Multimedia Communications Project: Pedestrian trajectory prediction

Simone Zamboni - 178641 - Information Engineering and Business Organization

Summary

Introduction

A software capable of analyzing the video streaming of a security camera installed in public places and detecting anomalous behavior could immediately identify critical situations e report anomalies to the surveillance agents. This would result in a reduction of staff necessary for surveillance and greater safety in public places.

To have a software with this functionality you need an algorithm that can predict the "normal" behavior of people, so as to be able to identify the differences between the forecast of the usual behavior and what is actually happening. If this difference is high it means that anomalous behavior is therefore taking place. In this project the behavior of an individual is identified as the trajectory that it traverses during its stay in the video.

The challenge is therefore to create an algorithm able to predict the trajectories of passers-by. There difficulty in creating such a model is that people walk in complex ways and therefore predicting their future location is difficult.

Various models of pedestrian behavior have been proposed: the linear model, Collision Avoidance (described in [1]), Social force (also described in [1]), and the Iterative Gaussian Process (exposed in [2]).

An innovative idea in this field was to use recurrent LSTM neural networks ("Long Short Term Memory") that contain within them a state and can thus provide for the future trajectory of the pedestrian taking into account the previous actions. In this a so-called LSTM "cell" is associated with each pass through. The model he uses this technique is described in publication [3].

This project starts from the knowledge base of the aforementioned publication and from the code (in Python language with the use of the Tensorflow library) which has been referred to as the official implementation of the publication (available at the link [4]).

objectives

The goal of the project is to modify the original model and then verify if these new variants of the starting model have superior performance. The changes will come carried out starting from the code of the official implementation of the model and will then be carried out tests to detect which of the models performs best, using the same parameters as configuration for all models and comparing the final positioning error in the test.

The changes to be made are the following:

1. Add to the current model inputs (which are the pedestrian position and the grid of the adjacent pedestrians) the objective of the pedestrian, defined as the last known position of the pawn in the video;
2. Replace the adjacent pedestrian grid with a vector containing pedestrian positions closer;
3. Combine the two previous changes into one model

Reference model

In the model of the publication [3] each pedestrian is an LSTM cell. Each cell has a state and produces a new state and an output based on the inputs received and the previous state.

The input consists of two parts: the current position of the pedestrian and the social grid it represents adjacent pedestrians at that time.

Below is a diagram that summarizes how the model works:

In which the square called LSTM represents the LSTM cell that simulates the behavior of the pawn, "h" represents the state of the cell, (x, y) the pair of coordinates indicating the position of the pawn, which are first used as an input part of the cell and which then constitute the output of the cell, Social Pooling indicates information regarding the position and status of other passers-by (ie the social grid) provided in input to the cell, t-1 the previous moment, t the current moment and t + 1 the next time.

After this general introduction we will now go into more detail to analyze all the components and the operations performed by the model.

The position of the pawn are two coordinates in meters normalized between -1 and 1 and represent the xey position of the pedestrian at the current frame. These coordinates are then transformed by one "Embedding function" which in the code was implemented with a multiplication between the vector of the coordinates and a matrix of weights, to which must be added another vector (called "biases" vector). The result of this multiplication must then be applied to a "rectifier" activation function.
In the publication this operation is described by the following formula:

In which and $_t$ ^{the}represents the part of the input concerning the coordinates of the pedestrian i at time t, x_t^{the} the x coordinate of the pedestrian i at time t, y_t^{the} the y coordinate of the pedestrian at time t, W_{and} the matrix weight (and the vector with the "biases") used for this operation and Φ the activation function rectifier type.

3

Page 4

The adjacent pedestrian grid is a square grid of side N that has depth D, with D che represents the size of the cell status. This grid is created as follows:

H_t^{the} represents the grid of the pedestrian i at time t, men represent the cell at the m-th line e to the nth column of the grid, j is a pawn in the frame other than the current pawn, $1_{m,n}$ one function that takes only the values 0 and 1, and is not null only if the pedestrian j is in the cell m, n of the grid, result obtained on the basis of the length of the distance vector between the pawn i and the pawn j, which we see calculated in square brackets, and represents $h_{j\ t-1}$ the status of the pawn j at time t-1.

This newly created matrix of NxNxD dimensions then goes through a phase completely analogous to the one observed first by the coordinates, that is, multiplied by a matrix of the weights, comes to him added a vector and goes through a "rectifier" activation function:

The term in $_{i\,t}$ represents the part of the cell input obtained from the social grid, $H_{t\quad i}$ the social grid, W_a the weight matrix (and the vector with the "biases") of this specific operation and as for the coordinates Φ represents the rectifier activation function.

Concatenating and $_i$ with $a_{t\,i}$ the pedestrian input $_i$ is obtained at time t.

They are inserted as input into the cell and $_{t\quad i}$ and a_{the} concatenated together with the previous state of the cell (h_{t-1}), and with the proper weights of the cell (W_l) we obtain the new state $_t(h_i$

The output of the cell are 5 parameters, which describe a multivariate Gaussian distribution to two dimensions representing the probability distribution of the next pedestrian position.
These 5 parameters are the mean in x and the mean in y (together indicated as $\mu_{t\quad i}$), the deviation standard in x and in y (indicated with $\sigma_{t\quad i}$) and the correlation coefficient ($\rho_{t\quad i}$), consequently the predicted coordinates ($\hat{\ }$, $\hat{y}_i$) can be considered as a normal distribution of parameters μ_i, σ_i and ρ_t

The goal of the algorithm is to minimize the negative summation of the logarithm of the probability that the real coordinates are at the point of the expected coordinates (this summation is called "Likelihood loss" and indicated with L_i):

4

Page 5

Notes on the results

Some notes on the use parameters of the algorithm used by the writers of the publication, parameters used to create the results also in this project.

Network training took place on 80% of the 4 out of 5 video annotated frames, leaving 20% of the 4 videos as validation, while the test was performed on the video not used in the phase of

training. The test involves letting the algorithm observe a pedestrian's behavior for 8 frames and predict the trajectory for the next 12. Annotated frames have a distance of one on the other hand of 0.4 seconds (so the annotations used are at 2.5 fps), this means that the algorithm observes the behavior of a pedestrian for 3.2 seconds and foresees the trajectory for the following 4.8.

Another very important parameter is the size of the status of the cells representing pedestrians which has been set to 128.

Implementation notes

Bookstores

The original code was implemented with Python 2.7 and uses the tensorflow library with version <1.0. At the beginning of the project a conversion was then made by modifying the code for bring it to the current version of tensorflow (1.5), while the Python version remained unchanged. All tests were performed on an Nvidia GTX 950M GPU with CUDA software 8.0 and installed CudNN 6.0 in a Linux environment (Ubuntu distribution).

The data

The same two datasets used in the publication and implementation were used. The first is called EHT and has two videos inside: hotel and univ. The second is called UCY and has inside three videos: zara1, zara2 and univ.

All these videos were recorded at 15 fps and annotated every 0.4 seconds, ie every 6 frames. Subsequently the creators of the datasets have interpolated the data to obtain an annotation for each frame and uniform the two datasets. It would therefore be possible to train and test the algorithm with 15 annotations per second, but to remain consistent with the publication [3] it was decided to provide in input to the model one frame every 6, thus using an annotation every 0.4 seconds as indicated by the publication.

To do this in data pre-processing, which is done every time it is started the training or the tests, only one frame is kept annotated every 6. The frequency with which i frames are kept is specified by the configurable parameter "takeOneEveryNFrames" present

in the SocialDataLoader component, so as to make it possible to make annotated tests a different temporal distance between them.

Test

The tests were performed as indicated by publication [3]: let the algorithm be observed for 8 frame the behavior of a pedestrian and making him predict the trajectory for the next 12. THE frames with annotations used are 0.4 seconds apart.

In the official implementation the output of each cell representing a pawn is a normal one multivariate in two dimensions. To obtain the aforementioned x and y coordinates from this distribution a function is used that takes the coordinates of a random point of this normal generated by the cell. Consequently every time the same test is repeated with a model already trained the result differs (in any case almost never more than 10% because the distribution is a lot narrow). It becomes time consuming to understand which trained model performs in a better way on a video, not to mention that in a simulation environment use a similar one method to move a virtual pawn would leave a lot to chance.

In light of these considerations it was decided to use the aforementioned future coordinates in the tests from the cell instead of a random point of the output distribution of the cell directly the values μ_x and μ_y of the distribution. This change has influenced the ranking of the best models: a model that performed in a way also with this modification performs very similarly, but it is sufficient to test it only once instead of several times to evaluate the performance, and can be better used for a simulation.

Implementation of variants

Goal

The goal of this variant of the original model is to add to existing inputs of the model the "goal" of the pedestrian. The last position was defined as the "goal" of a pedestrian known of the pedestrian within the video, ie the point where the passerby disappears from the scene.

It was therefore necessary to modify the data pre-processing algorithm contained in the class SocialDataLoader. This method reads the .csv file of the annotations of each when called video and create an array in which for each frame there is a list of pedestrian IDs with the x and y position, and then save this array in a .pkl file (in case it is called in the training phase i there are two arrays, one containing the training data and the other those for validation). The new variant of the component, besides performing the operations described above, it scrolls all the frames of each video and for each pedestrian saves the last known position, to then memorize these information in the .pkl file.

In the training or test phase the .pkl file of the pre-processed data is processed to obtain the input of each cell in the netx_batch () method of the SocialDataLoader class. In this new variant it was modified that method so that for each pedestrian the vector of the coordinates is expanded in so that inside it are present, in addition to the coordinates of the current passer-by position, also the coordinates of his goal.

This new 4-coordinate vector passes through the same multiplication process with the matrix of the weights, sum with the vector of the "biases" and transformation by the function of activation, which was previously crossed only by the position coordinates of the pedestrian.

So in the original model:

While in this variant of the model (with gx_i the x coordinate of the pedestrian's objective i and gy_i there y coordinate of the objective of the pedestrian i):

$$= (\quad , \quad , \quad , \quad ;)$$

The rows of the matrix of the weights W_{and} are then passed from 2 to 4 in order to be multiplied with the new coordinate vector, while the rest of the model has remained unchanged.

The pattern of the model is consequently modified, because in input what was previously the couple of the coordinates becomes a vector of 4 numbers which correspond to the coordinate pair of the current pawn position and another pair of coordinates corresponding to the pawn's goal:

The motivation that led to the creation of this variant is to provide the algorithm additional information to make it possible to better predict the trajectory of passers-by. Knowing where a pedestrian will disappear from the scene it is reasonable to think that one could foresee better its trajectory. However, there may also be some disadvantages in adding this information: pedestrians behave in a complex way and can move in various directions in the scene, even on the opposite side with respect to their "goal", potentially making this one new confusing additional information. A classic example of this behavior is when a pedestrian stops or radically changes his trajectory to see a window, it he is therefore not going directly to his goal, and consequently the algorithm could

be disoriented by the information of the location of the scene exit of that particular one pedestrian.

In the results table and in the conclusions we will be able to verify if this additional information improve performance or disorient the algorithm.

Social Array

The social part of this algorithm is given by a pedestrian-centered NxNxD grid which represents for each cell of the grid the sum of the states of the pedestrians present in the frame in that cell. The presence of a specific pawn in a given cell is given by the position of this pawn with respect to the pawn on which the grid is centered. Below is an illustration from publication to see how the social grid is created:

The goal of this variant is to replace this grid with a vector containing the positions of the nearby pedestrians. The component that calculates the grid has been modified to instead create a array.

This new implementation for each pawn first calculates the distance with all the other pawns, then populate the array with the x and y coordinates of the other pedestrians in order from the nearest (at position 0) to farther (in the N-1 position in the array). This array has a fixed size N * 2, with N number of neighboring pedestrians that you want to consider, number multiplied by a factor of two because for each one of these pedestrians the x coordinate and the y coordinate are stored.

In the case where the number of pawns in the frame is greater than N + 1, in the array of each pawn they will be only the nearest N passers are present. Position 0 in the array will contain the pedestrian closest to that of which the array is calculated, and at the N-1 position the farthest pawn which however falls within the N plus neighbors.

In the case where N> = number of pedestrians in the frame means that there are not enough passers for completely fill the social array. In this case, leave the empty positions in the array at 0 social creates problems, because the algorithm would interpret these 0 as pedestrians stopped at center of the scene, and consequently would make it stay away from that position. To avoid this inconvenience and leave some flexibility to the algorithm, so that it can also operate

in scenes with a number of pedestrians <= N, in the otherwise empty spaces of the array the le are repeated pedestrian coordinates already in the array, starting from those of the passer in position 0.

Here is an example:

8

Page 9

If N = 5 and there are 3 pawns (p1, p2, p3) in the frame, assuming we must calculate the array of the passer p1, and the closest to this pedestrian is p2, the array of p1 will be composed as follows: [p2x, p2y, p3x, p3y, p2x, p2y, p3x, p3y, p2x, p2y], that is we will have in order the coordinates of p2, p3, p2, p3 and p2. This it happens because the array is first populated with the first position occupied by p2 because it is closer, and then p3 because it is further away. Then the 3 positions are filled (corresponding to 6) coordinates) that are empty, repeating the coordinates of the pedestrians already present in the array starting from passing in position 0, until the array is completely filled.

This method allows to use the algorithm both in very populated videos and in video less populated, in which a lot of importance is given to the few pedestrians in the scene since the few passers-by present are found several times in the array.

In the remote case where in a frame there is only one pawn p1, in each position of this array a solitary passer is a fictitious pedestrian with x coordinates: x_{p1} -2 and y: $_{p1}$ -2, so this invented loop is very far away (all the coordinates being between -1 and 1 if p1 is in the upper right corner the fictitious pedestrian will be just beyond the lower left corner of the video) and therefore does not influence the behavior of the solitary pawn.

Note that with a parameter N = 4, parameter in the implementation and in the table of the results we will call the parameter "grid_size", the Social_LSTM model considers 16 cells (N x N) in which each can have more than one pawn inside, while in the Social Array variant (described in this chapter) with N = 4 the vector is of length 8 (N * 2) and therefore the algorithm can consider al maximum only 4 pedestrians nearby.

The original model that uses the grid can consider empty positions, has a social element more in-depth because it also considers the states of pedestrians in the grid and has no limit to the number of pedestrians considered because the sum of the states of the pedestrians is made for each cell inside. On the other hand, the array model has much more precise information (in fact it uses the exact position and not approximated on a grid) on the most important passers in the frame, ie those closer to the pedestrian whose array is calculated. These coordinates are also already sorted by distance so as to help the algorithm focus even more attention on the nearest pedestrians. There an array variant is therefore characterized by a smaller number of information, but more precise and important. These features could give a greater advantage to this variant

compared to the original model because the cell could learn better with clearer information and coincided.

Goal + Social Array

In this model the two variants previously described were combined into one model, so as to combine the benefits of the two models.

The implementation was a simple union of the code of the other 2 models and they were not make other changes.

Results

Results table legend

The table at the end of this chapter contains the results of the tests conducted on the various models with different input parameters.

The "Model" column describes the model used: Social_LSTM for implementation original, Goal for the variant with the "goal" and Social Array for the variant that replaces the grid with an array and Goal + Array for the variant that combines the "goal" and the array instead of the grid.

The "Video test" column shows on which video the model test was performed. Such as described in the publication each network was trained on the other four videos and then tested on the remaining one. The videos are 5 and have been listed as follows:

0 = UCY dataset - video zara01

1 = UCY dataset - video zara02

2 = EHT dataset - video univ

3 = ETH dataset - video hotel

4 = UCY dataset - univ video

The number of video frames considering a frame with annotations every 0.4 seconds are:

0 = 867 (in training train 694 and validation train 173)

1 = 1051 (in training train 841 and validation 210)

2 = 1433 (in training train 1147 and validation 286)

3 = 1900 (in training train 1520 and validation 380)

4 = 540 (in training train 432 and validation 108)

In the case where in this column the word "Media" is present instead of the video number means that those results refer to the average of the results of that model on all videos.

The "Grid size" column indicates the side of the social grid that is of size grid_size x grid_size x lunghezza_stato. In the case of the Social Array (and Goal + Array) model it indicates how many pedestrians can contain the social array (which will be a vector of dimension grid_size * 2).

The "N epochs" column indicates for how many epoch the training for was done generate the results obtained.

The "Best train epoch" column indicates the epoch with the lowest validation loss, value present in the column “Best train epoch val. loss ".

The "Best test epoch" column indicates the model of which epoch was used to generate the error minor in the test, value reported in the "Best test epoch error" column. Note that the publication defines three types of errors, and these results refer only to the type error "Average displacement error". The error value returned by the algorithm has been multiplied by a

factor 15 to be able to denormalize and obtain the error in meters, consequently the value in "Best test epoch error" column is in meters.

Other parameters

Other parameters that can be set in the code but have been left unchanged at the default values I'm:

- rnn_size = 128, (how big is the RNN cell state vector)
- maxNumPeds = 70 (maximum number of pedestrians in a frame managed by the algorithm)
- batch_size = 16
- grad_clip = 10,
- learning_rate = 0.005,
- decay rate = 0.95,
- drop_out_keep_probability = 0.8,
- embedding_size = 64, (the columns of the matrix and the length of the vector of the "biases" used for the creation of a_i $_t$ and e_i)
- lambda_param = 0.0005

- neighborhood_size = 32
- validation percentage = 0.2 (what is the percentage of each video used in the phase of training for validation)
- observation_length = 8 (in the test how many frames the algotir will be observed before predirre)
- prediction_length = 12 (in the test how many frames the algorithm must predict after observing observation_length frame)
- sequence_length = 20 (this value indicates how long a sequence of frames is consecutive on which training is done; for this parameter the was chosen number 20 because in the test the algorithm observes for 8 frames and predicts the other 12, so it is it seemed appropriate to carry out training on sequences 8 + 12 = 20 frames long

In all test cases and for all videos annotations were used every 0.4 seconds, such as indicated in the publication.

Results table

Here is the table of results:

Video Model	test	Grid size	N epochs	Best train epoch	Best train epoch val. loss	Best test epoch	Best test epoch error
Social LSTM	0	4	10	10	-54.01	10	1,362
Goal	0	4	10	6	-27.42	5	1,225
Social Array	0	8	10	8	**-57.88**	5	**1,127**
Goal + Array	0	8	10	5	-27.61	6	1,135

Social LSTM	1	4	10	9	-52.76	2	1,625
Goal	1	4	10	5	-26.64	8	1,466
Social Array	1	8	10	9	**-52.82**	2	2,112
Goal + Array	1	8	10	7	-27.83	8	**1,391**
Social LSTM	2	4	10	4	-39.02	8	1,440
Goal	2	4	10	6	-26.05	10	1,196
Social Array	2	4	10	9	-42.31	8	1,160
Social Array	2	16	10	10	**-42.66**	4	0983
Social Array	2	64	10	8	-42.93	6	1,258
Social Array	2	128	10	6	-40.32	4	1,164
Goal + Array	2	8	10	6	-26.35	9	**0964**
Social LSTM	3	4	10	9	**- 40.06**	4	1,155
Goal	3	4	10	6	-26.11	3	1,143
Social Array	3	8	10	9	-39.06	8	**0944**
Goal + Array	3	8	10	10	-21.7	10	0952

12

Page 13

Social LSTM	4	4	10	10	**- 48.91**	3	**2,070**
Goal	4	4	10	8	-37.05	8	3,165
Social Array	4	8	10	8	-47.31	10	2,475
Goal + Array	4	8	10	7	-36.34	8	3,210

Social LSTM	Average	4	10	-	-46.95	-	1,530
Goal	Average	4	10	-	-28.65	-	1,640
Social Array	Media 8 (16 in video 2)		10	-	**-47.95**	-	**1,528**
Goal + Array	Average	8	10	-	-27.96	-	1,530

Conclusions

Looking at the results table we can make interesting considerations on the various ones models taken into consideration.

The variant with the target has a much higher training validation loss value than the original model, but this does not affect the result on the test. In fact the "goal" model presents a performance that is always better than the original implementation (apart from video 4). From this information we can conclude that adding the information of the pedestrian's objective it substantially helps the algorithm to predict the behavior of passers-by in the test phase.

The variant that replaces the social grid with the array has an almost validation loss value always better than the original model, and at the same time in the tests it always performs (apart from in videos 1 and 4) better than the publication model [3]. This can lead to to think that humans think more like this model than the model with the grid, when it comes to taking into account the positions of neighboring passers-by, and this therefore allows this variant to obtain better results than those of the original model.

Regarding the influence of the parameter that specifies the number of pedestrians present in the array (in the table called grid_size), we can observe the tests carried out specifically for this About the video 2. In these tests we tried to observe how the Social model behaves Arrays with very different numbers for the number of adjacent pedestrians considered. With a value of 4 the algorithm performs better than the reference model, while with a value of 16 the result in the test it gets even better. This is because in video 2, pedestrians in a frame are often less than 16, therefore with an array capable of containing the positions of other 16 pedestrians the algorithm has at its disposal all scene information and is able to more accurately predict behavior of passers-by. With very high values, such as 64 and 128, we note instead that the performances of the model worsen considerably, going so far as to be even lower than when the algorithm takes into account only 4 pedestrians. This is because in a frame of video 2 they are not there

never more than 40 passers-by, so most of the array's positions are occupied by pedestrians repeated, consequently the algorithm is faced with a lot of mainly useless information that

they confuse him. We can therefore conclude that the number of pedestrians considered is a parameter very important for the performance of the algorithm and must be calibrated based on the videos of training, testing and also must reflect the number of nearby passers-by that really one person takes into consideration in deciding his movements, so as to simulate in a manner even better as a human being would behave. In the tests of all the other videos it was set a value of 8 adjacent pedestrians considered, because considered a good compromise compared to previous motivations, even compared to the density of passers-by in videos that is not much high.

The combination of the two variants (Goal + Array model) seems to effectively combine the advantages (and disadvantages) of the two models. In fact, as the Goal variant, it has a validation training value loss much higher than other algorithms, while in tests it often performs better than the two variants and the original model (except in videos 1 and 3 in which, however, is very close to best result and in video 4). We can therefore conclude that the modification of the original model with the addition of the "goal" of each pawn and the replacement of the social grid with an array results improve algorithm performance in almost all cases.

The results of video 4 differ substantially from the rest of the results (with an error in the test 2-3 times higher than other videos) probably because in that particular video there is one Passer density extremely higher than those used for training. In video 4 in a frame there can be up to 65 people simultaneously, while in those used in training there are never more than 35, practically half. Consequently this video may not be ideal to use in combination with others because much different from that point of view. For completeness, the values have been entered in the results table tests on video 4, but there are doubts about their correctness and usefulness.

Analyzing the results of the average it could be concluded that the best algorithm is the variant of the array, followed by the original algorithm and the Goal + Social variant on an equal footing and last the variant with the objective. If, however, we exclude the results of the video in the average calculation 4 we get a completely different ranking:

1. Goal + Array average error in the test phase: 1.046
2. Goal average error in testing: 1.258
3. Social Array average error in testing: 1.317
4. Social_LSTM original average error in testing: 1.396

According to this new ranking all the implemented variants improve the performances compared to the original model, and the best model is by far the union of the variant with objective e array, which in all tests performs (except always in the video 4) better than the others or however it arrives at a very short distance from the best model.

Note how the results were obtained from models with a duration of 10 epoch training, but to obtain completely reliable results that can confirm with certainty Absolute which algorithm is best would be to use a number in the training phase of epoch around 100.

It can therefore be concluded that the objectives of this project have therefore been achieved because the variants proposals have been implemented and present superior performances to the reference model exposed in [3].

Future developments

As future developments of this project we could use the Goal + Array model not only for prediction but also for the simulation of passers in virtual environments, in which to each cell that simulates a pedestrian a starting position is given in the scene and the coordinates of the final objective, so these LSTM cells will continue to update their position at regular intervals (giving the impression of walking) along trajectories similar to those that would be traveled by beings humans in circumstances identical to those of simulation.

To get an even more accurate simulation you could make the algorithm hold account not only of nearby passers-by but also of any obstacles in the image (such as the buildings), entering this information into the array that currently contains only information about others pedestrians.

References

[1] K. Yamaguchi, AC Berg, LE Ortiz, and TL Berg. "Who are you with and where are you going?" In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pages 1345-1352. IEEE, 2011. 2, 3, 5, 6, 7, 8

[2] P. Trautman, J. Ma, RM Murray, and A. Krause. "Robot navigation in dense human crowds: the case for cooperation. ”In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 2153–2160. IEEE, 2013. 5

[3] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces, "2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 961-971.

[4] https://github.com/vvanirudh/social-lstm-tf

15