A SCALABLE LSTM BASED APPROACH FOR MULTI-PEDESTRIAN TRACKING IN SURVEILLANCE CAMERAS

by

Pratik kulkarni

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2019

Approved by:

Dr. Hamed Tabkhi

Dr. Chen Chen

Prof. Stephen Welch

ABSTRACT

PRATIK KULKARNI. A scalable LSTM based approach for Multi-pedestrian Tracking in Surveillance Cameras. (Under the direction of DR. HAMED TABKHI)

There has been an ever growing interest in leveraging state of the art deep learning techniques for tracking objects in video frames. Such works primarily focus on using appearance based models which prove not to be effective in modelling the behaviour of objects in frame sequences. Moreover, not much work has been done to explore and exploit the sequence learning properties of Long Short Term Memory(LSTM) Neural Networks for tracking objects in video sequences.

In this thesis, we propose a novel LSTM based tracker, Key-Track, which effectively learns the spatial and temporal behavior of pedestrians after analyzing movement patterns of human key-point features provided to it by the OpenPose[1]. Key-Track is trained on Single-Object dataset containing a variety of human behaviours. These sequences have been wrangled and curated from the Duke Multi-Target Multi-Camera(Duke-MTMC)[2] dataset. We further scale the model at inference time to track multiple people with effective batching. The results reported on the Duke-MTMC dataset show that the tracker is capable of maintaining a high degree of accuracy which is independent of the number of objects to be tracked in the given scene. Along with that, we try to critically analyze the scenarios complexity and classify it according to the best performing configuration of the model. Batching ,in turn, helps in the effective GPU allocation of resources yielding high FPS scores for offline tracking. The total observed size of Key-Track is under 1 megabytes which paves its way into mobile devices for the purpose of tracking in real-time.

DEDICATION

I would like to dedicate this work to my Parents, Uttara and Rajesh Kulkarni. They have been with me through the thick and thin and have always supported me in all my endeavours.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Hamed Tabkhi for giving me a wonderful opportunity on working on a cutting edge research topic. This work would not have been possible without his guidance and immense support.

A special thanks to my committee members Dr. Chen and Prof. Welch whose insights and suggestions were really useful in shaping the pathway for my thesis. Finally, a sincere thanks to my labmates Shrey Mohan and Sam rogers for their support and contributions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AOS Average Overlap Score

API Applications Programming Interface

ARIMA Autoregressive integrated moving average

CNNs Convolutional Neural Networks

CPU Central Processing Unit

DBT Detection-based Tracking

DFT Detection-free Tracking

FPGA Field programmable gate array

FPS Frames per second

GPU Graphical Processing Unit

IoU Intersection Over Union

LSTM Long-Short Term Memory

MTMC Multi-Target Multi-Camera

ONNX Open Neural Network Exchange

RNNs Recurrent Neural Networks

SIMD Single Instruction Multiple Data

CHAPTER 1: INTRODUCTION

Video cameras have proven to be an important data source for various Computer vision applications specifically for Video analytics. There has been an increasing trend in equipping cameras with smart systems that are capable of making sense of their surrounding. Amongst various subdomains of video processing, visual object tracking poses an interesting challenge in the Computer Vision domain. It is very commonly used in applications like autonomous vehicles[7][8], robot navigation and surveillance systems. Generally object trackers are used to track the movement of the object and model their movements and their behaviour to predict their actions and track in future.

Visual object tracking is broadly classified into two categories namely detection-free tracking (DFT) or detection-based tracking (DBT)[3]. In DBT, objects in the scene are detected using an Object-detection algorithm and then associated to their corresponding tracks or trajectories. Generally the object detection algorithm is run on every consecutive frame. Such techniques tend to yield high accuracy but mostly at the cost of throughput. Hence lightweight detector frameworks favour faster inference. On the other hand DFT requires a manual initialization of objects in the first frame. The objects are later localised in subsequent frames. DFT based methods are generally more throughput oriented but tend to be less accurate because its difficult to re-associate a lost track. It also cannot cope with new objects appearing. Fig. 1.1 Shows a comparison between DFT and DBT

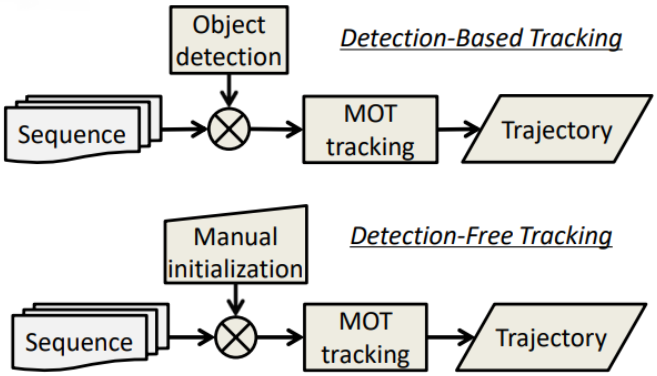

Figure 1.1: DBT vs DFT [3]

1.1 Problem Statement

Generally, tracking is performed either in an offline or online mode. Offline tracking, also known as batch processing makes use of the information in the past and future frames to handle the current tracks. However, in real-time applications offline tracking cannot be used,evidently. Hence, we perform online tracking which takes into account the frames that have been seen so far and track the object accordingly. Real-world scenarios often consists of many challenging characteristics such as illumination variation, occlusion, target deformation, and background clutter[9]. To improve robustness against these types of scenarios trackers have recently adopted many of the deep learning techniques used in other vision tasks such as object detection and classification[10] where they achieve human-like or even beyond-human accuracy in their respective domain [11], albeit at a high computational cost. When working on domains with real-time constraints, it is necessary to balance the computational load with real-time constraints. Unfortunately many of the deep trackers evaluated on the MOT benchmark[12] and other multi-object benchmarks are unable to achieve real-time throughput (FPS) and latency due to their reliance on large Convolution neural networks (CNNs)[13]. More recent works have begun to explore hybrid networks that combine CNNs with recurrent elements such as LSTM and GRU

cells to reduce computational overhead and improve performance[9][14][15].

With the growing interest in deep learning, more recent works like [16], [17],[18], and [19] use Convolution Neural Network(CNN) based appearance model to track objects. The task of tracking inherently has a strong temporal component, which is very costly to implement in CNNs. This has prompted interest in other solutions using recurrent neural networks (RNN). Some works like [20], [17],[9], and [21] use Long Short Term Memory (LSTM) but overload them with unnecessary background noise, increasing model complexity without significantly improving accuracy. These shortcomings have prompted us to explore more com-pact feature representations that prioritize the spatio-temporal characteristics of tracking targets. In this work we focus specifically on pose keypoints as a means of understanding and predicting the movement of human targets over time.

1.2 Contributions

In this work, we propose a hybrid tracking by detection framework Key-Track, that used OpenPose detection framework for human key-point detection. While another LSTM based network is built as a front end for future frame prediction in surveillance systems. The key idea here is to use only prominent human key-points instead of the whole appearance model to understand movement behaviors and patterns in humans using LSTM. We do this by training our LSTM model on different kinds of movement patterns individually(single object) and then scale it to multiple objects at run time with the help of batching. Our results show that our model scales well as we only use key-point information for predictions, improving the scalability of the system as the number of tracking targets increases. Furthermore the LSTM layer can be trained on single target examples, but evaluated on multi-object scenarios of varying complexity simply by batching.

We achieve this training and evaluation paradigm via the following contributions:

- Training on single object and testing on multiple objects by scaling the LSTM with effective batching
- Curation of a extensible single and multi object training/testing dataset by introducing a Data wrangling technique
- Lightweight multi-object LSTM-based tracker module less than 1 MB. capable for deployment over edge
- Model trainable at different FPS depending on the input stream
- Creating scalable training/inference configuration knobs based on the scene complexity and real-time requirements

CHAPTER 2: RELATED WORK

Recurrent neural networks handle data that showcase temporal behaviour, very well. LSTMs[7], specifically, are able to learn long term dependencies. Here we survey different works that use LSTMs to leverage the pattern recognition properties and other object tracking paradigms.

2.1 Pattern Recognition

Recent works like [22] and [23] have demonstrated the effectiveness of LSTMs on understanding patterns from unstructured data. Authors in [24] employed LSTMs for understanding texts and putting it to semantic context. On similar lines in [25] the authors improved the character recognition properties of CNNs by adding a recurrent layer in their model structure. Similarly in [26] LSTMs were used to predict end points in the Chinese language. Authors in [18] used LSTMs to model human motion detection to predict time dependent motion representation for human poses. All these aforementioned developments signify the importance to explore the role of LSTMs in understanding human behavior and leverage its pattern recognition properties.

2.2 Object Tracking

Initial works, such as [27], were heavily dependent on the Kalman Filters to be able to stabilise predictions of the movement of the object. However they do not prove to be much reliable due to the fact that they cannot account for any historic patterns in their predictions. Other works like [28] incorporated LSTM for applications such as object tracking like have exhibited promising results. Another work [29], is effective in understanding the underlying patterns in data. Recent, research like [16], [9] employed RNNs, specifically LSTMs, for the purpose of tracking. [9] uses the YOLOv1 object

detection framework for generating the rectangular bounding boxes. This framework also generates a feature vector of the image with 4096 length. These 4k features are appended to the bounding box dimensions and passed to an LSTM. This layer is responsible for predicting the position of the bounding in future frames. This feature vector provides ample context about the object beyond its mere position and scale, however it also contains excessive noisy features that increase computational overhead. Not only does this affecet the performance but also degrades the tracker accuracy. Another work[21] makes use of a 500-dimensional feature vector with VGG-16 as the feature generator. However it renders an FPS of merely. Another method uses an on-line object tracking strategy[12] that implements LSTM. This technique is inspired from Bayesian Filtering idea which makes data association, state updates and initiation and termination of tracks. The authors end up with a model that learns to track and gives good real time performance on the expense of accuracy. Furthermore, another detection based tracking approach [20]uses Faster RCNN in the backend as an object detector along with a Vanilla RNN for tacking the targets. The Vanilla RNN is less capable of addressing the long-term dependencies in lengthy video sequences. Further, the work focuses on a single object tracking mechanism rather than a Multi-object

CHAPTER 3: BACKGROUND

3.1 Tracking as a time series problem

Object tracking can generally be termed as a time series/sequence problem. A time series can be defined as a series of data points sequenced out in space in a timely order. The time series is then analysed to find the trends and behaviour of data points for the purpose of forecasting and estimation in future. The following example shows an Autoregressive integrated moving average (ARIMA) model used to forecast Australian beer sales.

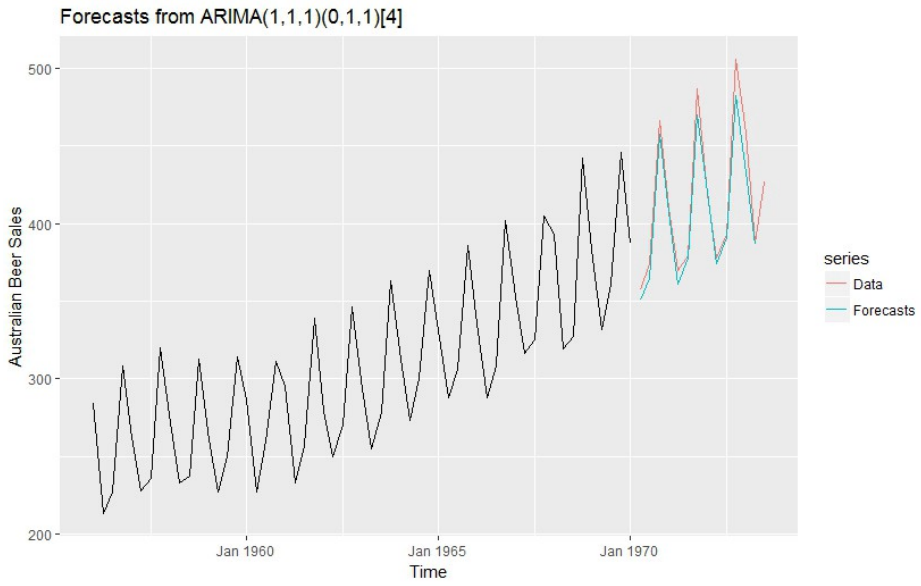

Figure 3.1: ARIMA model forecasting Australian beer sales [4]

Generally when speaking of statistics, quantitative finance, econometrics etc the prime aim of time series analysis is forecasting. While speaking of signal processing and communication engineering it is used for estimation and signal detection. Further, time series forecasting can be classified into various types. For example, a model pre-

dicting immediate next time step is a One-step model while a model predicting more than one time-step in future is a Multi-step model. Depending on how observations of data points are made a contiguous model is uniform over time while on the other hand, a dis-contiguous model makes observations not uniform over time. A structured time series is characterized by a systematic patterns which are time-dependent while an Unstructured series does not showcase a systematic time-dependent trends. Some common examples of time series include sunspot count, closing and opening values of stock markets, heights of tides of the sea etc.

Similarly, object tracking involves a set of data points represented terms of pixel values that are laid out in space, in a timely order. We aim to exploit the temporal locality and estimate the future set of data points. This corresponds to where the particular object will be in the future frames given its past trajectory. This technique helps for trajectory analysis and better predictive analysis. As discussed earlier, the our model performs One-step prediction and Multi-Step prediction as well while the time series is also characterized by structured and un-structured series.

3.2 Neural Networks

Neural networks can be defined as algorithms that are based roughly on the working of a human brain. These networks are generally designed to recognize patterns in the input and make predictions accordingly. They translate the sensory data using a kind of machine perception, further classifying or clustering the input. The patterns and trends recognized are simple numbers present in the form of vectors. Hence to learn the patterns and representation of input data, the inputs must be translated to these numerical vectors. This includes any sort of data ranging from sound / signals, images, text or time series. The neural networks is comprised of multiple layers that are made of nodes(neurons). These neurons are responsible for propagating inputs through to the output layer. The computations take place at neurons. [30]

Further a special type of Neural networks known as Convolutional neural networks

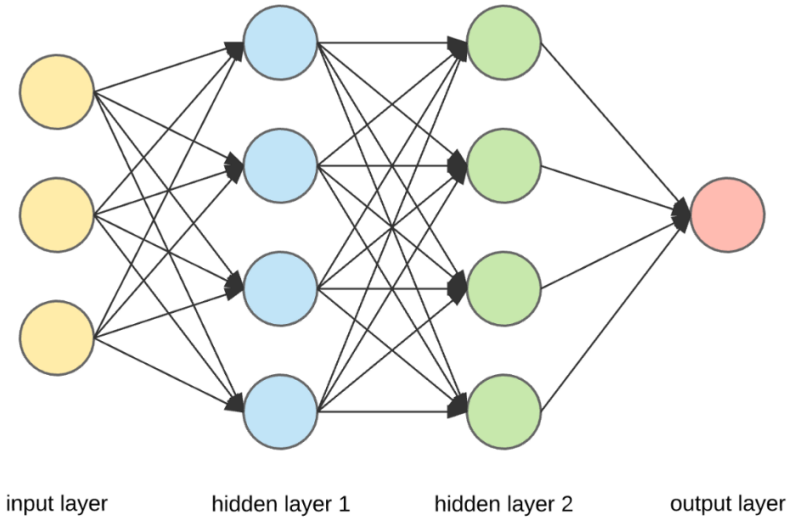

Figure 3.2: A neural network

are commonly used for Computer Vision tasks like object detection, object classification, object segmentation etc. A CNN has the following main components in its pipeline, Convolution, Sub-sampling, Activation and Fully Connected. Initially the image is convoluted with a kernel which is responsible for feature extraction. Here different features like edges, shapes, scales are extracted. Later the features are shrunk using sampling also known as pooling. This helps is filtering out the noise and redundant features. It also reduces the dimensionality of the feature map but still maintaining the important information. The activations helps the network learn non-linear functions. A variety of activations are used depending on the task. The final component is a fully connected layer. The feature vectors from the previous layers are directly mapped to this layer. It can also be viewed as decoder layer which yields the final output. While CNNs have proven one of the best algorithms for many Computer vision problem, they fail to address temporal domain, if present in the data.

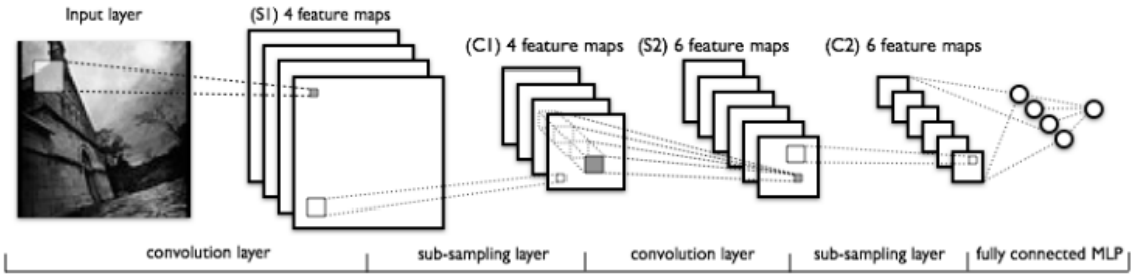

Figure 3.3: A Convolutional Neural network

3.3 Recurrent Neural Networks

Recurrent nets are artificial neural network algorithms used specifically for processing data that is characterized by a time dimension. Such data typically includes sound, time series (sensor) data or written natural language. The main idea is to predict or output a certain value based on its past behaviors. These networks differ from the regular neural networks because they include a feedback loop. This means that the output from the previous time step is fed back to the network to influence the outcome of next step or next subsequent steps. For example, if we would like to classify what kind of event is happening at certain point in a movie, we tend to think of the actions that took place in near past that would influence the outcome and then predict the next step. A typical unrolled RNN[5] is show here.

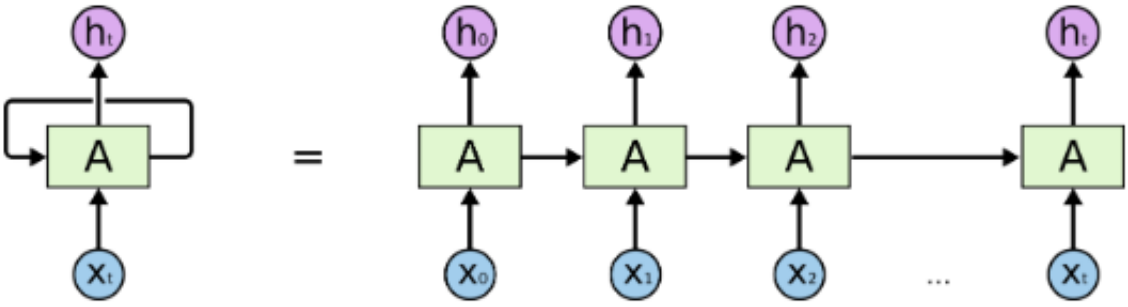

Figure 3.4: An Unrolled Recurrent Neural network[5]

The promise of RNNs is that they are able to associate previously gathered information to the current prediction, like using the earlier video frames should influence the understanding and behaviour of the current frame. Sometimes, we need to look at the very recent information for the predicting the current task. For example, a language model that predicts the immediate next word based on the previous one. Thus, If we want to predict the last word in âthe clouds are in the sky,â we wonât need any further context as the next word should be sky. In such cases, where the gap between the relevant information and the place that itâs needed is small, RNNs can learn to use the past information[5]. However, as that gap grows, RNNs become less capable of connecting the information. Thus to address such long-term dependencies, we use a special type or RNN known as Long short term memory â LSTM.

3.3.1 Long Short-Term Memory

Long Short Term Memory networks - LSTMs are capable of learning long-term dependencies. Introduced by Hochreiter Schmidhuber (1997)[7], they perform well on a variety of problems as mentioned in 2, LSTMs are designed to address long-term dependency problem by retaining information for long periods of time. All RNNs have a form of a chain of repeating nodes of neural network. In the basic RNNs, this repeating nodes have a simple structure, such as a single *tanh* layer. While the LSTM also exhibit a chain like structure but with different internals. Every node has 4 neural network layers instead of a single layer. Fig 3.5 shows the repeating nodes in LSTM.

The key in LSTMS is the cell state indicated by the horizontal line in the top of diagram 3.6. It is like a belt running through the LSTM node with linear interactions. The LSTMs have the ability to remove or add information to the cell state using a structure known as gates. The gates are composed of a sigmoid neural net layer and a pointwise multiplication opration as shown in Fig. 3.7

LSTM has 3 such gates, to protect and control the state of the cell. An LSTM

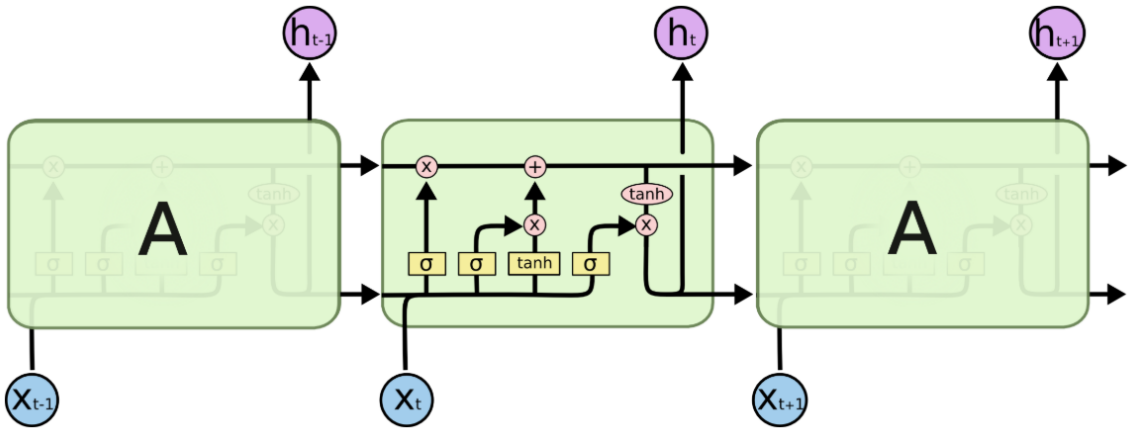

Figure 3.5: Repeating modules in LSTM[5]

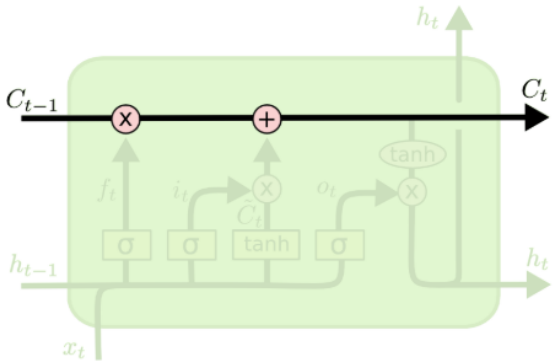

Figure 3.6: LSTM cell[5]

has three of these gates, to protect and control the cell state. The initial step for the LSTM is to decide on what information to throw away from the cell state. The decision is made by a sigmoid layer called the âforget gate layer.â It takes into account the last steps hidden state h_{t1} and x_t, and outputs a number between 0 and 1 for every number in the cell state C_{t1}. Here a 1 means, âcompletely keep thisâ while 0 represents âcompletely get rid of this.â The next step is to decide which new information do we want to retain. A sigmoid layer decides which values to update. Later a *tanh* layer creates a vector of that could be added to the cell state. The third step involves updating the old state, C_{t1}, into a new cell state C_t. The old state is multiplied by f_t, to forget the information that was supposed to be discarded. Then we add $i_t C_t$, which is the new value. Finally we then output based on our cell state. First a sigmoid layer

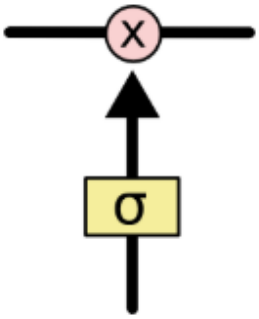

Figure 3.7: Gate with a sigmoid layer[5]

decides what to output and then a *tanh* to push values in between -1 to 1 and further multiply it by output of sigmoid gate to only output the information we decided to.

The equations for input gate i_t, output gate o_t, forget gate f_t and final state h_t defined as following:

$$i_t = \sigma(W_{x_i x_t} + W_{h_i h_{t-1}} + b_i) \tag{3.1}$$

$$o_t = \sigma(W_{x_0 x_t} + W_{h_0 h_{t-1}} + b_o) \tag{3.2}$$

$$f_t = \sigma(W_{x_f x_t} + W_{h_f h_{t-1}} + b_f) \tag{3.3}$$

$$h_t = o_t * tanh(c_t) \tag{3.4}$$

CHAPTER 4: APPROACH:Key-Track

In this work we propose a hybrid Multi-object tracking framework named Key-Track. The name Key-Track is derived from Keypoints generated by OpenPose coupled with a tracker system. The Fig. 5.5 shows the system architecture which is a multi stage pipeline aimed at building an end-to-end solution. The system relies on a preprocessing block responsible for generating unique training data. The tracker here is characterized by the key idea of training the framework on behavior of one object/person in the dataset and scaling it up at runtime to run an inference on Multiple objects with effective batching. We make use of keypoints pertaining to the object (shortened feature vector), removing unnecessary background features. This allows us to keep the tracker size less than a megabyte.

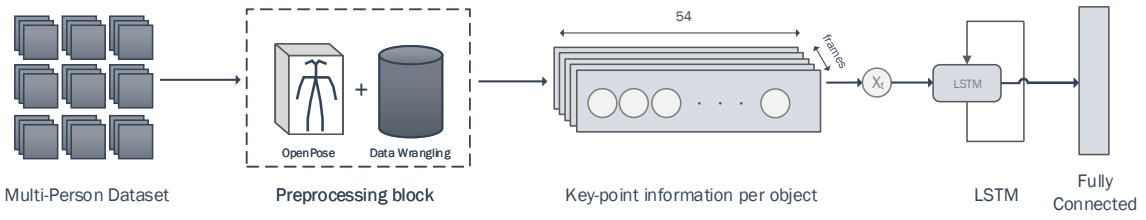

Figure 4.1: Overview of the System architecture pipeline depicting blocks and flow of data

4.1 Architecture overview

The detection based tracker starts with detections generated in the form of a feature vector of keypoints of a person by OpenPose[3]. The pre-processing block is a crucial first stage that isolates and re-identifies the keypoints associated to individual target groundtruth. The keypoints consist of the the X and Y coordinates along with the 18 different body parts. These are then aggregated across frames and fed to an LSTM

network. This network further uses these keypoints to map them to a fully connected layer. This FC layer is responsible for a regressed bounding box values (x,y,w,h) Main advantage of using an LSTM based on keypoint detections is that the keypoints suffice with plenty of context of the object to understand its motion. Similar works like ROLO and Re3[12] rely on deep features to represent the context of an object yielding bigger model sizes (100MB+) and higher training times. Our trained model capable of producing comparable performance with a small model size(<1MB).

4.2 Multi-Person dataset

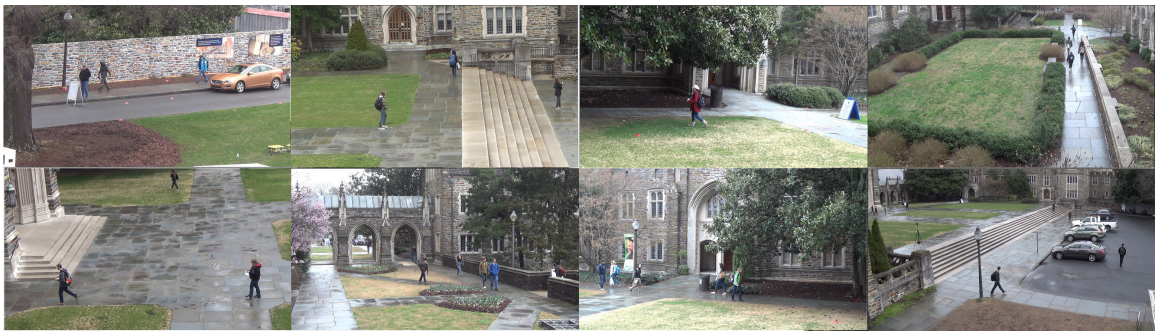

Figure 4.2: 8 different camera angles as obtained from the Duke dataset

The data-set we worked with is introduced by Ergys et al. [26] which is a large scale dataset, possibly one of the largest dataset. The Duke MTMC dataset is a Multi-Target Multi-Camera dataset which consists of 8 Camera angles with different environment settings. It has a massive number of 2834 different people annotated across those cameras. The total video comprises of more than an hour long of footage from 8 cameras recorded at around 60 frames per second FPS inside the campus of the Duke University. The footage is shot at a resolution of 1080p with more than 2 million frames involving a high number a occlusion scenarios (around 1800) and partially generated ground truth bounding boxes. The pedestrians generally carry day-to-day accessories like umbrellas and bags etc. Here, our qualitative analysis has shown different types of motion exhibited by the objects that include linear, non-linear behaviors, abrupt motions, occlusion, slow movements which pose an interesting

challenge to the tracking algorithm. This provides a real world scenarios that would be perfect for testing the robustness and efficiency of model.

4.3 Pre-processing block

4.3.1 OpenPose

A recurrent neural network, as mentioned in section 3.3, works well with data that exhibits a sequential/timely pattern. However, it expects a data in a suitable format to be processed and worked on. A feature generator that generates a feature vector of the original data is coupled with a recurrent neural net. Here, OpenPose serves as a feature detector responsible to capture the features of the object. In case of pedestrian tracking, Human pose estimation frameworks can be effectively used. One such widely used network is OpenPose[1] that provides accurate and fast pose estimates which can scale irrespective of total number of people in the scene, thus inherently scalable and capable of running at high speed in real time. It is exclusively trained on Humans. Hence it serves as a perfect backbone as it is capable of high quality detection at different resolutions as well.

4.3.1.1 OpenPose Architecture

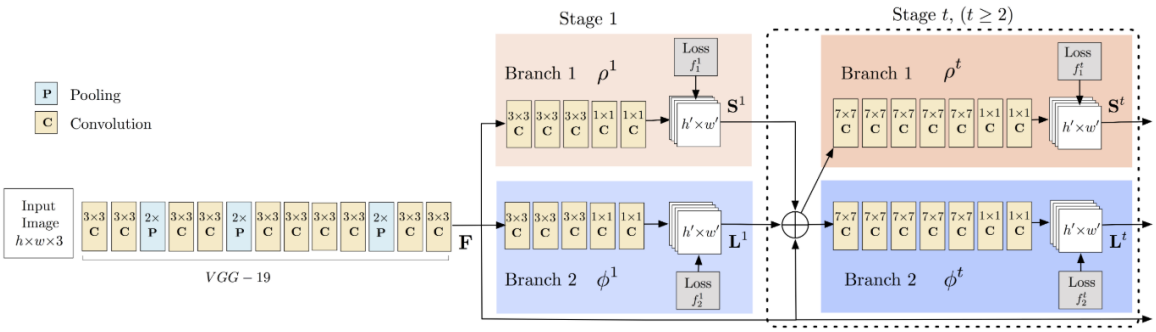

Figure 4.3: Multi-Person Pose Estimation model architecture[6]

The OpenPose framework is built on VGG-19 classification network. Here, the first 10 layers of VGGnet are used to create a feature map for the input image. As seen in fig 4.3, a 2-branch multi-stage CNN is used. Here, the first branch predicts a cluster

of 2-dimensional Confidence maps of body part locations. A Confidence Map is a grayscale image which has a high value at locations where the likelihood of a certain body part is high[6]. The 2nd branch predicts a set of 2-dimensional vector fields of Part Affinities (PAF) which represent an association between the keypoints.

(a) MPII (b) COCO (c) COCO+Foot

Figure 4.4: OpenPose detection Examples[1]

OpenPose detection are shown in Fig. 4.4 As seen from the figure, OpenPose returns a skeletal representation via keypoints. These individual keypoints correspond to certain parts of the body, depending on the model(MPII, COCO, COCO+foot) used.

4.3.2 Data wrangling

The DukeMTMC dataset as mentioned earlier consists of multiple pedestrians at any particular frame throughout the scene. The whole promise of this work is based on training the network with multiple behavior of a single object and scale it to be able to infer on multiple object/pedestrian. However as the Dataset itself contains multi-object sequences we employ a unique data wrangling technique. This lets us isolate individuals and their respective keypoints within camera sequences. The way this is done is we first localise objects present in the scene and associate its keypoints that we obtain from OpenPose to its corresponding groundtruth. This data wrangling technique is able to generate around 2700 pedestrian sequences (the number of

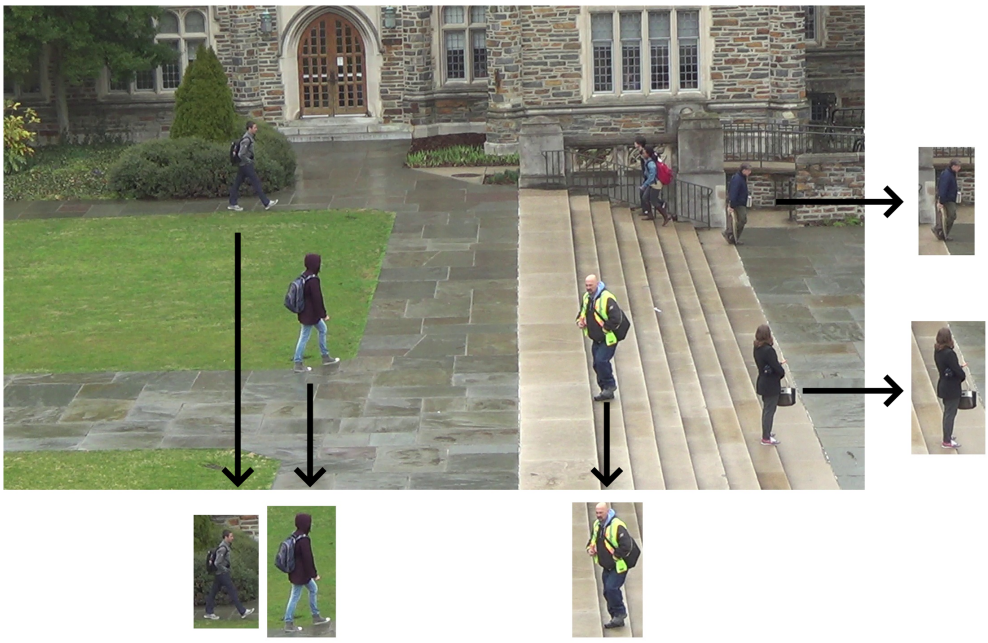

Figure 4.5: Data wrangling used to localize and isolate objects for training

pedestrians present in the dataset), thus, helping the model to generalise better.

For the purpose of testing and evaluation on the dataset, we choose an arbitrary starting frame and an end frame. This removes any bias when testing as we select a sequence at random from the immense test set present. This technique effectively yields a continuous sequence. All the new objects entering the scene are assigned its own index (exclusive ID) in the 3D input tensor with its respective keypoint-feature vector. This serves as input to the LSTM network. Such kind of configuration is necessary to retain its movement properties and trajectories in the subsequent frames. In case an object moves out of the scene, the corresponding row is filled with zeros to denote no object present. Similarly the Tensor is initially empty to be able to accommodate any (upper cap) number of objects. This way we maintain exclusivity for an arbitrary number of objects that enter or exit throughout the sequences.

4.4 Key-Track implementation

The key-track network is based on detections received from the OpenPose framework. Hence, we leverage the detections for every object in each frame. Here, a feature vector consisting of 54-data points for each pedestrian is fed as the primary input to the LSTM network. The keypoints represented here are the keypoint values in terms of pixels along with confidence values. The format is (x1,y1,c1,...,x18,y18,c18) where x and y is the location of the keypoint along with the confidence. This 54 dimensional vector is inputted to the network for every time step. The fig shows that the network receives the vector for 3 time steps. The crucial step further is to gather the prediction at final step and map it to the fully connected layer.

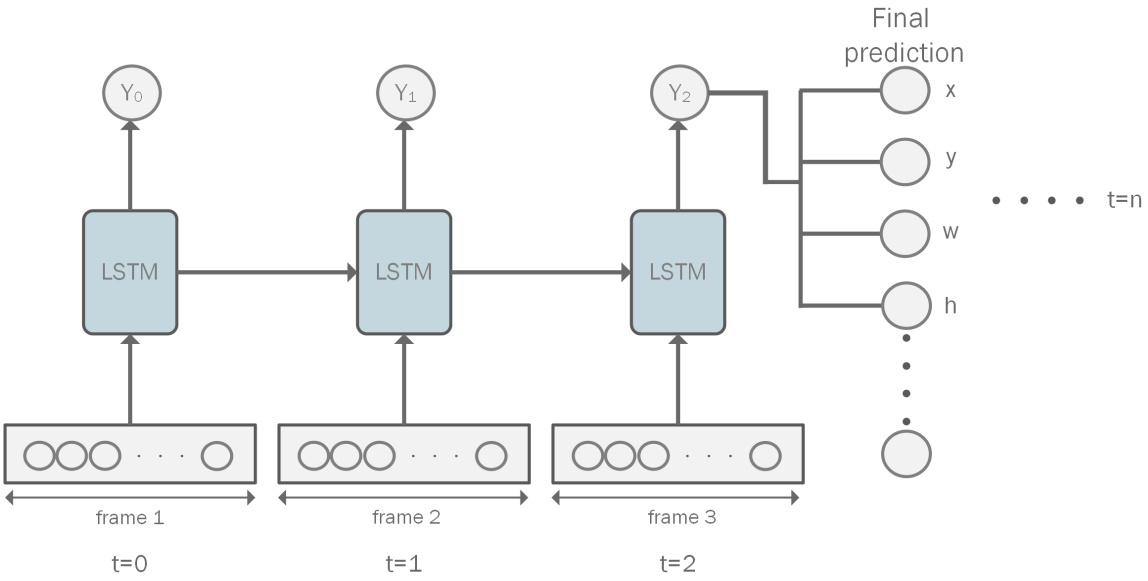

Figure 4.6: Training instance for single iteration

The fully connected layer works like a decoder circuit. The output predicted by the LSTM is regression based. Here the basic fully connected layer consists of 4 neurons responsible to predict x,y,w,h where x,y is the centroid and w and h denote the width and height of the bounding box for that person. However, this FC is responsible for the predicting the position of the object in immediate next frame. For a high FPS

video predicting just the immediate frame is equivalent to predicting the position at an extremely short amount of time in future. Hence, to facilitate the frame prediction for even more future frames, we try to employ a technique of predicting the future location of the object in spatial domain. This is synonymous to transforming the time domain to the space domain. The key here is to have the size of FC equal to the number of future frames we wish to predict the location times 4; where 4 indicates x,y,w,h

$$FC_{neurons} = future frames * 4 \tag{4.1}$$

Our results show that training on the centroid of the object makes the LSTM less susceptible to abrupt bounding box changes. This helps LSTM to learn better. The network is trained on a loss function commonly used for regression problems. The equation for Mean Square Error is shown below

$$L_{MSE} = 1/n \sum_{i=1}^{n} ||B_{target} - B_{pred}||^2 \tag{4.2}$$

where,

$$B_{target} = B_{pred} = x, y, w, h \tag{4.3}$$

4.5 Scalability of the network

Scalability is defined as the ability of a product to function at its peak performance when changed in size or volume. Generally, the scaling is done to cater multiple requirements at a higher volume but at a same or very less compromise in terms of accuracy or speed. Similarly, in our context we train our model on a single object/pedestrian and scale the network to be able to track multiple pedestrians at run time. We do this by batching the detections of individual pedestrians. Batching corresponds to a Single Instruction Multiple Thread operation. This type of execution model is used in parallel computing environment with multiple threads executing the

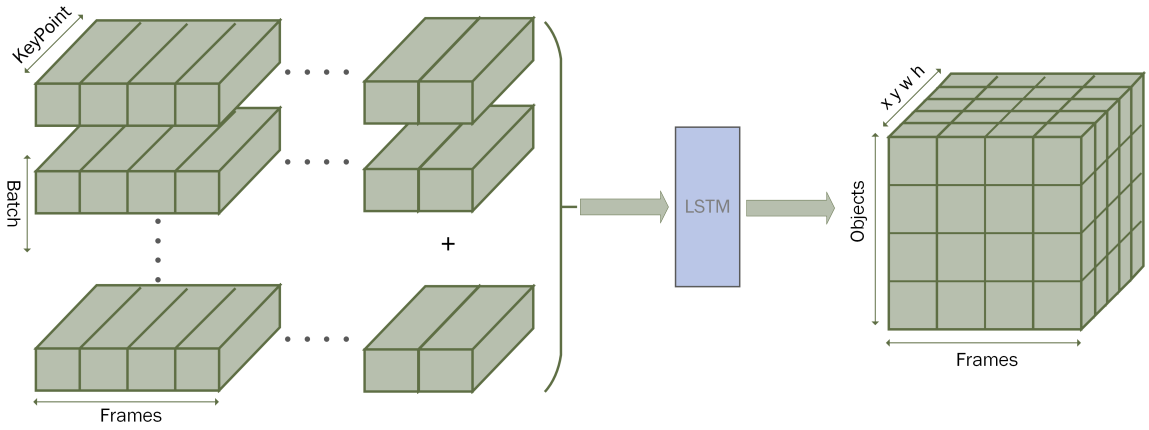

Figure 4.7: The input tensor fed during inference to demonstrate scalability

same command, in parallel. Here, Every unique detection is assigned an index/ data lane. Each data lane/index in the batch share the same weights and operations. This way we can dynamically change adjust the batch size depending on the number of objects to be tracked.

CHAPTER 5: EXPERIMENTAL RESULTS

As mentioned in the earlier chapter, we curated an altogether new dataset with 64 video sequences using our Data Wrangling technique explained in section 4.2. A total of around 41,268 frames were randomly picked for the purpose of training from a pool of more than 2 million frames present in the original dataset. We publish our results on an additional of 14400 frames for single-object tracking. This is done to specifically to obtain right model configurations to be further used for Multi-object tracking. Furthermore, 3600 frames are used for evaluating our results on multi-object tracking.

5.1 Environmental setup

5.1.1 Software Setup

To facilitate and incorporate the recent features provided by various frameworks, we use the latest versions as mentioned below. We make use of the Pandas 0.24 framework for the data wrangling and curation technique, while PyTorch 1.0 has been used to build, train and test the LSTM models. Further Numpy is used for fast vector computations and OpenCV 4.0 for image pre and post processing techniques. Lastly, we make use of the ONNX library to port the models to a Caffe2 version to be able to run for optimized efficient inference for Real-time scenarios.

5.1.2 Hardware Setup

We train our LSTM model on the NVIDIA Titan V GPU with 12GB memory. The test is carried out on the same GPU and an Intel Xeon CPU with 20 physical cores(40 multi-threaded) for a brief comparison.

5.2 Evaluation metrics

The object tracking community employs various different metrics according to the use case, suitability and the information provided by the tracker in the applications context. Some of the commonly used metrics include Center error, Tracking length, Region overlap (Intersection over union) etc. The center prediction error[31] measures the difference between the targetâs predicted center as predicted by the tracker and the ground-truth center. This metric is popular due to the minimal annotation effort required i.e only a single point per object per frame. However a drawback of this measure is its susceptibility subjective annotation i.e. what exactly is the center of the object. This measure also does not take into account the size of the object. Equation 5.1 denotes the delta calculated between the ground truth and prediction while Equation 5.2 denotes the summarized average error over N frames.

$$\delta_t = ||x_t^G - x_t^T|| \tag{5.1}$$

$$\Delta_\mu(\Lambda^G, \Lambda^T) = 1/N \sum_{t=1}^{N} \delta_t \tag{5.2}$$

Another measure used in this context is tracking length. It measures the number of frames successfully tracked by the tracker from its initiation to the very first failure. It makes use of center error, however, to keep a track of successful tracks. An error more than a threshold is considered as a failure to track. While it overcomes the center error drawbacks, it has some other significant drawbacks too. It only uses the video sequence up to the first failure. If in some case, the initiation is bad due to the target is not visible well, it would straight away fail even before beginning to measure the accuracy for the rest of the video. Another widely used metric is the Region Overlap or the average overlap score. The AOS can be calculated using the Intersection over Union method. It defines a ratio of the amount of intersection overlap between the ground truth bounding box and the predicted bounding box of

the object to the union between the ground truth and prediction. Generally an IOU score of 50% or more is considered as a successful detection denoting a True positive. Further the Intersection over Union score is then averaged for the number of frames the predictions have been made. This yields in the AOS.

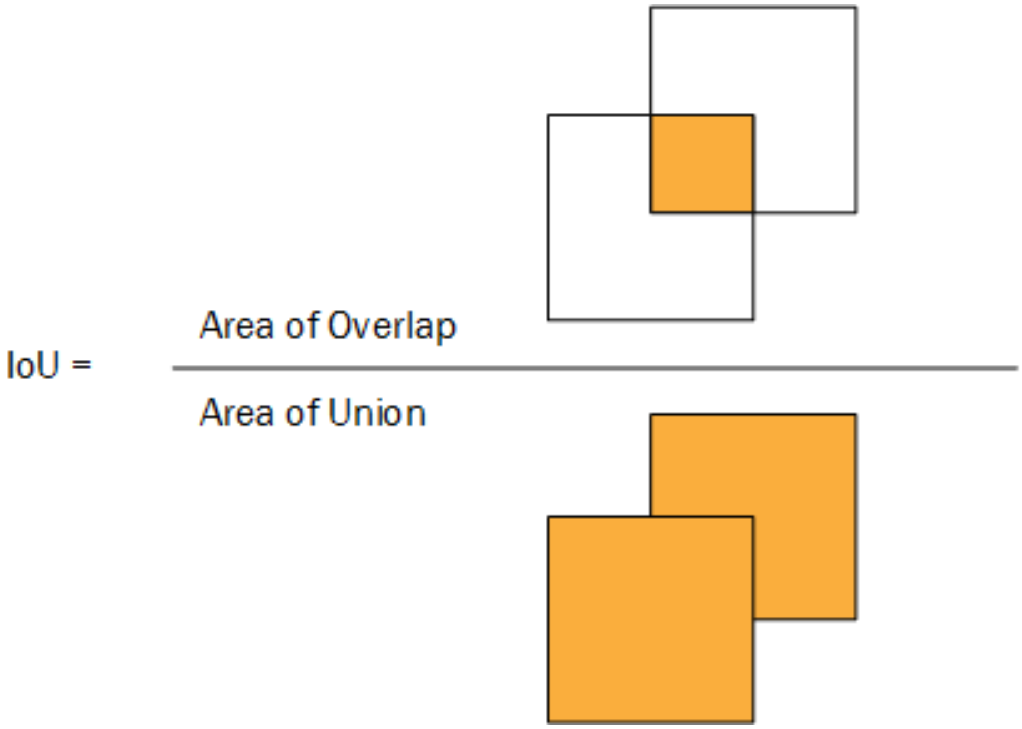

Figure 5.1: Intersection over Union

5.3 Quantitative results

5.3.1 Single-Object tracking results

We report our initial results on a diverse set of tracking scenarios on multiple single object sequences. Here, using our data wrangling technique we curate 24 unique single object sequences. To avoid any sort of biases and generalise our predictions, we choose these sequences from all the 8 available cameras. To evaluate which LSTM configuration performs the best, we measure the AOS score for 4 different timesteps. The time-step measure is crucial as it denotes the number of previous frames we take into account for the prediction. It essentially determines how good or bad the

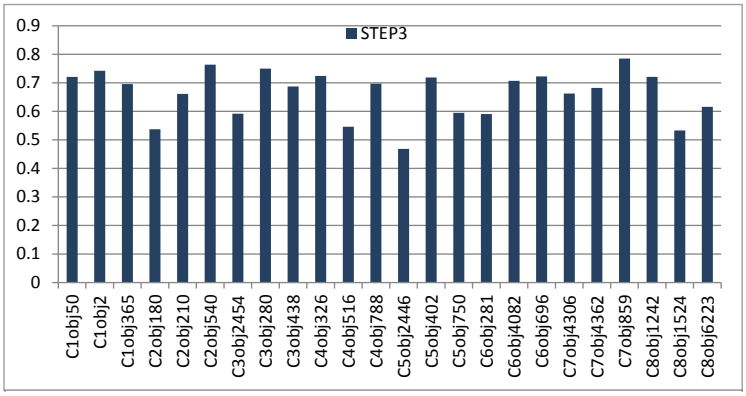

Figure 5.2: AOS for time step 3

prediction in the next or next few frames will be.

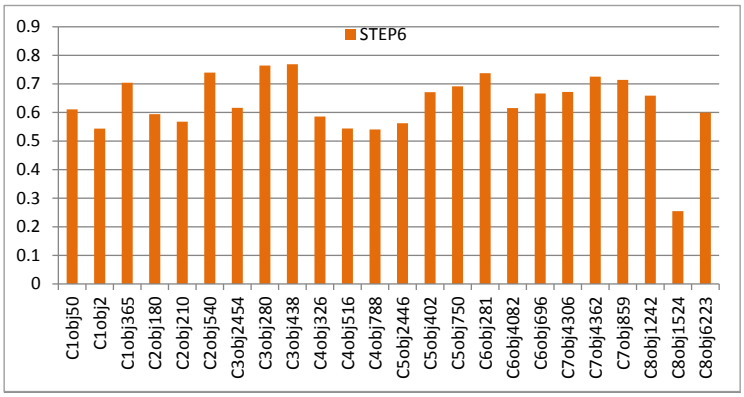

Figure 5.3: AOS for time step 6

The aforementioned graphs show various AOS scores. Here, 'C' denotes the camera and 'obj' denotes the object/pedestrian ID. Here the 3-step input performed best with a highest overlap of 78%. Based on these findings we cumulatively gathered an average over all the videos to find the best time step, quantitatively. Fig 5.6 shows a brief comparison as to which step performs the best for the entire test set. We found out that there was a weak correlation between the length of the input time series and the resulting accuracy. To further analyse why this was the case we *qualitatively* analyse the characteristics of the data set and report it in section 5.4.

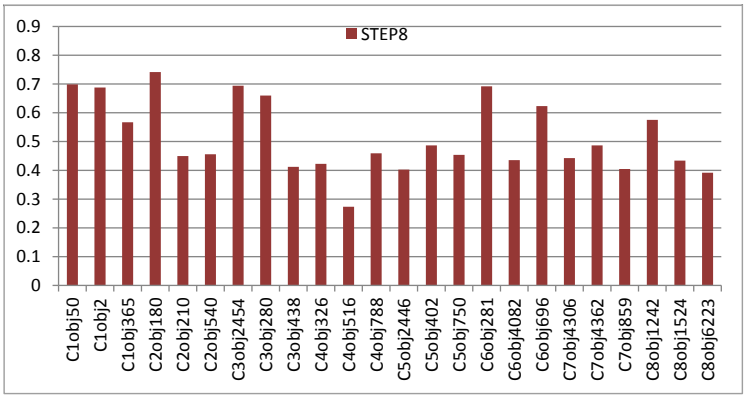

Figure 5.4: AOS for time step 8

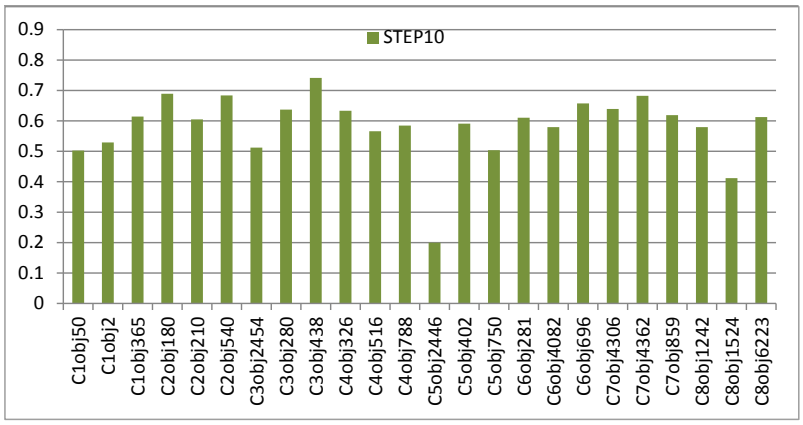

Figure 5.5: AOS for time step 10

5.3.2 Best model configuration

We report our initial results on a diverse set of tracking scenarios on multiple single object sequences. Here, using our data wrangling technique we curate 24 unique single object sequences. To avoid any sort of biases and generalise our predictions, we choose these sequences from all the 8 available cameras. To evaluate which LSTM configuration performs the best, we measure the AOS score for 4 different timesteps. The time-step measure is crucial as it denotes the number of previous frames we want to take into account for the prediction. It essentially determines how good or bad the prediction in the next or next few frames will be.

Further, we try to record a unique metric where Given a certain Average Overlap

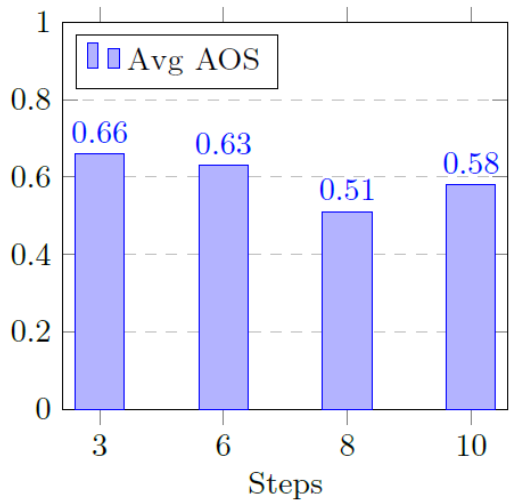

Figure 5.6: Step comparison with Average AOS

score threshold, we measure how many testing sequences showcased the Average AOS equal to or even better than the threshold score. This gives an intuition as to how well the tracker performed for an unseen amount of data. As we see from Fig. 5.7, the tracker maintained a high degree of accuracy for the entire test set till high threshold values. The flatter the curve is, the better is the accuracy for the entire test set. The results show that the tracker is able to maintain a 60% and higher average overlap score for more than 17 videos (71% of the dataset)

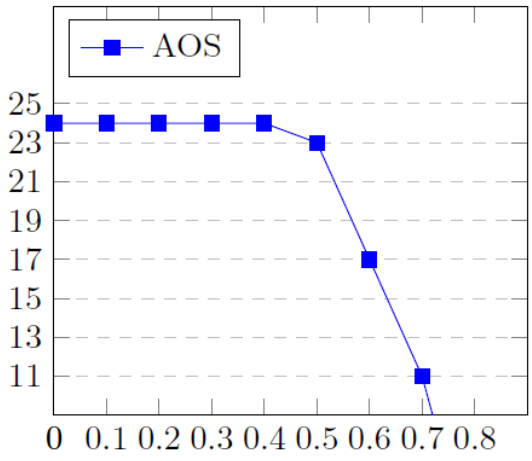

Figure 5.7: Number of sequences compared to the AOS threshold

5.3.3 Multi-object tracking results

After being able to find the best working model configuration by testing on single-object sequences, we further report our results on Multi-object sequences. We try to achieve a comprehensive comparison by using a concatenation technique. We curate a new single person test bed with 32 different people and concatenating the sequences to form a tensor. This technique allows us to demonstrate the effect of scaling for arbitrary number of objects and the average AOS for the sequences. We also report the performance numbers for scaling on edge and server, both.

First, to assess the behavior exhibited by different objects in the dataset, we analyse the videos and classify them based on their characteristics. Table 5.1 shows our analysis of videos. The abnormal motion class depicts scenarios with non-linear tracks and sudden motion changes. Such scenarios are hard to handle for smaller time steps. Hence they are naturally favoured by a longer time steps. As seen from the table time step 6 and 8 favor this class. Using higher timesteps we tend to exploit more of temporal history of the object resulting in better predictions. Similarly scale changes include cases like moving closer or further away from the camera or also deformations in the bounding box. Here an input of step 3 performs well. This way by adjusting the model configuration rightly, we can adapt to various scenario complexities.

Table 5.1: Preferred Input Time Series Length Based on Video Characteristics

Motion type	Slight Abnormal Motion	Abnormal Motion	Scale Change	Linear
Time-steps	3	6,8	3	6
Number of Sequences	13	4	3	4

- Effect of scaling on AOS:

The number of objects in the scene has almost a negligible effect on the tracker. Since all the objects in the scene are tracked individually and batched as inputs, the accuracy is stable over arbitrary number of objects in the scene.

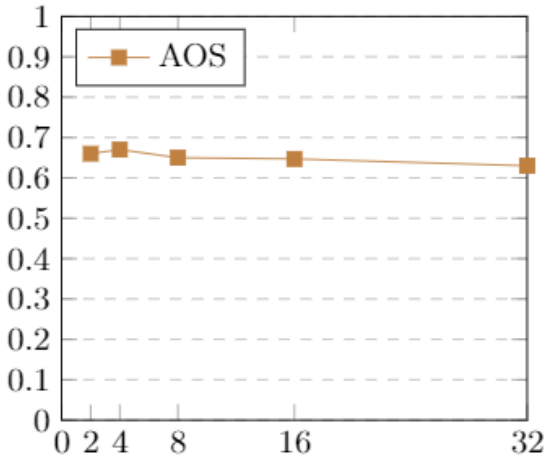

Figure 5.8: Effect of scaling on AOS

5.3.3.1 Performance results

- Effect of scaling on Inference time for an EDGE server:

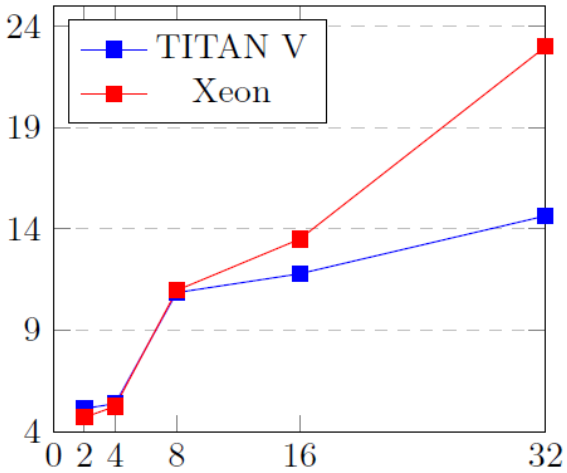

Figure 5.9: Effect of scaling on Inference time(s)

As shown in Fig. 5.9 , we vary the batch size corresponding to the number of pedestrians in the scene. We consider a sequence of 2343 frames here. It can be seen from the graph that an exponential increase in batch size has a linear effect on inference time making the model robust to any scale. Here, for lower batch size, the inference is faster for CPU. As soon as we switch to higher batch size (more pedestrians

to track), the GPU performs better owing to its ability to handle massively parallel applications. As mentioned earlier, the EDGE server is local server with a TITAN V and Intel Xeon.

- Effect of scaling on Inference time for an EDGE node:

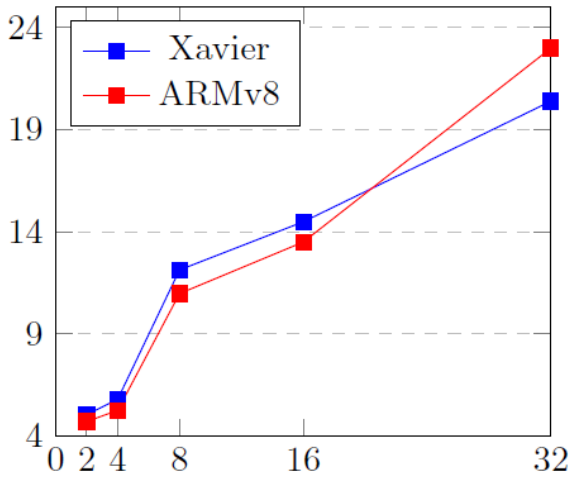

Figure 5.10: Effect of scaling on Inference time(s)

Moving on to the EDGE node, a similar behavior is observed as compared to the EDGE server. The NVIDIA Jetson Xavier is an embedded GPU platform with a a Xavier GPU and ARMv8 CPU. This GPU allows us to test the scalability performance when the tracker would be deployed online. As seen from Fig. 5.10, the inference time for lower batch size is less for the ARMv8. Naturally, as we move to higher, the Xavier GPU performs better, thus exploiting the inherent ability of processing parallel applications faster

- Throughput in terms of FPS:

Here 5.11 shows the throughput for different GPU devices. The throughput is defined as the ability of the network to process a specific amount of data per unit time. The measure we use here is Frames per second. The Frame rate recorded is for offline tracking. As seen from the graph every Device has its own advantage and

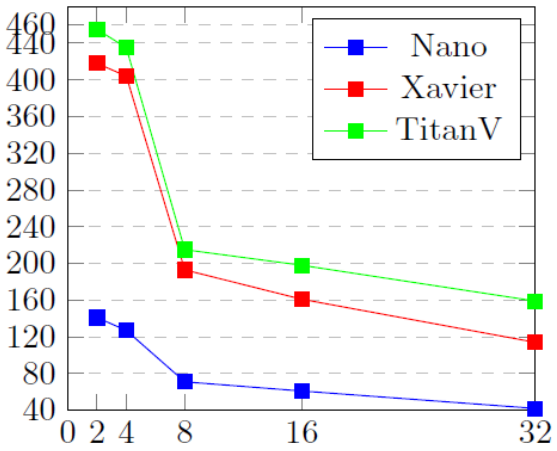

Figure 5.11: Frame rate vs Batch Size

drawback. The Titan V, which is the Server yields the highest FPS for every batch. However it operates at a mammoth 250W power. As we move to the Edge GPU, the NVIDIA Jetson Xavier, we get a comparable performance however at a peak power usage of just 30W. Further, the NVIDIA Jetson Nano, deliver the lowest throughput however at a very low peak power of 10W. Thus, the Xavier being the perfect choice for edge computing can be used as an edge device.

5.4 Qualitative results

The qualitative results indicate the robustness and performance of the system from a human readable perspective. The following figures show 6 frames from a chosen sequence. The Blue box is the ground truth while the red is the prediction. This sequence contains linear and slightly abnormal motion. We can also see a case of long term occlusion along with a person standing for long. Here, objects randomly enter from any direction in the scene. To accommodate this the tracker requires a warm-up time of *timestep* number of frames to be able to initiate tracking. Also, all the objects have comparable and high Overlap with respect to the groundtruth.

Figure 5.12: Continuous sequences from the test set

CHAPTER 6: CONCLUSIONS

6.1 Summary of Results

This work presents a hybrid approach for Multi-object tracking for Surveillance cameras without compromising much on throughput and maintaining the accuracy of detections irrespective of the number of objects to be tracked. This really showcases the pattern learning property of LSTM for object tracking. The usage of object specific keypoints allows us to get rid of unnecessary background noise and essentially yields in a tracker system which is super lightweight. We also introduce a unique idea of training on single object and testing on multiple objects by effective batching. The scalability reports show that the network scales well by maintaining throughput. The qualitative analysis on other hand gives us insights on the scenes in the video. Thus Sequences with varying complexities were successfully tracked using different timesteps.

6.2 Future scope

We thoroughly evaluated the scenes and built a tracker module based on every scenario possible. However, we would still like to improve on some aspects by incorporating following possible ideas:

1 Integrating the tracker module with Real-Time Object Detection and Re-id system. This way we can rely less on heavy Object detection modules (step towards detection free tracking) and thus increase the throughput in terms of FPS

2 Predicting many more future frames for a Trajectory analysis. This would help in getting a sense of Object direction and speed and allow us to model a long term behavior of the object

- Extending the Multi-Object paradigm for Multiple-Cameras. This would help in detecting and tracking an interested suspect for a spread out environment.

- Exploiting the LSTM's inherent pattern learning properties to predict actions. Action detection and Recognition would allow us to further understand the scene, semantically.

REFERENCES

[1] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *arXiv preprint arXiv:1812.08008*, 2018.

[2] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking*, 2016.

[3] W. Luo, X. Zhao, and T. Kim, "Multiple object tracking: A review," *CoRR*, vol. abs/1409.7618, 2014.

[4] R. J. Hyndman, "Measuring forecast accuracy,"

[5] C. Olah, "Understanding lstm networks." [Online; accessed 2019-05-14].

[6] V. Gupta, "Multi-person pose estimation in opencv using openpose." [Online; accessed 2019-06-07].

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] L. Bontemps, J. McDermott, N.-A. Le-Khac, *et al.*, "Collective anomaly detection based on long short-term memory recurrent neural networks," in *International Conference on Future Data and Security Engineering*, pp. 141–152, Springer, 2016.

[9] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang, "Spatially supervised recurrent convolutional neural networks for visual object tracking," *CoRR*, vol. abs/1607.05781, 2016.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[11] S. Schulter, P. Vernaza, W. Choi, and M. Chandraker, "Deep network flow for multi-object tracking," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2730–2739, July 2017.

[12] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler, "Online multi-target tracking using recurrent neural networks," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[13] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-convolutional siamese networks for object tracking," *CoRR*, vol. abs/1606.09549, 2016.

[14] D. Gordon, A. Farhadi, and D. Fox, "Re3 : Real-time recurrent regression networks for object tracking," *CoRR*, vol. abs/1705.06368, 2017.

[15] Z. Cui, R. Ke, and Y. Wang, "Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction," *CoRR*, vol. abs/1801.02143, 2018.

[16] Q. Chu, W. Ouyang, H. Li, X. Wang, B. Liu, and N. Yu, "Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 4846–4855, 2017.

[17] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Detect to track and track to detect," in *IEEE International Conference on Computer Vision*, 2017.

[18] R. Girdhar, G. Gkioxari, L. Torresani, M. Paluri, and D. Tran, "Detect-and-track: Efficient pose estimation in videos," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 350–359, June 2018.

[19] T. Kokul, C. Fookes, S. Sridharan, A. Ramanan, and U. A. J. Pinidiyaarachchi, "Gate connected convolutional neural network for object tracking," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 2602–2606, Sep. 2017.

[20] K. Fang, "Track-rnn: Joint detection and tracking using recurrent neural networks,"

[21] A. Sadeghian, A. Alahi, and S. Savarese, "Tracking the untrackable: Learning to track multiple cues with long-term dependencies," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[22] P. Lin, X. Mo, G. Lin, L. Ling, T. Wei, and W. Luo, "A news-driven recurrent neural network for market volatility prediction," in *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, pp. 776–781, Nov 2017.

[23] A. Ray, S. Rajeswar, and S. Chaudhury, "Text recognition using deep blstm networks," in *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*, pp. 1–6, Jan 2015.

[24] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 2873–2879, 2016.

[25] Q. Wang and H. Huang, "Learning of recurrent convolutional neural networks with applications in pattern recognition," in *2017 36th Chinese Control Conference (CCC)*, pp. 4135–4139, July 2017.

[26] D. H. Oh, Z. Shah, and G. Jang, "Line-break prediction of hanmun text using recurrent neural networks," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 720–724, Oct 2017.

[27] D. Reid, "An algorithm for tracking multiple targets," *IEEE transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[28] G. L. Masala, B. Golosio, M. Tistarelli, and E. Grosso, "2d recurrent neural networks for robust visual tracking of non-rigid bodies," in *Engineering Applications of Neural Networks - 17th International Conference, EANN 2016, Aberdeen, UK, September 2-5, 2016, Proceedings*, pp. 18–34, 2016.

[29] J. Dequaire, P. Ondruska, D. Rao, D. Z. Wang, and I. Posner, "Deep tracking in the wild: End-to-end tracking using recurrent neural networks," *I. J. Robotics Res.*, vol. 37, no. 4-5, pp. 492–512, 2018.

[30] A. Dertat, "Applied deep learning - part 1: Artificial neural networks." [Online; accessed 2019-05-19].

[31] L. Cehovin, A. Leonardis, and M. Kristan, "Visual object tracking performance measures revisited," *CoRR*, vol. abs/1502.05803, 2015.