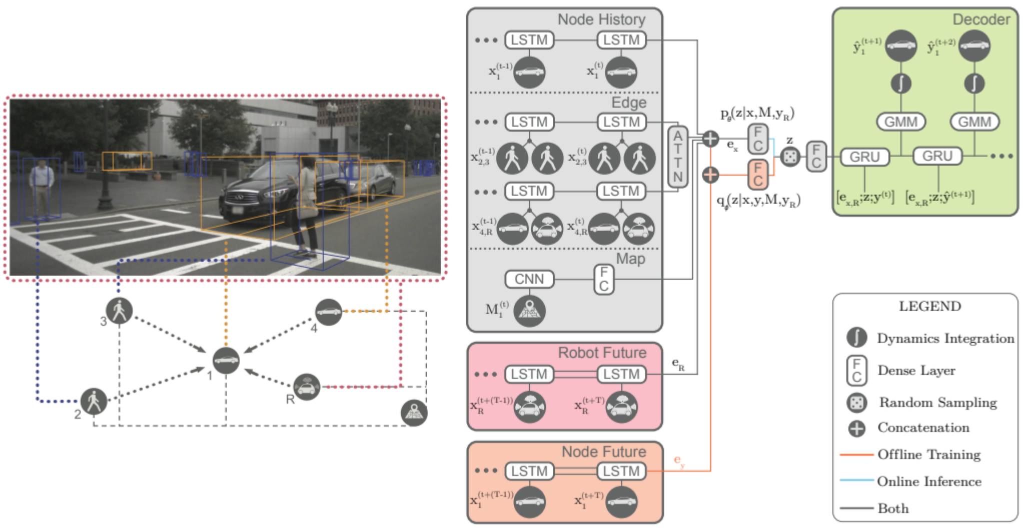

Figure 2.8: Overview of Trajectron++ architecture

outputs the parameters of a bivariate Gaussian distribution over control actions $u^{(t)}$ (e.g., acceleration and steering rate). The agents system dynamics are then integrated with the produced control actions $u^{(t)}$ to obtain trajectories in position space [5]. Their approach is uniquely able to guarantee that its trajectory samples are dynamically feasible by integrating an agents dynamics with the predicted controls.

- **Training the Model**: they adopt the InfoVAE [32] objective function, and modify it to use discrete latent states in a conditional formulation (since the model uses a CVAE).

2.1.8 NSP-SFM

Jiangbei Yue et. al. in [31] proposed a new model called Neural Social Physics or NSP which is a deep neural network within which they use an explicit physics model with learnable parameters.
This architecture is the first one in the global rank for trajectory prediction results (ADE world and FDE world) on ETH/UCY with an ADE world of 0,17 and an FDE of 0,24.
The architecture proposed is the Neural Social Physics - Social Force Models (NSP-SFM) shown in figure 2.9
They design the NSP-SFM by assuming each person acts as a particle in a particle system and each particle is governed by Newtons second law of motion.

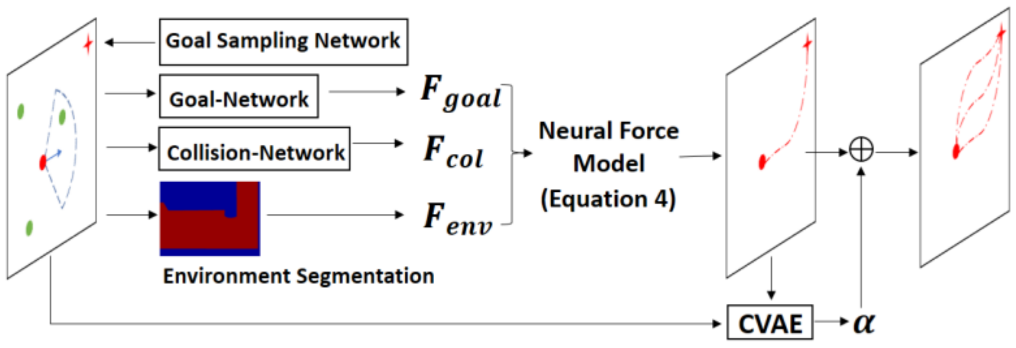

Figure 2.9: Overview of NPS-SFM architecture

Considering discrete observation up to time T ($t \in \{0, 1, ..., T\}$) the rate of change of linear momentum of a body $\ddot{p}(t)$ is designed to be dependent on three forces:

- $\mathbf{F_{goal}}$ goal attraction
- $\mathbf{F_{col}}$ inter-agent repulsion
- $\mathbf{F_{env}}$ environment repulsion

obtaining the equation:

$$\ddot{p}(t) = F_{goal}(t, q^T, q^t) + F_{col}(t, q^t, \omega^t) + F_{env}(t, q^t, E)$$

where:

- E: environment and it will be explained later.
- p : position
- $\dot{p}$: velocity
- q^t: state of a person at time t $[p^t, \dot{p}^t]^T$
- ω^t: neighborhood pedestrians states at time t.

Unlike social force model [14], the three forces are partially realized by neural networks, turning the equation into a neural differential equation.
They employ:

- Goal Sampling Network (GSN): network to sample p^T during prediction. It's similar to a part of Y-net [21]

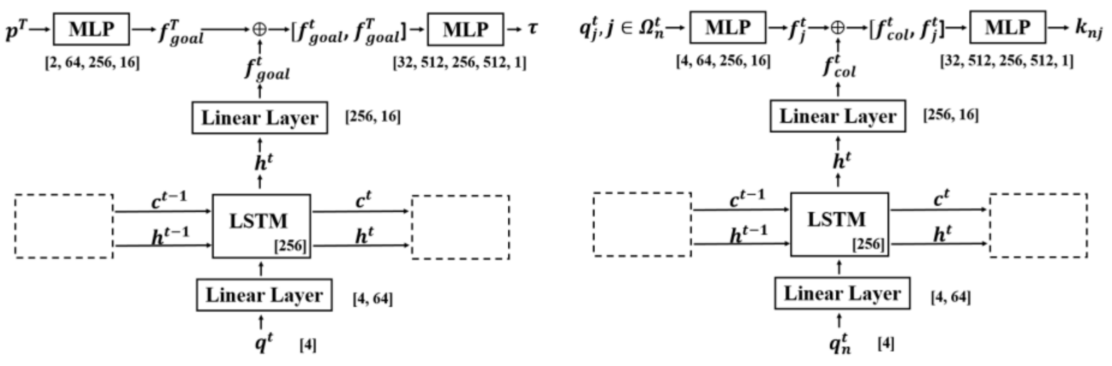

Figure 2.10: Overview of Goal-Network (Left) and Collision-Network (Right).

- Goal Network ($NN_{\phi 1}$): given the current state and the goal it computes F_{goal}.
  This network encodes q^t and then feeds it into a Long Short Term Memory (LSTM) network to capture dynamics.
  After a linear transformation, the $LSTM$ output is concatenated with the embedded p^T and finally the output is computed by an MLP (multi-layer perceptron). An overview of the model is reported in figure 2.10 (model on the left).

- Collision Network ($NN_{\phi 2}$): given the current state and the neighborhood pedestrians states it computes F_{col}.
  The network is similar to the Goal Network explained above: it encodes the agent state q^t_n and concatenate it to the encodes of every agent state in the neighborhood $q^t_j \in \omega^t_n$. An overview of the model is reported in figure 2.10 (model on the right).

- Environment Repulsion: they modeled F_{env}, that represents the repulsion from the environment given the current state, using the formula:

$$F_{env} = \frac{k_{env}}{||p^t_n - p_{obs}||} \left(\frac{p^t_n - p_{obs}}{||p^t_n - p_{obs}||} \right)$$

  where p_{obs} is the position of the obstacle and k_{env} is a learnable parameter that NSP-SFM learns via back-propagation and stochastic gradient descent.

- Conditional Variational Autoencoder (CVAE): in order implement the dynamics stochasticity concept they use a CVAE [29].
  An overview of the model is reported in figure 2.11.

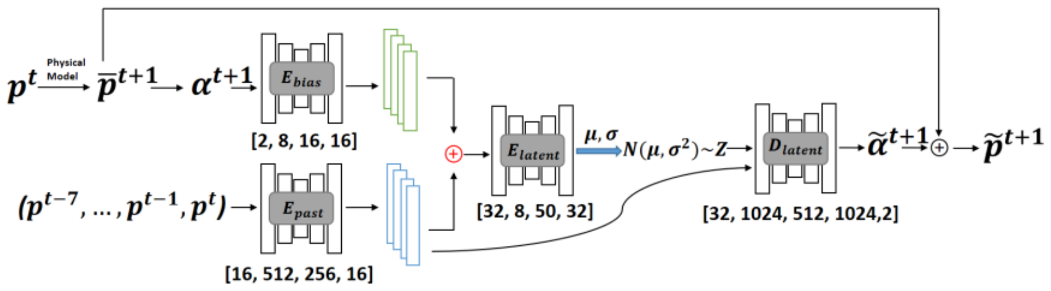

Figure 2.11: Overview CVAE architecture of NSP-SFM
$\tilde{p}^{t+1}$ is the intermediate prediction out of our force model and $\alpha^{t+1} = p^{t+1} - \tilde{p}^{t+1}$.
Encoders E_{bias} E_{past} E_{latent} and decoder D_{latent} are MLP networks.

2.2 Knowledge Transfer

Let's now focus on transfer learning concept and in this section we are going to describe different ideas formulated over time in order to be able to transfer knowledge when we are dealing with trajectory prediction problems.

2.2.1 Synthetic Dataset

This idea comes from the Patrick Dendorfer et al. [10], the same paper in which the aforementioned Goal-GAN architecture has been proposed.
They create a small synthetic dataset by generating trajectories using the Social Force Model [14] in the hyang 4 scene of the SDD dataset.
To ensure the feasibility of the generated trajectories, they use a two-class manually labeled semantic map, that distinguishes between feasible (eg. walking paths) from unfeasible (eg. grass) areas.
They simulate 250 trajectories approaching and passing the two crossroads in the scene.

2.2.2 Local Descriptor

Lamberto Ballan et al. [4] proposed a Knowledge Transfer technique applied for human trajectory prediction.
The main idea is that the elements of the scene define a semantic context, and they might determine similar behaviours in scenes characterized by a similar context. For the aforementioned reason they compute three steps:

1. **Scene Parsing**: they labeled the scene taking inspiration from the scene parsing method proposed in [30]. For each image they extract several local and global features (SIFT + LLC encoding, GIST and color histograms). The algorithm first retrieves a set of image neighbors from the training set, then they used superpixel classification and MRF inference to refine the labeling.

2. **Semantic context descriptors**: The final patch descriptor p_i is a weighted concatenation of the global (g_i) and local (l_i) semantic context: $\mathbf{p_i} = w\mathbf{g_i} + (1w)\mathbf{l_i}$ where:

   - Global context descriptor g_i: is a C-dimensional vector, where C is the number of labels in the ground-truth. This is obtained by computing the Euclidean distance between the centroid of the patch and the closest point in the full image labeled as c, for each $c \in C$ (e.g., c can be the class road). The role of the global context descriptor is to account for the relative distance between each patch, and all the other semantic elements of the scene.
   - Local context descriptor l_i: they partitioned the space surrounding the current patch i into concentric shells, considering patches at distance 0, 1 and 2, in the grid.
     For each patch, the local histograms are formed by counting the number of pixels labeled as c in that shell. These histograms are then averaged, providing the final l_i.

3. **Descriptor matching**: For each patch descriptor pi in the query image, they rank all patches from training images using $L2$ distance and keep the set N_i of K nearest-neighbors.
   Then they compute the average of previously collected information (eg. popularity scores) among the neighbors in N_i for each patch i and they transfer this information to that particular patch i.

3 Datasets and Evaluation Metrics

3.1 Datasets

More specifically, in pedestrian trajectory prediction the data available can be in two different formats: in image coordinates or real-world coordinates.
Image coordinates means that each pedestrian is represented with the pixels it occupies in the camera image, while real-world coordinates means that each pedestrian is represented by its position in meters with origin in an arbitrary point of the world. The datasets used in this thesis are the ETH/UCY and SDD because they are widely used in literature and publicly available.

3.1.1 SDD

The Stanford Aerial Pedestrian Dataset [24]: Consists of annotated videos of pedestrians, bikers, skateboarders, cars, buses, and golf carts navigating eight unique scenes on the Stanford University campus.
The eight main scenes are divided in a total of 47 scenes:

- bookstore: 4 scenes
- coupa: 3 scenes
- deathCircle: 5 scenes
- gates: 9 scenes
- hyang: 10 scenes
- little: 4 scenes

- nexus: 10 scenes
- quad: 4 scenes

Each video for each scene in the videos directory has an associated annotation file (annotation.txt) and exemplary frame (reference.jpg) in the annotations directory.

Annotation file format

Each line in the annotations.txt file corresponds to an annotation. Each line contains 10+ columns, separated by spaces. The definition of these columns are:

1. Track ID. All rows with the same ID belong to the same path.
2. xmin. The top left x-coordinate of the bounding box.
3. ymin. The top left y-coordinate of the bounding box.
4. xmax. The bottom right x-coordinate of the bounding box.
5. ymax. The bottom right y-coordinate of the bounding box.
6. frame. The frame that this annotation represents.
7. lost. If 1, the annotation is outside of the view screen.
8. occluded. If 1, the annotation is occluded.
9. generated. If 1, the annotation was automatically interpolated.
10. label. The label for this annotation, enclosed in quotation marks.

For our experiments we split SDD dataset into 60 recordings where complex human dynamics show strong interactions with the surrounding environment. We use the same split proposed in Kothari et al. [16] and used by most recent works [22, 26], where 30 scenes are used as train and 17 as test data, and only pedestrian are retained. Moreover data is down-sampled at 2.5 FPS.

3.1.2 ETH and UCY

The ETH (name taken from the ETH Zurich University) dataset [23] contains two scenes (named Eth and Hotel) taken from a birds eye view. In total it comprises of 750 different pedestrians trajectories. One frame is annotated with pedestrian positions every 0.4 seconds. The UCY (name taken from the University of Cyprus) dataset [18] contains three scenes (Zara1, Zara2 and Univ),

taken from a birds eye view. In total it comprises of more than 900 different pedestrians trajectories. One frame is annotated with pedestrian positions every 0.4 seconds.
The two datasets are usually used together and they are sampled at 2.5 FPS and contain five different scenes (ETH, HOTEL, UNIV, ZARA1 and ZARA2) monitoring entrances of buildings and sidewalks from RGB cameras typically used in video surveillance applications.
In total we considered 1536 pedestrians mainly showing human-human interactions. The training and testing are done with the leave-one-out approach [13]: the model is trained on four scenes and tested on the fifth, and this procedure is repeated five times, one for each scene. Since these two datasets are mainly used in combinations with each other from now on the two datasets together will be referred to as the ETH/UCY dataset.

3.2 Metrics

In data-driven approaches, there is the need to standardize test settings and commonly used metrics to have quantitative results.
These quantitative results permit scientists to understand how different models compare with each other, and what are their strength and their weaknesses.
for the aforementioned reasons the metrics used in this thesis are ADE and FDE.

3.2.1 Average Displacement Error (ADE)

The Average Displacement Error (ADE), which was introduced in [23], represents the error over all the predicted points and the ground truth points from T_{obs+1} to T_{pred} averaged over all pedestrians. This error is calculated using the Euclidean distance between the predicted position and the real pedestrian position for each time steps. The formula of the ADE is:

$$ADE = \frac{\sum_{i=1}^{n} \sum_{t=T_{obs}}^{T_{pred-1}} ||\hat{Y}_t^i - Y_t^i||}{n(T_{pred} - T_{obs})}$$

where n represent the number of pedestrians, $\hat{Y}_t^i$ are the predicted coordinates for pedestrian i at time t, Y_t^i is the real future positions and $||$ represents the Euclidean distance. The ADE for ETH/UCY dataset is a measure in meters

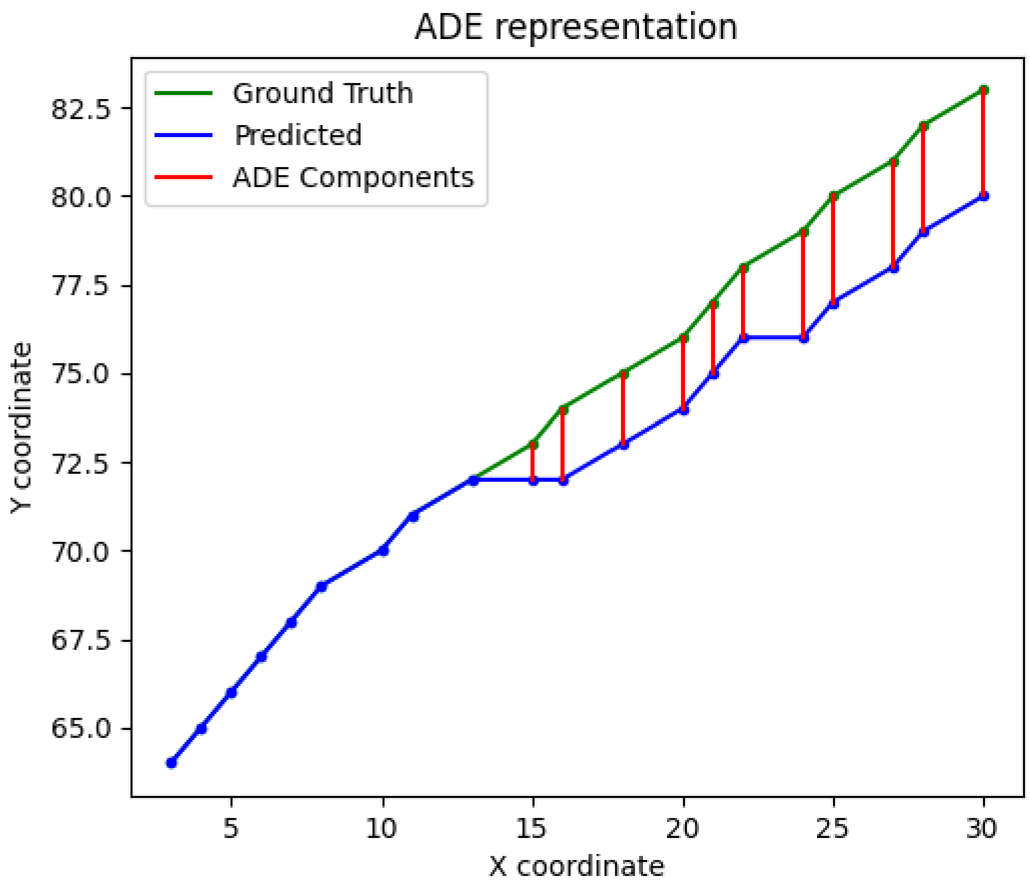

Figure 3.1: Average Displacement Error (ADE) Representation.
Its value is the length of the red line on average.

since it has real-world coordinates, while for SDD is a measure in pixels since it has pixels coordinates.
To resume, the ADE (Fig 3.1) is the average distance from every position of the prediction to every corresponding position of the real trajectory.

3.2.2 Final Displacement Error (FDE)

The second metric that is used in pedestrian trajectory prediction is the Final Displacement Error (FDE), which was also introduced in [23]. The Final Displacement Error is the error between the final predicted position at T_{pred} and the real position at T_{pred}. The formula of the FDE is:

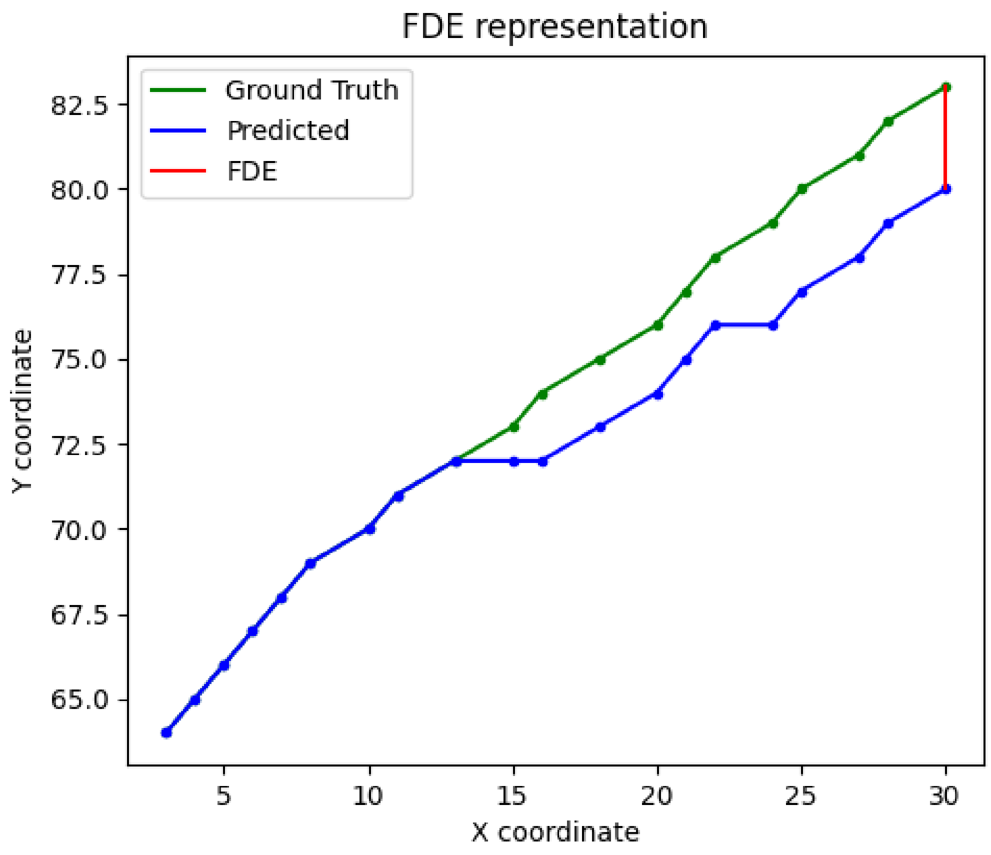

Figure 3.2: Final Displacement Error (FDE) Representation.
Its value is the length of the red line

$$FDE = \frac{\sum_{i=1}^{n} ||\hat{Y}^{i}_{T_{pred-1}} - Y^{i}_{T_{pred-1}}||}{n}$$

where n represent the number of pedestrians, $\hat{Y}^i_t$ are the predicted coordinates for pedestrian i at time t, Y^i_t is the real future positions and $||$ represents the Euclidean distance. The FDE for ETH/UCY dataset is a measure in meters since it has real-world coordinates, while for SDD is a measure in pixels. In essence, the FDE (Fig 3.2) is how big is the distance error between the last predicted position and the last real position.

On our experiments since we frame our analysis in a stochastic setting, we report $\min_{20} ADE$ and $\min_{20} FDE$ metrics, which are obtained by generating 20 predicted samples for each input trajectory and retaining the one that provides the smallest errors.

4 Experiments

4.1 Idea

Taking inspiration from [4] the elements of the scene define a semantic context, and scenes characterized by a similar context could determine similar behaviors of the trajectories of human agents. The idea of this thesis is to implement a parallel module to add to the baseline architecture (Goal-SAR 4.1) in order to extract a local descriptor looking at a semantic patch around each position of each agent.

4.1.1 Baseline

As baselines two recurrent networks have been used: SAR and Goal-SAR presented in [7].

SAR

The self-attentive recurrent backbone (module number 1 in Fig. 4.1) is only based on temporal information and processes sequences of 2D locations $p_i^t = (x_i^t, y_i^t)$ for the agent i^{th} at time t. Sequences span from $t = 0$ to $t = T_{obs} + T_{pred} = T_{seq}$, where:

- T_{obs}: the observation time window.
- T_{pred}: the prediction time window.
- T_{seq}: the total sequence length.

The model is composed of three different parts:

- **Trajectory Embedding**: every agent is considered independently from others and input coordinates $P_i^{0:t_n} = (p_i^0, ..., p_i^{t_n})$ from $t = 0$ to t_n are encoded in a feature space as follows:
  $$e_i^t = \phi(p_i^t; W_e)$$
  where $\phi()$ is a linear embedding function with ReLU nonlinearity and W_e are embedding weights.

- **Temporal Attention**: they leverage on the encoding part of a transformer with the aim to predict future positions given variable length input sequences in a recurrent fashion.
  Temporal dependencies across subsequent time steps are taken into account by linearly projecting embedded positions $(e_i^0, ..., e_i^{t_n})$ into three different vectors:

  - q_i^t: query.
  - k_i^t: key.
  - v_i^t: value.

  A dot product between queries and values is performed to compute the attention coefficients used to weight the values v and provide the corresponding output as follows:
  $$ATT(Q_i, K_i, V_i) = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$
  where d_k represents a normalization factor.
  This operation is performed N_{head} times using different linear projections of Q_i, K_i and V_i, yielding a vector of new embedded positions $(h_i^0, ..., h_i^{t_n})$, which incorporate temporal dependencies.

- **Trajectory decoder**: The last encoded feature vector $h_i^{t_n}$ is then fed to a decoder defined by a linear layer $\psi()$ with $ReLU$ nonlinearity and weights W_d, to extract the decoded positions in time $t_n + 1$. To increase the variance of the generated sequences, they concatenate a Gaussian random vector $z_i \sim N(0, I_z)$ into this hidden state.
  $$\hat{p}_i^{t_n+1} = \psi(concat(h_i^{t_n}, z_i); W_d)$$
  Instead of using a hidden state, they recursively concatenate estimated locations into the previous input sequence.

Goal-SAR

Goal-SAR architecture (Fig 4.1) is composed by the aforementioned Self-Attentive Recurrent backbone plus a Goal Module.
The goal module purpose is to model the multimodality of human motion, and this is achieved by predicting a probability distribution of plausible final positions (i.e. goals) for each input trajectory.
They used the same goal module proposed in Mangalam et al. [21], modifying its preprocessing steps and output format.
This module concatenates both observed positions and visual scene information, which are then fed to a U-Net [25] model that directly outputs a probability map of future final locations. This module is characterized by different aspects:

- **Scene semantic**: obstacles may influence human dynamics or sidewalks may be the natural choice for pedestrians rather than roads for these reasons the context of the scene is an important aspect that needs to be taken into consideration to estimate more realistic paths. To this end, semantic information is extracted from birds eye view RGB images using a pre-trained semantic segmentation network taken from Mangalam et al. [21]. For the semantic scene, they used six classes denoted as C = pavement, terrain, structure, tree, road, not defined so the result is a semantic tensor $S \in \Re^{W \times H \times C}$ where W and H represent input image sizes.

- **Goal Encoder-Decoder**: The semantic tensor S is concatenated to N_{obs} distribution maps depicting past motion history. More precisely, for each observed position we consider a heat-map of spatial sizes W and H, and create a $2D$ Gaussian probability distribution map with mean p_i^t and variance $\sigma_S^2 I_2$. They denote with M the projection from $2D$ (x, y) coordinates to $W \times H$ heat-map representations. After concatenation, we obtain a $W \times H \times (C + N_{obs})$ trajectory-on-scene input tensor H_S. This tensor is then fed into a U-Net architecture consisting of L blocks that reduce the input spatial dimension $H \times W$ using double convolutional layers with ReLU non-linearity and max-pooling operations.
Each intermediate output H_l $(1 \leq l \leq L)$ is then passed through skip-connections to the decoder. In the expanding arm, L decoder blocks process H_L, doubling its resolution using bilinear up-sampling, double convolutions, and ReLU nonlinearity. Skip connections fuse H_l tensors from the contracting arm, and a final output convolutional layer followed by a pixel-wise sigmoid returns the spatial probability distribution of the final position $M(p^{T_{seq}})$.

- **Distribution Sampling**: The goal module outputs a 2D heat map that represents the probability that the monitored agent will be in a specific final location at $T_{obs} + T_{pred}$ given the information from $t = 0$ to T_{obs}. The estimated goals are sampled from these probability maps. When more than one sample is required, they find it beneficial to use the Test-Time-Sampling-Trick TTST proposed in Mangalam et al. [21], where 10, 000 goals are initially sampled and then clustered with K-means to obtain the 20 output modalities. To inject the estimated destination into their temporal backbone, they concatenate to $P_i^{0:t_n}$ the goal and the following three additional inputs, denoted as I_i:

- last position
- current distance to the estimated goal
- time step value t

They also find it beneficial to use a skip connection to feed their backbone with this additional information, which is concatenated both before and after the self-attention layer.

- **Loss Function** In order to train their proposed architecture they considered two losses:

$$L_{goal} = \frac{1}{N_p} \sum_{i=1}^{N_p} BCE\left(M(p_i^{T_{seq}}), \hat{M}(p_i^{T_{seq}})\right)$$

$$L_{traj} = \frac{1}{N_p \dot{T}_{pred}} \sum_{i=1}^{N_p} \sum_{t=T_{obs}+1}^{T_{seq}} \|p_i^t - \hat{p}_i^t\|_2^2$$

They first train the goal module to minimize a Binary Cross-Entropy loss between predicted and ground-truth probability maps obtained as 2D Gaussian distributions $N(p_i^{T_{seq}}, \sigma_S^2 I_2)$ centered at ground-truth final destinations. Second, they train their recurrent backbone minimizing the mean square error between the predicted and ground-truth positions from $T_{obs} + 1$ to $Tseq$. The two terms are then normalized with respect to the number of processed agents N_p.
Their total loss function is defined as:

$$L = L_{goal} + \lambda L_{traj}$$

where λ is an hyper-parameter that balances the contribution of each network.

4.1.2 ORB or SIFT Local Descriptor Module

One implementation has been done using the ORB [19] descriptor and another one using SIFT [20]. Each agent is considered independently from others and for every input coordinate $P_i^{0:t_n} = (p_i^0, ..., p_i^{t_n})$ from $t = 0$ to t_n we define as the keypoint the position of the agent and we compute the ORB or SIFT descriptor of the gray scale version of the input RGB image.

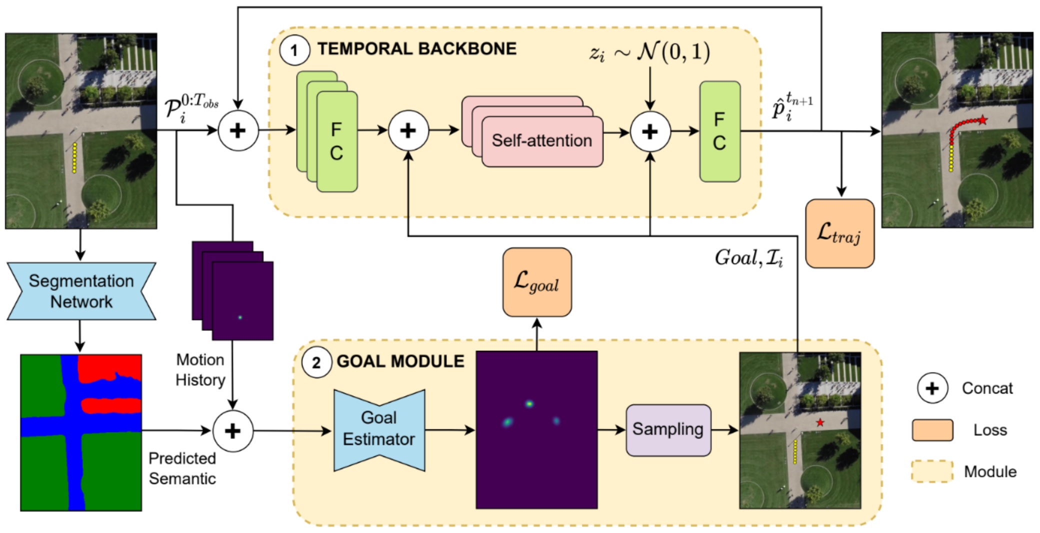

Figure 4.1: Overview of Goal-SAR architecture

SIFT

Scale Invariant Feature Transform proposed by David Lowe in [20] is composed by different steps:

- **Scale-space peak selection**: SIFT algorithm uses Difference of Gaussians (DoG) which is an approximation of LoG and is obtained as the difference of Gaussian blurring of an image with two different σ.
  This process is performed for different octaves of the image in Gaussian Pyramid.
  This procedure acts as a blob detector that detects blobs in various sizes due to changes in σ. So, we can find the local maxima across the scale and space which gives us a list of (x, y, σ) values which means there is a potential keypoint at (x, y) at σ scale.

- **Keypoint Localization**: Once potential keypoints locations are found, they have to be refined to get more accurate results. Algorithm uses Taylor series expansion of the scale space to get a more accurate location of the extrema, and if the intensity at this extrema is less than a threshold value, it is rejected. The algorithm then uses a 2×2 Hessian matrix (H) to compute the principal curvature. If the ratio of eigenvalues is greater than a threshold, that keypoint is discarded.
  After this phase the algorithm has eliminated any low-contrast keypoints and edge keypoints and what remains are strong interest points.

- **Orientation Assignment**: An orientation is assigned to each keypoint to achieve invariance to image rotation. A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with

36 bins covering 360 degrees is created.
The highest peak in the histogram is taken (with other high peaks) to calculate the orientation. The algorithm creates keypoints with same location and scale, but different directions and this contribute to stability of matching.

- **Keypoint descriptor**: A neighborhood 16×16 is taken around the keypoint. It is divided into 16 sub-blocks of size 4×4 . For each sub-block, 8 bin orientation histogram is created. So, a total of 128 bin values are available. It is represented as a vector to form a keypoint descriptor.
- **Keypoint Matching**: The keypoints between two images are matched by identifying their nearest neighbours. If the ratio of the closest distance to the second closest distance is greater than 0.8, they are rejected.

ORB

ORB stands for Oriented FAST and rotated BRIEF and it is an amazing alternative to SIFT and SURF because it is faster and has less computation cost.
ORB makes use of a modified version of the FAST keypoint detector and BRIEF descriptor. FAST features are not scale-invariant and rotation-invariant. Therefore, to make it scale-invariant ORB uses a multiscale pyramid. A multiscale pyramid consists of multiple layers where each successive layer contains a down-sampled version of the previous layer image. ORB detects features at each level/ different scales. An orientation is assigned to each keypoint (left or right) depending upon the change in intensities around that key point. Hence, ORB is also rotation invariant.

Problems

This two implementations has a big problem: the training phase of the implemented model takes more than 3 weeks, so we gave up on this idea.

4.1.3 Semantic Local Descriptor - Square Patch

Taking inspiration from [4] we decide to extract a semantic local descriptor. Given in input:

- $p_j^{t_i}$: position of agent j at time t_i
- Semantic image: obtained from the segmentation network considering 6 classes:

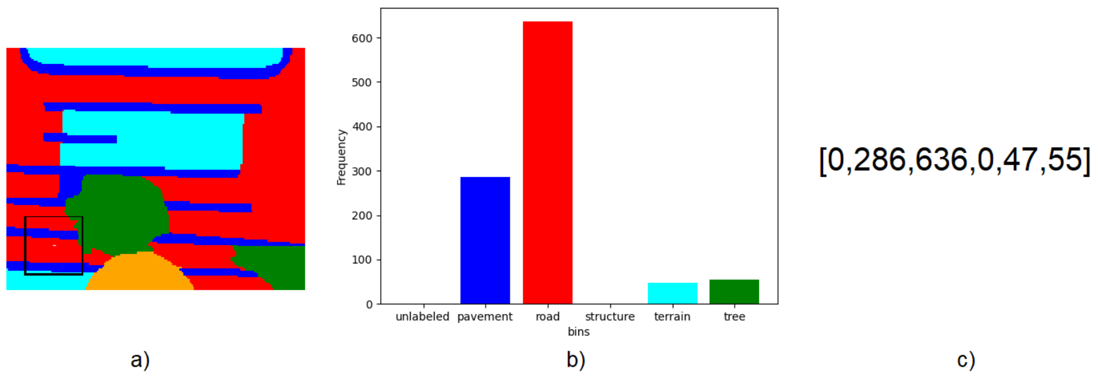

Figure 4.2: Overview of Local semantic descriptor patch
a) representation of the square patch of size 32×32 around agent4 position on the bookstore3 scene of the SDD dataset
b) representation of the semantic descriptor obtained
c) numerical values of the descriptor

1. Unlabeled
2. Pavement
3. Road
4. Structure
5. Terrain
6. Tree

Each pixel in the semantic image is a 6-dimensional vector, as the number of classes, with all zero values but a 1 in the i-th position where $c_i \in C$ represents the class of the pixel (classes order is the one defined in the list above).
For example a pixel p with value $[0, 0, 1, 0, 0, 0]$ means that it is labeled as a Road pixel.
In order to get our local descriptor, we considered a square patch of size 32×32 or 64×64 around the agent position and we compute an histogram by counting for each class $c \in C$ how many pixels inside the patch are labeled as c. An example is reported in the figure 4.2.

An important thing to highlight is that the order of the 6 bins of the descriptor is always the same so what we are basically doing is to exploit for each semantic class its pixel frequency inside the patch.

Pro and Cons

The pro of this technique are:

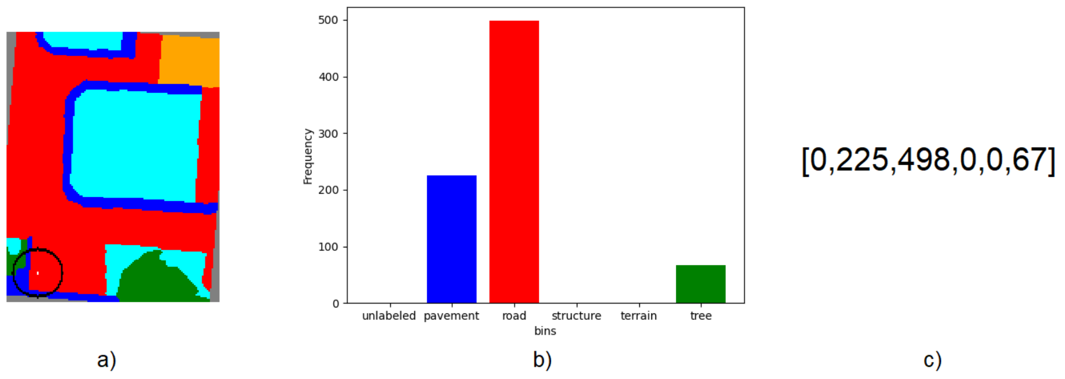

Figure 4.3: Overview of Local semantic descriptor patch
a) representation of the circular patch of diameter of 33 pixels around agent3 position on the bookstore2 scene of the SDD dataset
b) representation of the semantic descriptor obtained
c) numerical values of the descriptor

- It is fast: small computational time.
- It is easy to implement.

The cons are:

- It's not Rotation Invariant

From now on the models defined in this subsection will be called Goal-SAR-Square32 or Goal-SAR-Square64 (number represents the patch dimension in pixels).

4.1.4 Semantic Local Descriptor - Circular Patch

In order to cope with the square semantic patch problem we decide to implement a circular patch to obtain a rotation invariant local descriptor.
The idea is the same as the previous one but instead of using a square patch of sizes 32×32 or 64×64 we used a circular patch with diameter of sizes 33 or 65. An example is reported in figure 4.3.

To compute our descriptor:

- We created a fixed circular mask of the desired dimension: each pixel is represented with a 6-dimensional vector, for the white part it's $[1, 1, 1, 1, 1, 1]$ while for the black part it's $[0, 0, 0, 0, 0, 0]$. An example of masks for different diameter dimensions is reported in figure 4.4
- We multiply it to the square patch around our agent position, obtaining what we called the resulting patch.

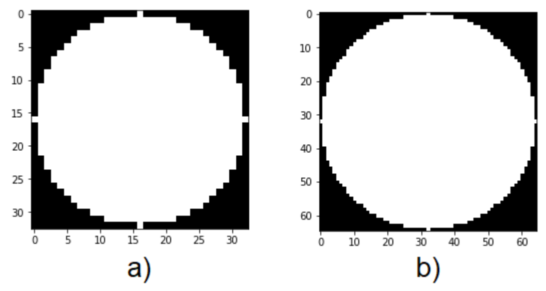

Figure 4.4: Example of circular masks
a) circular mask of diameter 33 pixels
b) circular mask of diameter 65 pixels

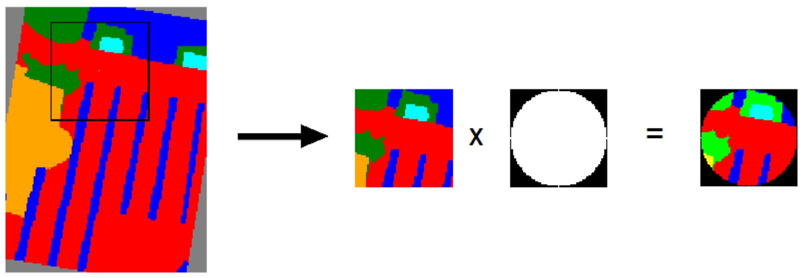

Figure 4.5: Representation of the circular patch extractor for circular patch of size 64

- We summed all the values of the resulting patch so that we get a 6-dimensional vector that represents our semantic frequency descriptor.

The visualization of the algorithm is resumed in the figure 4.5, note that the black parts around the resulting circular patch are 0 vectors, so they do not influence our semantic frequencies descriptor. The overview of the model obtained is reported in figure 4.6.
From now on, the models defined in this subsection will be called Goal-SAR-Circle33 or Goal-SAR-Circle65 (number represents the diameter dimension of the circular patch in pixels).

4.1.5 Visual Transformer of Semantic Local Descriptor

Last model presented in this thesis extracts a feature vector from the semantic image using a Visual Transformer [11].
First of all we consider a square patch of size 64×64 (with 6 channels representing each pixel's class) around human agent current position.

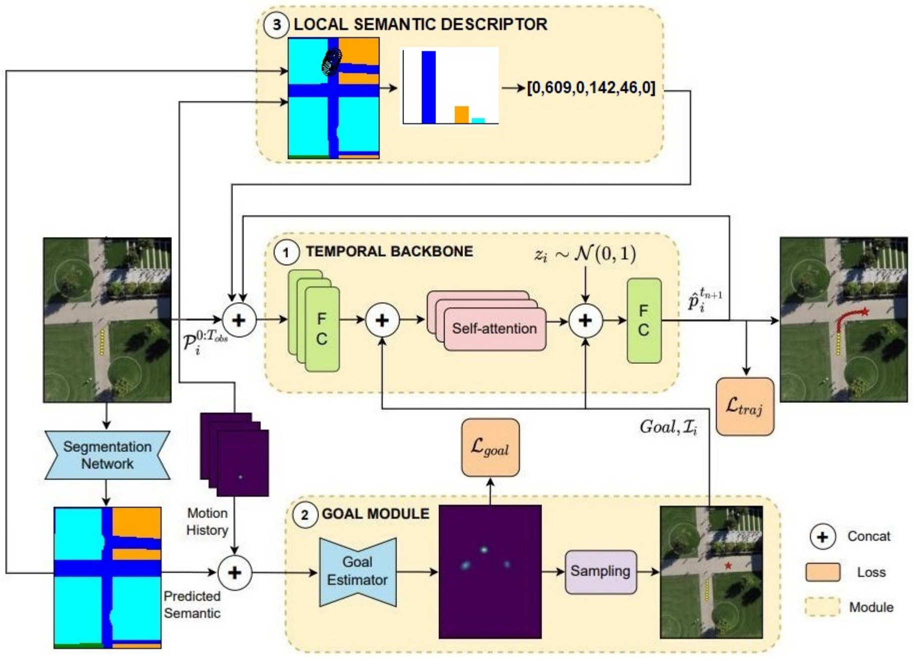

Figure 4.6: Overview of the Goal-SAR model with the local semantic circular descriptor extractor module

After that we concatenate it to 4 previous trajectory points heatmaps obtaining a tensor with 6 + 4 = 10 channels.
Then we use it as input to a Vision Transformer and the output obtained is flattened and then concatenated to the fusion layer of the Goal-SAR model.
An overview of the model created is presented in figure 4.7.

In order to create this module (called Semantic Vision transformer in the figure 4.7) we took inspiration from the Routing module of the GoalGAN model [10].
In the aforementioned paper they take an RGB patch around the agent position and they used MLP based self-attention in order to extract a feature vector from the patch.
Our idea instead is to consider a semantic patch around the agent position, consider as said before 4 previous trajectory points converted to heatmaps and use a Visual transformer based self-attention [11] in order to extract a feature vector from the semantic patch. It's important to note that due to memory over-load problems we had to decrease the batch size parameter to value 8 (while on the other models in this thesis we used value 32).

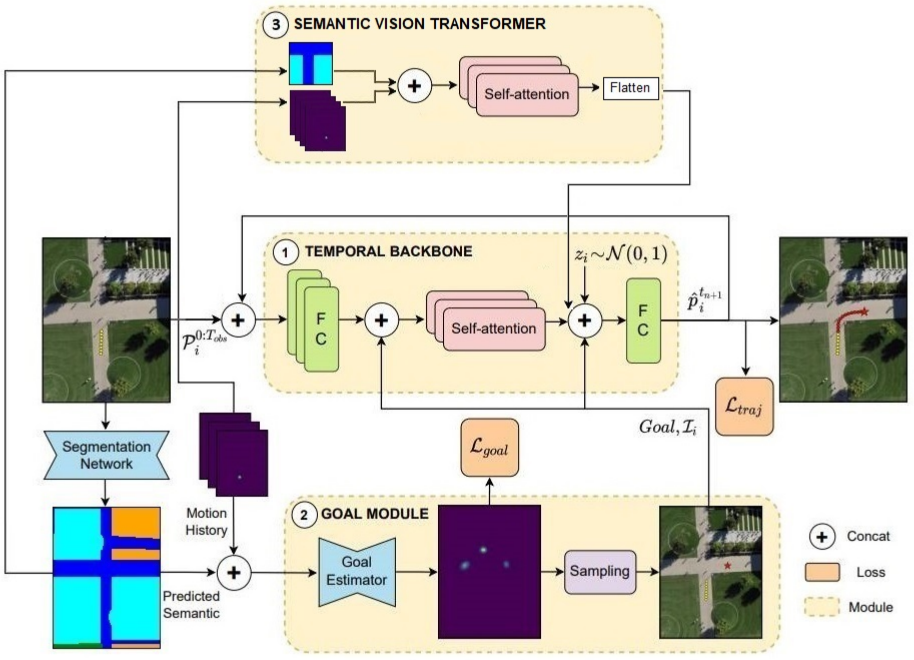

Figure 4.7: Overview of Goal-SAR with Semantic Visual Transformer

4.2 Results

In this section we report the results obtained with all the different networks used starting from the baseline and going through all our implementations and then we will focus on the results obtained applying transfer learning.
In this section for each aforementioned model we will report a table with all the results obtained in which you can find:

- First Line: results obtained training the model on SDD and test on SDD.
- From line 2 to line 6: results obtained applying transfer learning meaning training the model on SDD and test on all the five scenes of ETH/UCY dataset
- From line 7 to line 11: results obtained training the model on ETH/UCY and test on ETH/UCY.

Moreover, at the end, we will report three different tables which resume and compare the averages ADE world and FDE world metrics over the ETH/UCY and SDD dataset in both the cases of transfer learning applied and not.

4.2.1 Baseline

In table 4.1 you can find the baseline obtained for the SAR model on both the datasets (SDD and ETH/UCY) while in table 4.2 you can find the baseline obtained for the Goal-SAR model on both the datasets (SDD and ETH/UCY).

Model	**Train Set**	**Test Set**	**ADE**	**ADE world**	**FDE**	**FDE world**
SAR	sdd	sdd	10.247	10.259	18.212	18.218
SAR	sdd	eth	8.443	0.369	15.680	0.680
SAR	sdd	hotel	9.653	0.187	17.704	0.344
SAR	sdd	univ	14.334	0.317	26.175	0.580
SAR	sdd	zara1	13.352	0.296	25.184	0.558
SAR	sdd	zara2	10.280	0.228	18.600	0.411
SAR	eth5	eth	11.536	0.501	20.366	0.862
SAR	eth5	hotel	9.204	0.176	16.071	0.309
SAR	eth5	univ	18.905	0.419	35.481	0.793
SAR	eth5	zara1	17.470	0.390	31.882	0.719
SAR	eth5	zara2	10.906	0.243	19.949	0.447

Table 4.1: Baseline results for SAR

Model	**Train Set**	**Test Set**	**ADE**	**ADE world**	**FDE**	**FDE world**
Goal-SAR	sdd	sdd	7.644	7.642	11.688	11.679
Goal-SAR	sdd	eth	7.823	0.345	14.254	0.622
Goal-SAR	sdd	hotel	7.264	0.140	12.378	0.241
Goal-SAR	sdd	univ	14.326	0.316	26.001	0.573
Goal-SAR	sdd	zara1	11.379	0.251	21.060	0.513
Goal-SAR	sdd	zara2	9.189	0.204	16.304	0.360
Goal-SAR	eth5	eth	6.608	0.290	11.631	0.512
Goal-SAR	eth5	hotel	6.349	0.122	9.168	0.177
Goal-SAR	eth5	univ	12.033	0.267	21.055	0.466
Goal-SAR	eth5	zara1	7.644	0.171	11.529	0.258
Goal-SAR	eth5	zara2	6.604	0.147	10.767	0.238

Table 4.2: Baseline Results for Goal-SAR

4.2.2 Local Descriptor Module - Square Patch

In table 4.3 we report the results obtained with the Goal-SAR Semantic Sqaure Patch of size 32 while in table 4.4 we report the results obtained with

Goal-SAR Semantic Sqaure Patch of size 64.

Model	**Training Set**	**Test Set**	**ADE**	**ADE world**	**FDE**	**FDE world**
Goal-SAR S32	sdd	sdd	12.877	12.885	15.672	15.672
Goal-SAR S32	sdd	eth	18.036	0.792	36.404	1.583
Goal-SAR S32	sdd	hotel	29.484	0.568	51.348	0.995
Goal-SAR S32	sdd	univ	25.761	0.567	48.838	1.075
Goal-SAR S32	sdd	zara1	49.901	1.106	101.318	2.236
Goal-SAR S32	sdd	zara2	27.614	0.611	55.638	1.226
Goal-SAR S32	eth5	eth	6.585	0.291	9.903	0.439
Goal-SAR S32	eth5	hotel	6.651	0.129	10.408	0.202
Goal-SAR S32	eth5	univ	12.574	0.279	22.790	0.505
Goal-SAR S32	eth5	zara1	8.221	0.182	12.544	0.278
Goal-SAR S32	eth5	zara2	6.859	0.153	10.698	0.236

Table 4.3: Results Goal-SAR Square 32

Model	**Training Set**	**Test Set**	**ADE**	**ADE world**	**FDE**	**FDE world**
Goal-SAR S64	sdd	sdd	11.478	11.497	18.217	18.221
Goal-SAR S64	sdd	eth	14.221	0.627	26.185	1.137
Goal-SAR S64	sdd	hotel	17.970	0.347	30.820	0.594
Goal-SAR S64	sdd	univ	19.254	0.428	37.082	0.825
Goal-SAR S64	sdd	zara1	23.653	0.519	48.330	1.065
Goal-SAR S64	sdd	zara2	16.320	0.359	32.190	0.708
Goal-SAR S64	eth5	eth	7.302	0.321	9.734	0.431
Goal-SAR S64	eth5	hotel	6.737	0.130	10.269	0.199
Goal-SAR S64	eth5	univ	12.982	0.288	23.740	0.529
Goal-SAR S64	eth5	zara1	8.442	0.188	12.541	0.278
Goal-SAR S64	eth5	zara2	6.819	0.151	11.023	0.244

Table 4.4: Results Goal-SAR Square 64

4.2.3 Local Descriptor Module - Circular Patch

In table 4.5 we reported the results obtained with the Goal-SAR Semantic Circular Patch of size 33 while in table 4.6 we reported the results obtained with Goal-SAR Semantic Circular Patch of size 65.

Model	Training Set	Test Set	ADE	ADE world	FDE	FDE world
Goal-SAR C33	sdd	sdd	7.738	7.738	11.782	11.782
Goal-SAR C33	sdd	eth	7.720	0.339	9.949	0.429
Goal-SAR C33	sdd	hotel	6.962	0.134	10.571	0.204
Goal-SAR C33	sdd	univ	14.556	0.321	26.869	0.593
Goal-SAR C33	sdd	zara1	11.830	0.261	19.951	0.439
Goal-SAR C33	sdd	zara2	10.386	0.230	16.691	0.368
Goal-SAR C33	eth5	eth	6.475	0.284	9.980	0.438
Goal-SAR C33	eth5	hotel	6.608	0.127	10.658	0.207
Goal-SAR C33	eth5	univ	13.220	0.293	24.663	0.548
Goal-SAR C33	eth5	zara1	8.115	0.181	12.082	0.269
Goal-SAR C33	eth5	zara2	6.840	0.152	10.225	0.226

Table 4.5: Results Goal-SAR Circle 33

Model	Training Set	Test Set	ADE	ADE world	FDE	FDE world
Goal-SAR C65	sdd	sdd	7.872	7.877	12.162	12.166
Goal-SAR C65	sdd	eth	12.862	0.585	15.475	0.679
Goal-SAR C65	sdd	hotel	9.233	0.180	0.323	0.323
Goal-SAR C65	sdd	univ	14.380	0.318	27.158	0.599
Goal-SAR C65	sdd	zara1	13.295	0.291	25.622	0.557
Goal-SAR C65	sdd	zara2	10.772	0.236	20.145	0.439
Goal-SAR C65	eth5	eth	6.716	0.295	9.609	0.422
Goal-SAR C65	eth5	hotel	6.699	0.129	10.390	0.202
Goal-SAR C65	eth5	univ	13.971	0.311	26.414	0.588
Goal-SAR C65	eth5	zara1	8.437	0.187	12.404	0.276
Goal-SAR C65	eth5	zara2	7.343	0.163	12.026	0.266

Table 4.6: Results Goal-SAR Circle 65

4.2.4 Vision Transformers

In table 4.7 we reported the results obtained with Goal-SAR with Vision Transformer on Semantic Patch.

4.2.5 Average Final Results

In this section we report all the average results for each model and for each dataset in order to better understand which are the final results to highlight.
In table 4.8 you can see the comparison among all the different proposed models and the baseline on training and testing on SDD dataset.

Model	Train Set	Test Set	ADE	ADE world	FDE	FDE world
Goal-SAR ViT	sdd	sdd	7.914	7.904	12.385	12.388
Goal-SAR ViT	sdd	eth	8.735	0.380	15.619	0.664
Goal-SAR ViT	sdd	hotel	12.280	0.237	23.266	0.453
Goal-SAR ViT	sdd	univ	15.775	0.347	28.803	0.633
Goal-SAR ViT	sdd	zara1	14.960	0.331	29.875	0.660
Goal-SAR ViT	sdd	zara2	14.054	0.307	28.008	0.609
Goal-SAR ViT	eth5	eth	7.872	0.348	13.222	0.587
Goal-SAR ViT	eth5	hotel	9.010	0.174	13.914	0.269
Goal-SAR ViT	eth5	univ	15.764	0.351	28.915	0.644
Goal-SAR ViT	eth5	zara1	9.232	0.206	15.342	0.342
Goal-SAR ViT	eth5	zara2	8.431	0.187	13.246	0.291

Table 4.7: Goal-SAR Results with Vision Transformer on Semantic Patch

As you can see, the best model for both the ADE world and the FDE world metrics is Goal-SAR. In table 4.9 we compared the average ADE world and average FDE world over the five scenes of ETH/UCY dataset obtained training and testing all the models.
The best model for both the average metrics is Goal-SAR.
In table 4.10 we reported the average ADE world and average FDE world results obtained training the models on SDD and testing them on ETH/UCY (application of transfer learning). As we can see the best average ADE world value is obtained using Goal-SAR while the best average FDE world value is obtained using Goal-SAR Circle 33.

MODEL	TRAIN SET	TEST SET	avg. ADE world	avg. FDE world
SAR	sdd	sdd	10.259	18.218
Goal-SAR	sdd	sdd	**7.642**	**11.679**
Goal-SAR S. 32	sdd	sdd	12.885	15.672
Goal-SAR S. 64	sdd	sdd	11.497	18.221
Goal-SAR C. 33	sdd	sdd	7.738	11.782
Goal-SAR C. 65	sdd	sdd	7.877	12.166
Goal-SAR Sem. ViT	sdd	sdd	7.904	12.388

Table 4.8: Average Results for all models trained and tested on the SDD dataset

MODEL	TRAIN SET	TEST SET	avg. ADE world	avg. FDE world
SAR	eth-ucy	eth-ucy	0.346	0.626
Goal-SAR	eth-ucy	eth-ucy	**0.199**	**0.330**
Goal-SAR S. 32	eth-ucy	eth-ucy	0.207	0,332
Goal-SAR S. 64	eth-ucy	eth-ucy	0.216	0.336
Goal-SAR C. 33	eth-ucy	eth-ucy	0.208	0.338
Goal-SAR C. 65	eth-ucy	eth-ucy	0.217	0.351
Goal-SAR Sem. ViT	eth-ucy	eth-ucy	0.253	0.426

Table 4.9: Average results for all models trained and tested on ETH/UCY dataset

MODEL	TRAIN SET	TEST SET	avg. ADE world	avg. FDE world
SAR	sdd	eth-ucy	0.279	0.515
Goal-SAR	sdd	eth-ucy	**0.251**	0.462
Goal-SAR S. 32	sdd	eth-ucy	0.729	1.423
Goal-SAR S. 64	sdd	eth-ucy	0.456	0.866
Goal-SAR C. 33	sdd	eth-ucy	0.257	**0.407**
Goal-SAR C. 65	sdd	eth-ucy	0.322	0.519
Goal-SAR Sem. ViT	sdd	eth-ucy	0.321	0.604

Table 4.10: Average results for all models when transfer learning is applied

4.3 Considerations and Problems

The quantitative results presented in the last section can be analyzed to gain insight into the reasons why some approaches work better than others:

- **SAR**: comparing the table 4.10 with table 4.9 we can see that SAR model is more prone to get better results when transfer learning is applied.
  In table 4.10 in fact we got an improvement of both the average ADE world and average FDE world metrics of about 20%.
  In opposition looking at table 4.9 using SAR we got the worst average ADE world and FDE world when trained and tested on ETH/UCY dataset.
- **Goal-SAR**: comparing Goal-SAR with the SAR model we can see that the introduction of the Goal module let us always get better results on both SDD and ETH/UCY datasets as shown in tables 4.9 and 4.8. On the other hand, the Goal module does not allow Goal-SAR (compared to SAR) to get benefits from the application of transfer learning.
  In fact, comparing Goal-SAR results in table 4.10 with table 4.9, applying transfer learning we got worst results for both averages ADE world=0.251 and FDE world=0.462 compared to baseline ones ADE world=0.199 and FDE world=0.330.
- **Goal-SAR Semantic Square**: looking at tables 4.8 and 4.10 applying a square semantic patch of size 32 to extract a local semantic descriptor led us to the worst model in both cases.

Increasing the size of the patch helps get better results but still really worse compared with Goal-SAR baseline ones.
Contrastingly looking at table 4.9 we got really similar results compared to the baseline ones training and testing this model on ETH/UCY .
Considering these results we can state that using a square semantic patch in this way, we are introducing useless information to the temporal transformer and this could be related to its non-rotation invariant feature.

- **Goal-SAR Semantic Circle**: among all the models created in this thesis so far, this is the only one that has been able to obtain better performance compared to the baseline when transfer learning is applied.
  As we can see from the table 4.10 we got an average ADE world of 0,257 and an average FDE world of 0,407 compared to the same situation for the baseline where we got an average ADE world of 0,251 and an average FDE world of 0,462.
  From these results we can state that introducing the semantic Circle patch (of diameter 33) descriptor extractor module to Goal-SAR, can lead to a better knowledge transfering. Despite this, transfer learning results (Table 4.10) are not able to beat the baseline ones obtained without using transfer learning (Table 4.9), which are an average ADE world of 0,199 and an average FDE world of 0,330.
  It's important to note that, different from the previous model, increasing the size of the circular patch does not find any benefit.

- **Goal-SAR Semantic Vision Transformer**: this is the most complex model among the proposed ones.
  Looking at the three tables 4.8 , 4.9 and 4.10. The introduction of more complexity did not lead us to a better model; in fact, we always got worst results compared to Goal-SAR and Goal-SAR Circle.
  It's important to highlight that we are currently doing more experiments with this model on long-term trajectory prediction. On this last problem, using the same batch size parameter as the one used in [7] for Goal-SAR, we obtained better results for both ADE world and FDE world on SDD datset with respect to the baseline.

4.3.1 Problems

There could be different reasons why we are dealing with these slightly worst results obtained with our proposed models compared to Goal-SAR baseline.
All of these reasons have in common that we are concatenating to the input of the temporal transformer useless or partial information.
This happens differently times for all of our approaches:

1. The semantic patch always covers the same semantic class for the entire agent trajectory.

2. Some agents remain in the same position for the entire sequence.
3. Some agents' positions are on the border of the scene or outside of it, leading to, respectively, an incomplete descriptor (the patch is partially inside the scene) or a 0 descriptor.

To get a better comprehension of the probability in which the aforementioned events happen, for both the SDD and ETH/UCY datasets, we report the statistics obtained when dealing with a circular patch to compute the local semantic descriptor.

1) Similar Descriptor for entire sequence

In tables 4.11 (considering Goal-SAR Circle 33) and 4.12 (considering Goal-SAR Circle 65) we report the percentages of the agents whose trajectories define a sequence of semantic patches with similar descriptors.
We defined 3 different thresholds **T** related with the distance between descriptors of the same agent path:

- 1: same descriptors for the entire sequence
- 5: The sequence of descriptors for an agent has less than 5 pixels labeled differently with respect to the descriptor computed at its starting position.
- 10: The sequence of descriptors of an agent have less than 10 pixels labeled differently with respect to the descriptor computed at its starting position.

T	Model	Dataset	Agents Same Sequence	Total Agents	Percentage
1	Goal-SAR C. 33	sdd	202	1522	13.27%
1	Goal-SAR C. 33	eth-ucy	58	193	30.05%
5	Goal-SAR C. 33	sdd	212	1522	13.92%
5	Goal-SAR C. 33	eth-ucy	62	193	32.12%
10	Goal-SAR C. 33	sdd	215	1522	14,13%
10	Goal-SAR C. 33	eth-ucy	63	193	32.64%

Table 4.11: Percentage of similar descriptor sequences (Goal-SAR C : Goal-SAR circle) where T represents the threshold value in pixels

As we can see from table 4.11 the percentages on the ETH/UCY dataset are really high considering a circular patch of diameter 33 pixels.

T	Model	Dataset	Agents Same Sequence	Total Agents	Percentage
1	Goal-SAR C. 65	sdd	212	1522	13.92%
1	Goal-SAR C. 65	eth-ucy	20	193	10.36%
5	Goal-SAR C. 65	sdd	212	1522	13.92%
5	Goal-SAR C. 65	eth-ucy	25	193	12.95%
10	Goal-SAR C. 65	sdd	214	1522	14.06%
10	Goal-SAR C. 65	eth-ucy	26	193	13.47%

Table 4.12: Percentage of similar descriptors (Goal-SAR C : Goal-SAR Circle) where T represents the threshold value in pixels

Model	Dataset	Agents Same Position	Total Agents	Percentage
Goal-SAR C. 33	sdd	202	1522	13.27%
Goal-SAR C. 33	eth5	17	193	8.81%

Table 4.13: Percentage of agents always in the same position

2) Agents in the Same Position for entire sequence

In table 4.13 we reported the percentages of human agents who remain for the entire sequence in the same position in the scene. This case represents a subset of the previous one because the descriptors computed for the aforementioned agents are exactly the same for the entire sequence.

3) Partial Descriptors

In tables 4.14 and 4.15 we reported the percentage of the agents positions which are outside or near the border of the scene.
We computed it for both datasets (SDD and ETH/UCY) considering the total number of agent positions.

Model	Dataset	Incomplete Descriptors	Total Descriptors	Percentage
Goal-SAR C. 33	sdd	4552	28918	15.74%
Goal-SAR C. 33	eth5	432	3667	11.78%

Table 4.14: Percentage of partial descriptors calculated for the Goal-SAR Circle 33

These situations led to partial semantic descriptors. As we can see from table 4.15, using a circular patch of 65 pixels in diameter, about 40% of the times we concatenate incomplete information with the input of our temporal transformer.

Model	Dataset	Incomplete Descriptors	Total Descriptors	Percentage
Goal-SAR C. 65	sdd	11969	28918	41.39%
Goal-SAR C. 65	eth5	1401	3667	38.21%

Table 4.15: Percentage of partial descriptors computed for the Goal-SAR Circle 65

Agents outside the scene

In table 4.16 we reported the percentages of agents positions outside the scenes for each dataset (SDD and ETH/UCY) with respect to the total number of agent positions.
These is a subset of the previously considered case.

Dataset	Outside Positions	Total positions	Percentage
sdd	2095	28918	7.24%
eth5	76	3667	2.07%

Table 4.16: Percentage of Positions Outside the scene

5 Conclusions and Future Works

5.1 Conclusions

Pedestrian trajectory prediction is one of the many topics in which the advent of deep learning completely changed the approach to the problem. While the first solutions were based on physics modelling, nowadays state-of-the-art approaches are mostly based on deep learning techniques. Data-driven models such as deep learning models are very sensitive to the quantity and quality of the training data and how this data is then presented to the model. This is the reason why the goal of the thesis was to find ideas in order to efficiently apply transfer learning. The final contribution of this thesis is a study on the effectiveness of different techniques to transfer knowledge between different scenes. All the different techniques used are based on trying to extract a local descriptor from the semantic image in correspondence to the agent position. This comes from the idea that scenes characterized by a similar semantic context could determine similar behaviors of human agents trajectories. In chapter 4 we proposed different techniques. The one that led to the better results in the transfer learning setting introduced a parallel module to the GoalSAR model. This module extracts a circular patch from the semantic image and generates a histogram representing the frequency of each semantic class inside the patch. The numerical vector that represents the computed frequency histogram is then concatenated to the temporal transformer of the GoalSAR model. When transfer learning is not applied this technique led to similar results compared to the baseline GoalSAR model. Moreover, we deal with short-term trajectory prediction while new tests and studies at the University of Padua are considering long-term case in which using the same model proposed in chapter 4.2.4 (Vision Transformer for semantic feature vector extraction) led to better results with respect to the GoalSAR baseline for both average ADE and average FDE metrics.

5.2 Future Works

New techniques to extract a semantic local descriptor or new way to create a semantic histogram could increase performances of the models when dealing with transfer learning. Moreover, the creation of a big synthetic dataset that could be used to apply transfer learning for both SDD and ETH/UCY datasets could be interesting in order to get a stronger demonstration of the importance of the semantic context when dealing with pedestrian trajectory prediction.

References

[1] Alexandre Alahi et al. "Social LSTM: Human Trajectory Prediction in Crowded Spaces". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 961–971. DOI: `10.1109/CVPR.2016.110`.

[2] Javad Amirian, Jean-Bernard Hayet, and Julien Pettre. "Social Ways: Learning Multi-Modal Distributions of Pedestrian Trajectories With GANs". In: June 2019, pp. 2964–2972. DOI: `10.1109/CVPRW.2019.00359`.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. eprint: `arXiv:1409.0473`.

[4] Lamberto Ballan et al. *Knowledge Transfer for Scene-specific Motion Prediction*. 2016. eprint: `arXiv:1603.06987`.

[5] Tamer Basar. "A New Approach to Linear Filtering and Prediction Problems". In: *Control Theory: Twenty-Five Seminal Papers (1932-1981)*. 2001, pp. 167–179. DOI: `10.1109/9780470544334.ch9`.

[6] Denny Britz et al. "Massive Exploration of Neural Machine Translation Architectures". In: *ArXiv* abs/1703.03906 (2017).

[7] Luigi Filippo Chiara et al. *Goal-driven Self-Attentive Recurrent Networks for Trajectory Prediction*. 2022. eprint: `arXiv:2204.11561`.

[8] Kyunghyun Cho et al. "Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation". In: *EMNLP*. 2014.

[9] Patrick Dendorfer, Sven Elflein, and Laura Leal-Taixé. *MG-GAN: A Multi-Generator Model Preventing Out-of-Distribution Samples in Pedestrian Trajectory Prediction*. 2021. eprint: `arXiv:2108.09274`.

[10] Patrick Dendorfer, Aljoa Oep, and Laura Leal-Taixé. *Goal-GAN: Multimodal Trajectory Prediction Based on Goal Position Estimation*. 2020. eprint: `arXiv:2010.01114`.

[11] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. eprint: `arXiv:2010.11929`.

[12] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. eprint: `arXiv:1406.2661`.

[13] Agrim Gupta et al. *Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks*. 2018. eprint: `arXiv:1803.10892`.

[14] Dirk Helbing and Péter Molnár. "Social force model for pedestrian dynamics". In: *Phys. Rev. E* 51 (5 May 1995), pp. 4282–4286. DOI: `10.1103/PhysRevE.51.4282`. URL: `https://link.aps.org/doi/10.1103/PhysRevE.51.4282`.

[15] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–1780.

[16] Parth Kothari, Sven Kreiss, and Alexandre Alahi. "Human Trajectory Forecasting in Crowds: A Deep Learning Perspective". In: *IEEE Transactions on Intelligent Transportation Systems* 23 (2022), pp. 7386–7400.

[17] Mihee Lee et al. *MUSE-VAE: Multi-Scale VAE for Environment-Aware Long Term Trajectory Prediction*. 2022. eprint: `arXiv:2201.07189`.

[18] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. "Crowds by Example". In: *Computer Graphics Forum* (2007). ISSN: 1467-8659. DOI: `10.1111/j.1467-8659.2007.01089.x`.

[19] Tony Lindeberg. "Scale Invariant Feature Transform". In: vol. 7. May 2012. DOI: `10.4249/scholarpedia.10491`.

[20] Tony Lindeberg. "Scale Invariant Feature Transform". In: vol. 7. May 2012. DOI: `10.4249/scholarpedia.10491`.

[21] Karttikeya Mangalam et al. *From Goals, Waypoints & Paths To Long Term Human Trajectory Forecasting*. 2020. eprint: `arXiv:2012.01526`.

[22] Karttikeya Mangalam et al. *It Is Not the Journey but the Destination: Endpoint Conditioned Trajectory Prediction*. 2020. eprint: `arXiv:2004.02025`.

[23] Stefano Pellegrini, Andreas Ess, and Luc Van Gool. "You'll Never Walk Alone: Modeling Social Behavior for Multi-target Tracking". In: Sept. 2009, pp. 261–268. DOI: 10.1109/ICCV.2009.5459260.

[24] Alexandre Robicquet et al. "Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes". In: ed. by Bastian Leibe et al. 2016.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. eprint: arXiv:1505.04597.

[26] Amir Sadeghian et al. "CAR-Net: Clairvoyant Attentive Recurrent Network". In: (2017). eprint: arXiv:1711.10061.

[27] Amir Sadeghian et al. *SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints*. 2018. eprint: arXiv:1806.01482.

[28] Tim Salzmann et al. *Trajectron++: Dynamically-Feasible Trajectory Forecasting With Heterogeneous Data*. 2020. eprint: arXiv:2001.03093.

[29] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *NIPS*. 2015.

[30] Jimei Yang et al. "Context Driven Scene Parsing with Attention to Rare Classes". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3294–3301. DOI: 10.1109/CVPR.2014.415.

[31] Jiangbei Yue, Dinesh Manocha, and He Wang. *Human Trajectory Prediction via Neural Social Physics*. 2022. eprint: arXiv:2207.10435.

[32] Shengjia Zhao, Jiaming Song, and Stefano Ermon. *InfoVAE: Information Maximizing Variational Autoencoders*. 2017. eprint: arXiv:1706.02262.

Acknowledgments

I would like to thank my supervisor Professor Lamberto Ballan, my co-supervisors Sourav Das and Luigi Filippo Chiara. Their guidance and assistance have been fundamental for all the aspects of this work, especially in the definition of the tasks and in the discussion of the results. I thank my university colleagues Alberto, Alessandro, Nicola and Pietro for their support and help during all my five years long computer engineering career. I also thank the University of Padova for having given me the possibility to accomplish my goals. I thank my girlfriend Sofia who helped me psychologically to go through the adversities of my carrier path and my friends who accompanied me during this experience. Most important I want to thank my family, parents and brothers, who have allowed all this. They support and motivate me through all the most important steps in my life.