CS230

A Siamese Network Approach to Multiple-Object Tracking (Computer Vision)

Frederic Ladouceur
Institute for Computational and Mathematical Engineering
Stanford University
ladouceu@stanford.edu

1 Introduction

The problem tackled in this project is that of Multiple-Object Tracking (MOT). This is a hot and relatively recent subject in the field of machine learning (ML) since the publication frequency in this field had a sudden growth in the last few years with the advent of Deep Learning (DL). This field evolved from the field of Single-Object Tracking (SOT).

MOT consists in tracking the displacement of multiple objects from frame to frame in videos. Algorithms approaching this problem are usually tackling it in 4 different stages (2):

- Detection stage: detection of the object in each frame.
- Feature extraction/motion prediction stage: extraction of visual features and/or prediction of the motion of objects in the frame.
- Affinity stage: features and motion predictions from the previous stage are used to compute a similarity/distance between pairs of objects from one frame to the next.
- Association stage: Assignment of IDs to pairs of objects that are identified as being the same object in subsequent frames.

In this project, we focus on the three latter stages which is a common practice in the field of MOT. Therefore, our problem setting is: given a set of objects detected by a state-of-the-art ML detection system, perform online MOT on all frames in a video sequence. Here the term "online" means that the tracking algorithm cannot peak into the future and can only use the current and past frames when performing the tracking. Also, we will say that our algorithm uses public detections since it uses the set of objects detected by a state-of-the-art ML detection.

Some interesting applications for this field of study are video surveillance, self-driving cars, sports analytics, etc.. Being personally very interested in everything related to stochastic control (e.g. Reinforcement Learning), I see how this kind of technology could be used in control problems such as self-driving cars and other autonomous systems that are designed to interact with humans, which is why I think this field is so fascinating.

This report is divided into multiple sections: section 2 covers the data used to train and test our approach, as well as the code, section 3 covers the algorithm developed to perform MOT, section 4 presents the experimental results, section 5 presents a discussion of the approach and its results, and section 6 concludes this report.

2 Source Code and Datasets

The source code for my project can be found in this GitHub repository. The code for this project consists of 3 main notebooks: *train_siamese_model*, *produce_test_tracking*, and *visualize_test_tracking*. The first notebook is used for training the model we use for MOT, the second one is used to perform MOT on video sequences and the third one is used for visualizing the MOT results produced. The other notebooks found in under 'archive/' are messy development notebooks. See the *README* file in the repository for more details.

The dataset used for training and testing are taken from the MOTChallenge website as theses are benchmark datasets (1) used by the field of MOT to evaluate on compare various approaches. More specifically, we used the dataset MOT16 that is made of 14 videos, 7 of which are for training and 7 for testing. Videos are filmed with both static and moving cameras and objects are tracked and identified. The ground truth MOT results (i.e. bounding boxes and ID for each object in each frame) and the public detection results (i.e. each frame is run through a state-of-the-art detection algorithm which gives us bounding boxes around each detected object) are also provided with the image data.

3 Algorithm

Recall the 4 stages described in section 1: detection stage(i), feature extraction and motion prediction stage(ii), affinity stage (iii), and association stage (iv). In this section, the algorithm will be described from the perspective of the last three stages (because the public detections are provided to us).

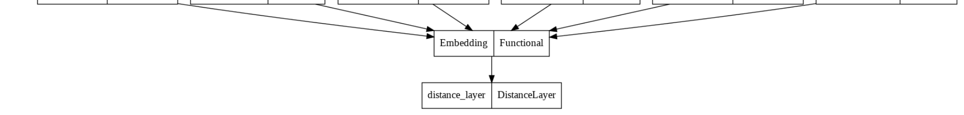
3.1 Feature extraction and motion prediction stage (ii)

In this algorithm, the features of interest are visual features from the object image (object cropped out of the frame image and resized to a (200,200) shape) and object location features (relative object bounding box coordinates, i.e. each of the four coordinate points falls in the range from 0 to 1). This is done by feeding the object image into the pre-trained ResNet50 model (3) which output a (7,7,2048) tensor as output. This output tensor is then flattened and fed into a dense layer with *relu* activation and batch normalization before being concatenated with a 4-element-long array representing the objects location. This resulting array is finally fed into one 128-wide dense layer with *relu* activation and batch normalization, and one 128-wide dense layer which outputs a 128-element-long vector embedding representation of object. We call this first model: '*Embedding*', and it has 49M parameters, of which only 26M are trainable as we set the Resnet parameters to be non-trainable. See figure 3 in the appendix for a diagram showing the last few layers of the *Embedding* model.

3.2 Affinity (iii)

In this algorithm, we define a distance between two objects as the euclidean-norm of the difference of their embedding vectors. This distance is implemented as the last layer in a siamese neural network. The siamese neural network is designed to take as input a triplet of inputs (anchor, positive and negative examples), each one is fed into the *Embedding* model. Then, the distance layer takes as input the embedded triplets and outputs the euclidean distance between the anchor and the positive example and that between the anchor and the negative example. We call this siamese network the *Siamese Model*, and its goal is to model the distances between an anchor object in a frame, and a negative and a positive example in the successive frame. See figure 1 for a diagram presenting the *Siamese Model* architecture.

Figure 1: Diagram showing the *Siamese Model*'s architecture


anchor_image | InputLayer
positive_image | InputLayer
negative_image | InputLayer
tf.__operators__.getitem | SlicingOpLambda
tf.__operators__.getitem_1 | SlicingOpLambda
tf.__operators__.getitem_2 | SlicingOpLambda
tf.nn.bias_add | TFOpLambda
anchor_location | InputLayer
tf.nn.bias_add_1 | TFOpLambda
positive_location | InputLayer
tf.nn.bias_add_2 | TFOpLambda
negative_location | InputLayer
Embedding | Functional
distance_layer | DistanceLayer

3.3 Training

The goal of the *Siamese Model* is to learn to that the anchor in one frame is close in distance (distance as defined in section 3.2) to the positive example in the next frame (same *id*) and that it is far in distance from the negative exampe in the next frame (different *id*). To accomplish that, the loss that was chosen to train the *Siamese Model* is defined as follows:

$$\mathbb{L}(a, p, n; margin) = max\{distance(a, p) - distance(a, n) + margin, 0\} \tag{1}$$

Here, the *margin* is some positive scalar that is set to 0.5, $\{a, p, n\}$ represents the triplet: {anchor, positive example, negative example}, and the distances are the outputs of the *Siamese Model* as described above. The optimizer used for to minimize this loss is the *Adam* optimizer with a starting learning rate of 0.0001. This model also constantly monitor the training loss and reduces the learning rate by a multiplicative factor of 0.1 when the training loss plateaus for at least 2 epochs (see figure 4 in appendix).

Note that this loss function with this model and these inputs is minimized when the distance between the anchor and the positive example shrinks and the distance between the anchor and the negative example grows. The max operator and the *margin* term allows us to stop the optimizer from focusing solely on maximizing the distance between the anchor and the negative example.

As described in section 1, the data we use for training counts 7 videos. In order to get good estimate of the ability of our model to generalize, we use only the first 6 videos for training and the last video only for validation. Having this validation dataset is useful when making choices on what combinations of hyperparameters to use for our model as explained in section 4.

3.4 Association (iv)

The association is done as follows. Labelling all detected objects in the first frame of each video with a unique identifier (e.g. {1,2,3,...}). Then, following the natural sequential order of the frames, iterate through all pairs of frame in each video $(frame_i, frame_{i+1})$. For each of these pairs, perform one-to-one matches between the objects in $frame_i$ with those in $frame_{i+1}$. This matching process (or association). This matching simply finds, for each object in $frame_i$, the object in $frame_{i+1}$ closest in distance given that this distance is smaller than some constant $threshold = 1.5$. If two objects in $frame_i$ have the same best match in $frame_{i+1}$, then only the one with that is closest keeps the match. The other one is matched to the next best available object of $frame_{i+1}$. The action of matching simply means to assign the labelled identifier of an object in $frame_i$ to its matched object in $frame_{i+1}$. If an object in $frame_{i+1}$ finds no object in $frame_i$ that is closer than $threshold = 1.5$ in distance, then a new unique identifier is created and assigned to it. Once this process is performed for all pair of successive frames in all videos, the association stage is completed.

4 Hyperparameters tried

In this project, multiple model architectures and hyperparameters were tried. The most promising one based on its evaluation on the validation set was kept and is described in the sections above. More precisely, the one that produced the lowest validation loss (loss (1) as defined in section 3.3) was kept. Here are a few other things that were tried:

- Only use the the object image information and not the object location (bounding box coordinates). The same number of epochs of training would result in a higher validation loss.
- Concatenate the location data of the object on a lower layer of the embedding. Again, the same number of epochs of training would result in a higher validation loss.
- Have some of the last layers of the pre-trained ResNet50 model being trainable (non-fixed). This caused the model to show signs of overfitting: the training loss would keep getting closer to 0, but the validation loss would increase as the training went on.
- Increase the number of dense layers at the end of the *Embedding* model. This caused the model to show signs of overfitting: the training loss would keep getting closer to 0, but the validation loss would increase as the training went on.

- Keep the learning rate of the *Adam* optimizer constant throughout the training. This lead to the training and validation losses converging to larger values. The adjustable learning rate we have now is able to find a lower minimum for the loss 1 function.

5 Experimental Results

The MOT Challenge (1) website allows us to evaluate our approach to MOT on the test data that they provide, and to compare our results to that of others. The metrics that are used to evaluate the different approaches are numerous, but we will simply focus on three metrics that seems important in the literature (2): MOTA, MT, and ML.

	MOTA(%)	MT	ML
Ours	12.8	100	280
MrMot	75.0	330	129
GOT	74.7	299	144

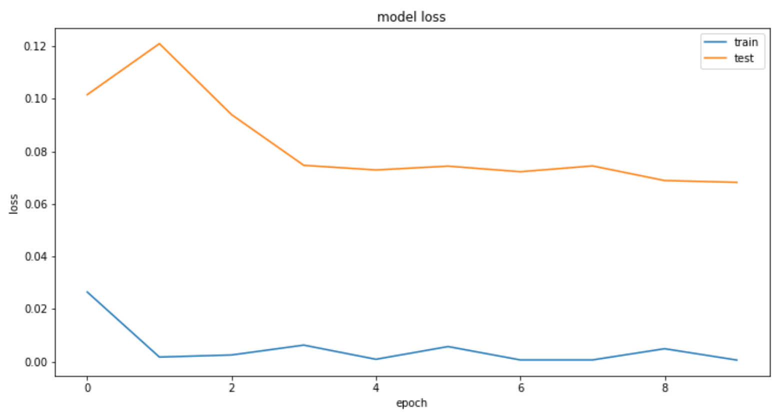
The table 5 above presents the MOT results of three online MOT approaches on the MOT16 (1) test data. The first row shows the results of our approach, and the second and thrid rows show the results of two of the best online MOT algorithms on the MOT Challenge leaderboard.

Here is a short description of each metric (2):

- MOTA: $1 - \frac{FN+FP+IDSW}{GT}$, where GT is the number of ground truth boxes there are, FP is the number of hypothesis tracking that cannot be associated with a ground truth object, FN is the number of ground truth objects that cannot be associated with a hypothesis tracking, and IDSW is the number of times the ID of a tracked object is incorrectly changed. This value is expressed as a percentage.
- Mostly Tracked (MT) trajectories (4): number of ground-truth trajectories that are correctly tracked in at least 80% of the frames.
- Mostly Lost (ML) trajectories (4): number of ground-truth trajectories that are correctly tracked in less than 20% of the frames.

Additionally, the average training and validation losses 1 of the *Siamese Model* are also quantities of interest for the evaluation of this approach. You can observe a plot in figure 2 showing their evolution as the training progresses. The lowest average validation loss was 0.06818 and the associated training loss was 0.00056.

Figure 2: Plot showing the evolution of the training and validation loss during training



6 Discussion

MOT projects are difficult because the metrics that are used to evaluate the performance of algorithms are not directly linked to the NN model performance, but rather to the ensemble of algorithms used to tackle all the different stages, i.e. detection, feature extraction and motion prediction, affinity and association (as seen in section 1. Therefore, even though the training of the *Siamese Model* neural network shows promising results based on the validation loss, as we can observe in figure 2, the actual performance of the MOT algorithm is poor when evaluated according to the metrics shown in table 5.

The approach developed in this project has a definite worst performance than the other two approaches for all 3 metrics. The MOTA score, which is a summary statistics of the quality of the overall MOT, is very low for our approach.The MT, which captures the number of mostly tracked trajectories, is very low for our approach. Finally, the ML, which captures the number of mostly lost trajectories, is very high for our approach. After visually observing the tracking results produced by our approach, it is clear that the relative bad performance as measured by these three metrics when comparing to other approaches has the following causes:

- The object tracking works by matching detected objects in one frame with detected objects in the next frame. If some objects are not detected in the next frame, then the object is lost, and in the next frame that it is encountered, it will get assigned a new ID.
  - Possible improvement: Perform motion prediction for the next frame using past frames as is often done in this field. This motion prediction could be done by feeding the past object location vectors into an LSTM model to predict the next location vector.
- The *Siamese Model* is trained with triplets formed using the ground truth bounding boxes. However, when using this same model on test data, its input are formed using the public detections which are not as precise, and noisier than the ground truth bounding boxes.
  - Possible improvement: Add noise to the ground truth bounding boxes to form the training data set.

7 Conclusion

One of the main key learnings we can take home from this project is not all projects that involve NN are easy to evaluate, and this property plays a big role in our ability to iterate and improve our approach. MOT is a field where it is quite difficult to pin-point which part of our algorithm would need improvement.

The MOT approach that we propose leverages transfer-learning by using the pre-trained ResNet50 NN to capture visual features and a siamese network to perform embedding of the objects in each frame based on these extracted visual features and the object location. Even though our approach seems conceptually correct at first glance, it was trained using ground truth data and has some difficulties coping with the imperfections of the public detections it uses when being tested. A natural future direction for this work would be to find ways to make our algorithm more robust to imperfect object detections.

References

[1] Leal-Taixé, L., Milan, A., Reid, I., Roth, S. and Schindler, K., 2021. *MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1504.01942v1> [Accessed 6 November 2021].

[2] Ciaparrone, G., Luque Sánchez, F., Tabik, S., Troiano, L., Tagliaferri, R. and Herrera, F., 2021. *Deep learning in video multi-object tracking: A survey*.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.90

[4] Bo Wu and Ram Nevatia. *Tracking of multiple, partially occluded humans based on static body part detection*. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), volume 1, pages 951–958. IEEE, 2006.

8 Appendix

Figure 3: Diagram showing the last few layers of the *Embedding* model

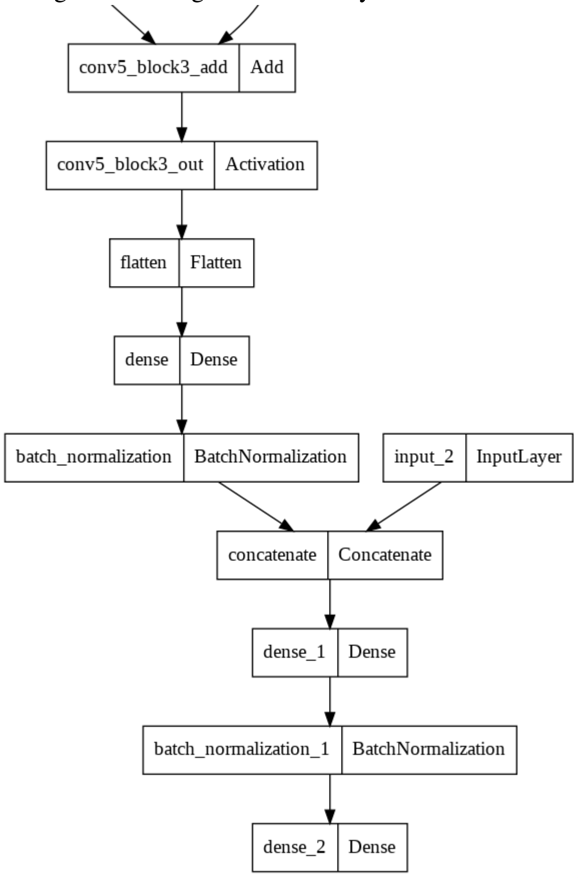
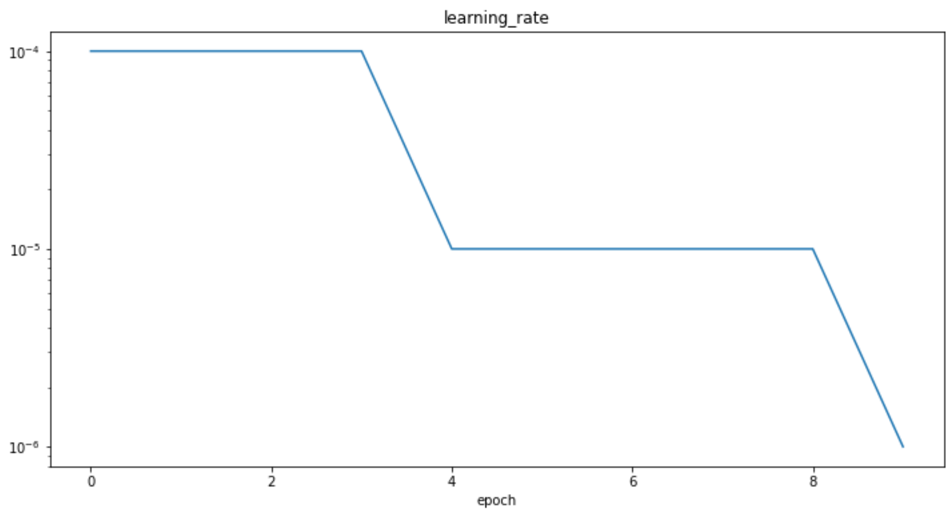

conv5_block3_add | Add
conv5_block3_out | Activation
flatten | Flatten
dense | Dense
batch_normalization | BatchNormalization
input_2 | InputLayer
concatenate | Concatenate
dense_1 | Dense
batch_normalization_1 | BatchNormalization
dense_2 | Dense

Figure 4: Plot showing the evolution of the learning rate as the training progresses


learning_rate
10^{-4}
10^{-5}
10^{-6}
epoch