CSCI 1430 Final Project Report:
Trust Your Tesla: Collision Course Prediction For Simulated Autonomous Vehicles

Team Carla: Byron Butaney, Kaleb Newman
Brown University

Abstract

In recent times, self-driving cars have become increasingly integrated in our everyday lives, with car companies already releasing semi-autonomous capabilities for their high-end cars. At its core, self-driving relies on a vehicle's ability to make predictions based on its current environment at a moment's notice. Interested in exploring this scenario, this project used the Carla driving simulator to simulate an autonomous vehicle and its environment. The dataset used in this project to train our model was taken from images of the Carla environment. Using TensorFlow and Keras, we trained a Convolutional Long Short Term Memory model on a time series data set of 14,000 sequences of 8 images to predict whether a simulated vehicle was on a trajectory at-risk for collision. This architecture achieved a testing accuracy of 91.70% on our testing set, meaning that it can predict whether the Carla vehicle is on a collision course accurately 91.70% of the time.

1. Introduction

We live in an exciting time for autonomous vehicles. With companies like Tesla working to bring self-driving technologies to the masses, it is interesting to understand how autonomous vehicles can use computer vision and deep learning to predict whether various trajectories and decisions are safe or risky. Improving upon autonomous vehicles' capabilities to make these predictions could save numerous lives.

This problem is difficult for a number of reasons. Firstly, we wanted to be able to make predictions from data moments before a collision. In order to do so, we needed to look at more than just a single image – ideally, one could use video data. However, training a model on a large dataset of video data is computationally expensive. To combat this, we decided to look at sequences of 8 consecutive images (time series data). Data collection was also a problem, as we can't afford to test our results on a real autonomous vehicle. Instead, we opted to use Carla, a driving similar designed for deep learning projects, to both source our data and demonstrate our model's prediction capabilities. Lastly, since we were using samples that were sequences of images with a single label (safe or collision), we needed to find a way to take in and predict time series data.

After researching different architectures, we found that Long Short Term Memory (LSTM) cells could be used to make predictions based on time series data. Moreover, we saw that these cells were often combined with Convolutional Neural Networks (CNNs) to improve accuracy. As we learned more about this CNN+LSTM architecture, where a convolutional layer was followed by multiple LSTM layers, we discovered one more approach to making predictions for time series data: Convolutional LSTMs (ConvLSTM). Not to be mistaken with CNN+LSTM, ConvLSTM applies convolutions to the hidden and cell states of every LSTM layer, rather than applying a single convolutional layer before a sequence of LSTM layers. We saw that many approaches to time series prediction for autonomous vehicles in Carla used CNN+LSTM, but not many used ConvLSTM. Yet, ConvLSTM generally has a better spatio-temporal understanding of images within a sequence of related images and requires less parameters than a fully connected LSTM. We thought we could therefore improve upon the results achieved by other Carla users by implementing our own ConvLSTM architecture.

As previously mentioned, if collision predictions can be made at a moment's notice using only a sequence of a few images, this could greatly improve the safety of, in our case, a self-driving car in Carla. The results of using this ConvLSTM architecture could be an interesting way to better understand and improve upon the methods used in real-life autonomous vehicles.

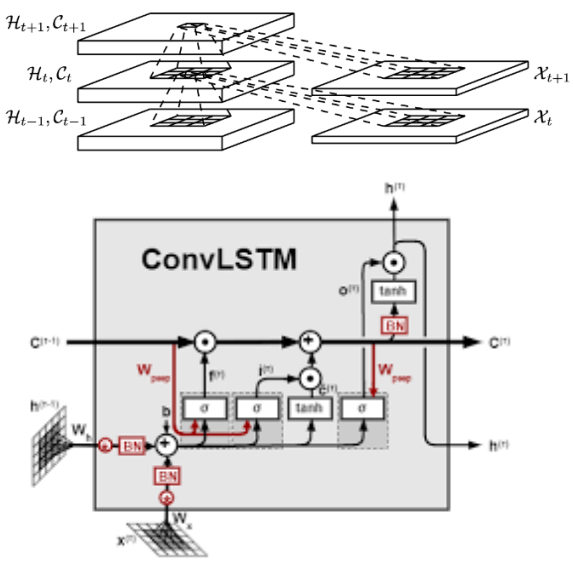
2. Related Work

We entered this project by taking a broad dive into video prediction. Many types of architectures were considered, and have been used traditionally for this type of task. Commonly, Convolutional Neural Networks are used for images and Long-Short Term Memory networks are used for sequential data.

We noticed the CNN+LSTM model was used often for

objectives that require spatial, as well as temporal understanding. We came across:[Brownlee] which explains the architecture. This includes passing data through a CNN model, a LSTM model, then fully connected layers for the desired output. In this architecture the CNN extracts spatial features from the data, the output of that model is then flattened, and fed, into the LSTM model to support sequence prediction.

Another architecture we studied came from [N. Srivastava [2015]]. Srivastava et al., use a multilayer LSTM encoder-decoder network. Their encoder maps an input sequence into a fixed length representation. Then their decoder reconstructs the hidden representation to predict the future sequence. To build upon, and out perform these approaches we came across The Convolutional LSTM network (different from the CNN +LSTM). [X. Shi [2015]]



The Convolutional LSTM unit is defined as:

[1] $i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i$

[2] $f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f)$

[3] $C_t = f_t \circ C_{t-1} + i_t \circ tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c)$

[4] $o_t = \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o)$

[5] $H_t = o_t \circ tanh(C_t)$

$\circ$ Represents the Hadamard product

$*$ Represents convolution

Lines 1,2, and 4 represent gates, which are integral in preserving relevant features of data over time. Specifically, they are used to calculate the hidden state H_t and the cell state C_t, which represents the networks short and long term memory.

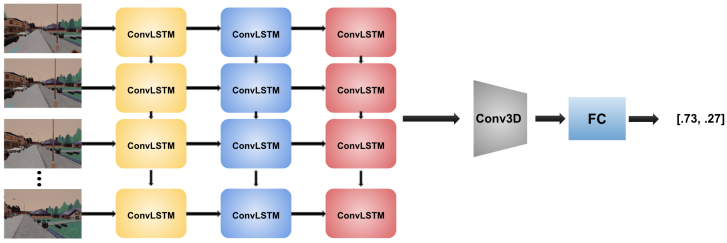
The equations that govern the Convolutional LSTM cell are similar to the fully connected LSTM in that, this network determines the future state of a cell based on the input, hidden state, and the cell state. But instead, the matrix multiplication between weights and inputs or weights and hidden states, is replaced by the convolution operation. Applying this convolution through the gated structure allows relevant spatial features preserved and related across a sequence. This learning is also augmented by the partially-connected nature of the ConvLSTM– garnering translational invariance, reducing redundancy, and reducing the number of parameters. This transition is analogous of that from FC-Networks to the CNN.

Shi et al,. showed that this model performs better on spatiotemporal data compared to purely LSTM based approaches. Video data can also be fed directly into the ConvLSTM. The input, hidden states, cell states, and gates are all 3D tensors whose dimensions are: (height, width, no. channels).

We used our model to build on the efforts by [gpr]] This was used solely to get our dataset as we did not have the compute power to create our own. Also the interface meant to run this model in is CARLA Driving Simulator. To implement this model we used Tensorflow as well as Google Colaboratory.

3. Method

To predict whether or not the passenger will be in danger, the model needs to develop spatio-temporal reasoning. The goal is, given a sequence of frames, our model can learn hidden representations across time in order to predict the state of the passenger's safety. The predictive output of the passenger's safety will be given before the "final action" occurs. For example, the model would predict that the passenger is at risk if it was on a trajectory to crash into another car, before the crash happens. To achieve this, we used Convolutional LSTM networks.



The inputs to the model are batches of 4-Dimensional tensors in the shape of tensor $\chi \in \mathbb{R}^{F \times M \times N \times C}$ where F denotes the sequence length and each frame is M x N with C color channels. This is in alignment with the input

size requirement of a ConvLSTM which are 5-Dimensional tensors (including batch size). The input is fed into 3 stacked Convolutional LSTM networks, each with 64 filters and ReLU non-linearity functions. The final ConvLSTM returns the hidden state at each timestep as a sequence which is a 5D tensor $Y \in \mathbb{R}^{B \times F \times L \times P \times C}$, where B is batch size and L and P are the dimensions of an output frame from the ConvLSTM.

This is fed into a 3D convolution layer with one filter. The output is then passed through a fully connected network of hidden layer size 512, 64, and 2 respectively. A dropout rate of 0.5 was used in between the linear layers and the Softmax activation function was used on the last layer.

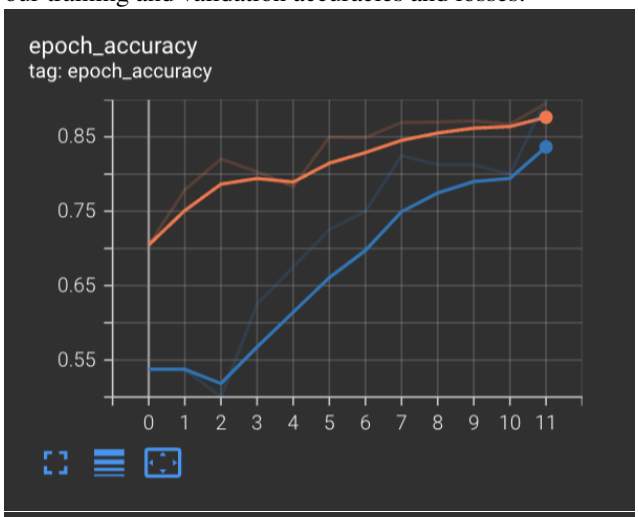
Layer (type)	Output Shape	Param #
conv_lstm2d_3 (ConvLSTM2D)	(8, 8, 140, 210, 64)	429056
time_distributed_8 (TimeDistributed)	(8, 8, 140, 210, 64)	256
time_distributed_9 (TimeDistributed)	(8, 8, 47, 70, 64)	0
conv_lstm2d_4 (ConvLSTM2D)	(8, 8, 47, 70, 64)	819456
time_distributed_10 (TimeDistributed)	(8, 8, 47, 70, 64)	256
time_distributed_11 (TimeDistributed)	(8, 8, 16, 24, 64)	0
conv_lstm2d_5 (ConvLSTM2D)	(8, 8, 16, 24, 64)	295168
time_distributed_12 (TimeDistributed)	(8, 8, 16, 24, 64)	256
time_distributed_13 (TimeDistributed)	(8, 8, 6, 8, 64)	0
conv3d_1 (Conv3D)	(8, 8, 6, 8, 1)	1729
flatten_1 (Flatten)	(8, 384)	0
dense_4 (Dense)	(8, 512)	197120
dropout_3 (Dropout)	(8, 512)	0
dense_5 (Dense)	(8, 64)	32832
dropout_4 (Dropout)	(8, 64)	0
dense_6 (Dense)	(8, 2)	130

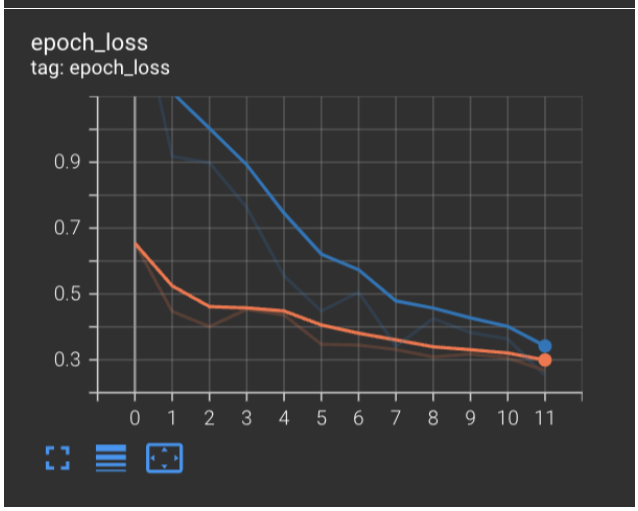
Total params: 1,776,259
Trainable params: 1,775,875
Non-trainable params: 384

We chose the 3D Convolution layer to have one filter to compress the output of the ConvLSTM. Also in our implementation we used time distributed batch normalization and pooling to apply these operations to each frame in the sequence.

4. Results

Though we tried a variety of settings for our training process, we ended up settling on the following configuration: due to the computational limitations Google Colab Pro, we limited our mini-batch size to 8 and kept our epochs at 11. We used sequences of 8 images (or frames) and had 2 class: Safe and Collision. We also used callbacks for tensorboard and to save our model, and shuffled our data using TensorFlow's model.fit() function. Here are the curves for our training and validation accuracies and losses:





After training our model, we were able to achieve predictions based on sequences of images. Below are two examples of sequences, one safe and one dangerous, that were properly labeled by the model. Though sequences of four images are depicted, eight were used.

4.1. Technical Discussion

Our model was built for spatio-temporal reasoning, which raises the question whether solely vision is sufficient to assess the safety of the passenger. Our model only takes in

sequences of images as opposed to sensor data or even audio. An interesting extension of this project would be to see if the accuracy can be improved upon if this task was treated as multimodal. This debate is relevant in 2022 as Tesla has decided to use only vision for their autonomous vehicles.

Our data set choice raises a few interesting questions. Firstly, due to lack of compute resources, even on Google Colab Pro, we could only use sequences of 8 images. Perhaps access to stronger GPUs would have allowed us to train the model on larger sequences of images, which would train it to have a better understanding of the trajectory of the vehicle. In addition, we would be able to experiment whether or not an increase in sequence length would be directly proportional to accuracy.

Secondly, since our images are from a simulated environment, they are much simpler than real life scenarios, which helped us to achieve such a high accuracy. Moreover, while some of the images in the dataset included cars, the model was not trained to deal with busy intersections filled with many cars with different trajectories. Perhaps having more compute power and access to a better dataset would allow us to make our model more generalizable to real-world scenarios. Lastly, we found that changing the number of filters of our Conv3D layer to 1 allowed us to achieve a large accuracy jump, from around 79% to the 91% we currently have. This raises an interesting question as to why 1 filter worked so well. Here is the difference in results between our 32 filter Conv3D layer and our 1 filter Conv3D layer:

Architecture	ConvLSTM(3 x64)-Conv3D(32)-FC	**ConvLSTM(3x64)-Conv3D(32)-FC**
Train Accuracy	0.8312	**0.8951**
Validation Accuracy	0.7375	**0.9000**
Test Accuracy	0.7872	**0.9170**

4.2. Societal Discussion

As autonomous features become increasingly present in vehicles, from lane assist that keeps the vehicle between lane lines and automated lane changing to safety features like automated emergency breaking. While these (and future) features are impressive and exciting, such technology also raises some important ethical concerns. For example, what happens if the automated emergency breaking misfires and causes an accident? Is it fair for the driver of the car to be responsible for the accident? If not, who is responsible? This scenario can be extrapolated to a more general scenario as well: is a company responsible for accidents or deaths caused by their vehicles? After all, it is impossible to achieve a perfect accuracy for any AI model. At what frequency of incidence, then, should car companies be responsible?

Our project seeks to illuminate the types of decisions and features that autonomous driving systems learn. Despite being in a simulated environment with less complexity than real-life environments, our model was only able to achieve an accuracy of, approximately, 91%. This is not nearly high enough for any functional autonomous vehicle, and our accuracy is, again, based on a simulated dataset. This illuminates both the difficulty of developing a fully autonomous vehicle and the need for a rigorous and extensive dataset that is consistently updated for a real-world self-driving algorithm that is good enough to be put to use.

One way we could have improved our project, and something that further highlights the ethical complexity of self-driving algorithms, is the prospect of having a car that can generate decisions to avoid collisions. Though we did not implement this functionality, it is the natural next-step-forward for our project. How, then, should a model deal with two equally bad scenarios? What determines the level of danger of a potential collision? An interesting ethical consideration would be finding a way to classify collisions from mild to severe. In this implementation, hitting a human might be the most severe, whereas hitting a bush may be mild. In a scenario where the car is either going to hit a bush or a pedestrian, then, we would want to improve our project to be able to veer towards the bush instead of the pedestrian. These are some of the many ethical concerns that need to be considered when working on autonomous vehicles in any capacity.

5. Conclusion

Overall, we managed to design an architecture and train our model that can determine whether a simulated vehicle in Carla simulator is about to experience a collision. This is incredibly relevant to the trend of autonomous driving capabilities becoming increasingly prevalent in modern society. Going forward, being able to improve upon models like our ConvLSTM architecture could save countless lives. Our project highlights the necessity not only for high-performing architectures that can take in sequential data, but also for rigorous and thorough datasets to train new models on. If autonomous driving is going to be a reality, reliable safety will be of the utmost importance.

Appendix

Team contributions and TA meeting dates

Dates we met with our TA Monday, November 21 and Thursday, December 8.

Byron Byron handled converting the data set into a usable form. He also figured out the dimensions of the samples in our dataset. Lastly, he wrote the training loop and

evaluation code, as well as the callback code to save tensorboard logs and train weights.

Kaleb Kaleb worked on researching different architectures and came up with the idea to use some form of LSTM. He worked on experimenting with different architectures with some help from Byron. He also created the final architecture that we ended up using.

Both Both Byron and Kaleb worked together to design the goals of the project and to experiment with different technologies. For example, Byron and Kaleb initially tried to write their code in PyTorch. They wrote a ConvLSTM cell from scratch (since, unlike TensorFlow, PyTorch doesn't have a built-in for ConvLSTM) and wrote the PyTorch Dataset/DataLoaders. However, PyTorch ended up being computationally more expensive than TF, and we ended up switching to our current TensorFlow implementation. Byron and Kaleb both also worked on designing the poster.

References

Vehicle_collision_prediction_using_cnn-lstms. https://github.com/perseus784/Vehicle_Collision_Prediction_Using_CNN-LSTMs. Accessed: 2022-11-25. 2

J. Brownlee. 2

R. Salakhutdinov N. Srivastava, E. Mansimov. Unsupervised learning of video representations using lstms. *arXiv*, 2015. 2

W. Yeung W. Wong W. Woo X. Shi, Z.Hao. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *arXiv*, 2015. 2