Neural ODEs for Latent Feature Recurrence in Streaming Perception

Author: Zack Dugue
Mentor: Ivan Jimenez
Principal Investigator: Yisong Yue

Abstract:

Streaming perception is a type of computer vision used to label the location and class of objects in a stream of video in real time. Current streaming perception architectures rely on rapid inference that can be completed before the next frame of video is available. This means that, for each frame, a great deal of latent feature extraction occurs, but these latent features are essentially thrown away when the next frame is considered. We propose a novel architecture which utilizes a neural ODE to maintain a continuous state between frames, thus allowing features from prior frames to impact the inference of the current frame. By training and testing this architecture on the ArgoverseHD streaming dataset, we hope to show improved performance on streaming perception benchmarks. We hope that this architecture will allow for more parameter-light, more accurate streaming perception architectures for applications such as autonomous driving.

Background:

Streaming Perception is a computer vision application that involves labeling the location and class of objects in a stream of images in real time. The locations of the object are described via “bounding boxes” which are rectangles given by coordinates on the image itself intended to contain the object. The class of the object is usually labeled according to the streaming perception task. For example, in self-driving applications “car, traffic light, pedestrian” could be the classes of objects. All objects which are not a car, traffic light, or pedestrian, are labeled as “background.”

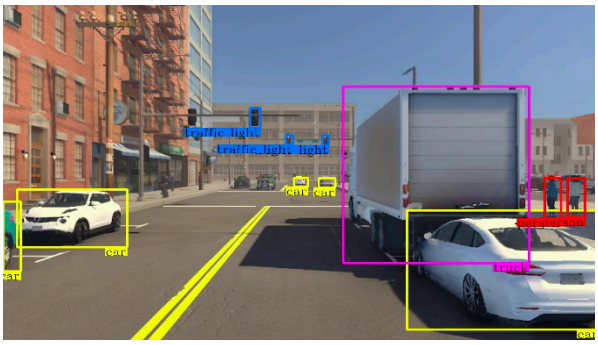


Diagram 1 – An image augmented by the bounding boxes and labels of a detector model.

The objects in a scene must be bounded and labeled in real time, which, for a camera with 30 fps video, is about 30 frames per second. How quickly the inference occurs is relevant to the quality of the detection, because in deployment tasks objects will move in the frame. So a 100% accurate inference made with two seconds of latency, may not be as useful as an 80% accurate inference made with one second of latency.

Faster RCNN:

Current state of the art streaming detection models emphasize rapid inference, focusing on trying to output an accurate inference before the next frame appears. The model we will be working with most is

known as Faster RCNN. Faster RCNN takes an image as input, and extracts "features" from that image using a convolutional neural net backbone. Then it uses those features as part of a "two stage detector". One stage of the detector called the "Region Proposal Network" identifies "Regions of Interest." Through a method called "ROI-Pooling" the second stage of detector, the ROI-Head, evaluates the extracted features in these regions of interest to determine the bounding box of the object contained and the class of the object. Most of our work will focus on the Convolutional Neural Net Backbone of the Faster RCNN.

A neural ODE is a type of machine learning algorithm which learns "dynamics" of an evolving state. I would refer to the description and notation used in [cite1] and [cite2].

Standard implementations of modern streaming perception models treat each new frame as though it were brand new, even though each frame is nearly identical to the frame before it. This means that much of the feature extraction between each frame is redundant, and in practice this leads to higher latency for models and thus lower accuracy.

Our approach:

Consider some layer of the CNN backbone. We represent the activations of said layer with the vector η . Now I reference the notation from cite1 and cite2. Where, for us, said equations instead look like this:

$$\eta_0 = \psi_\theta(x_0) \frac{d\eta}{dt} = f_\theta(\eta_t, \xi(x_{t+1}), t) \hat{y}_t = \varphi_\theta(\eta_t)$$

In our case $\eta(0)$represents the activations of the layer at time 0. x_0 is simply the first frame of the video. $\psi_\theta(x_0)$ is the initialization function. $\frac{d\eta}{dt} = f_\theta(\eta, \xi(x_{t+1}), t)$ describes how the state of the layer changes over time. $\xi_\theta(:)$ represents the network layers before the current layer. $\hat{y}(t)$ is the inference made by the model, and $\varphi_\theta(:)$ represents the rest of the network that comes after x.

A "good" state is a state η_t s.t. $L(\hat{y}_t, y_t)$ is small. Where $L(\hat{y}_t, y_t)$ is some measure of the difference between the predicted bounding boxes and labels, represented by $\hat{y}_t$ and the true bounding boxes and labels, represented by y_t

Now consider a good state η_a, and the good state η_{a+1}. If both these states are good then $L(\hat{y}_a, y_a)$ is small and $L(\hat{y}_{a+1}, y_{a+1})$ is also small. Note also that y_a and y_{a+1} are likely very similar. This is because the object bounding boxes and class labels are unlikely to have changed much between frames.

They are also likely to have changed in relatively predictable ways. For example, a bounding box around a car driving left, is likely to shift left in the next frame.

Note then the evolution from η_a to η_{a+1}.

$$\eta_{a+1} = \eta_a + \int_0^1 f_\theta\left(\eta, \xi(x_{a+1}), \tau\right) d\tau$$

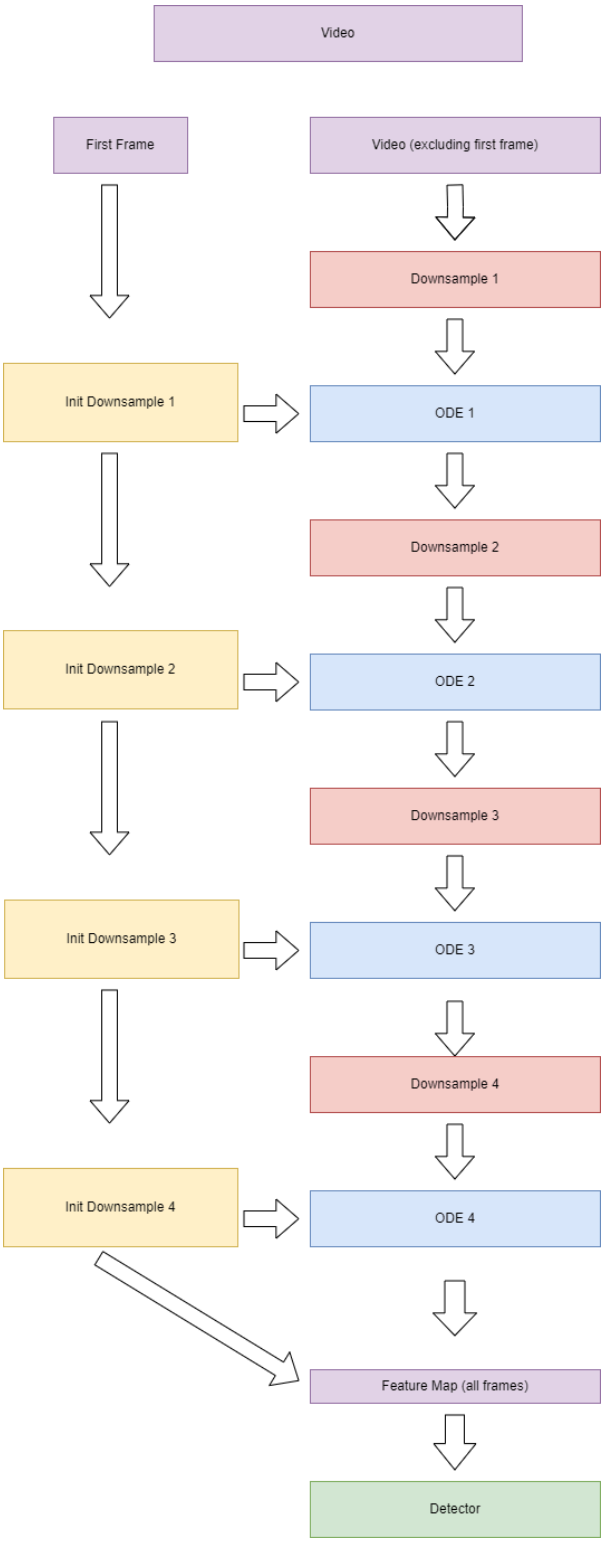
So:

$$\eta_{a+1} - \eta_a = \int_0^1 f_\theta\left(\eta, \xi(x_{a+1}), \tau\right) d\tau$$

Since we know that the difference $\eta_{a+1} - \eta_a$ is small and predictable. It then follows that $\int_0^1 f_\theta\left(\eta, \xi(x_a), \tau\right) d\tau$ is also small, and that the dynamics within the integral should be relatively easy to learn. This is the idea behind our Neural ODE approach. Given a "good" starting state ($\eta_0 = \psi_\theta\left(x_0\right)$ s.t. $L\left(\hat{y}_0, y_0\right)$ is small) we want to learn the dynamics, that can transition to the next good state. And the calculations for this integral $\int_0^1 f_\theta\left(\eta, \xi(x_a), \tau\right) d\tau$ should be faster than computing a new "good" state directly from x.

The Architecture:

The Convnet backbone we use in our network looks like this:

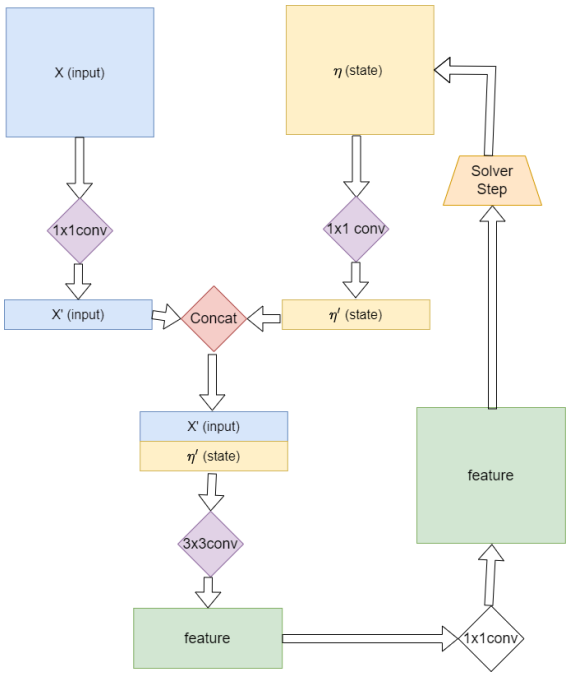


When a stream of video is passed to our model, the first frame is passed through some “initialization backbone”. This initialization backbone is used to determine the initial states of the ODE blocks. Then the remaining frames of video are passed through the main body of the detector. The downsample blocks are similar to what would be found in a normal Resnet. The ODE blocks integrate the dynamics as described previously. After the integration, the state of the ODE-layer at each frame is passed to the next layer. This architecture of down sample layers combined with ODE layers is loosely based off of the ContinuousNet architecture.

To initialize the states of each layer we use a Resnet50 that’s been pre-trained on a detection task. This Resnet 50 is then frozen for the training of the larger network. In deployment this initialization would be used to get a good starting state for the network, and then the ODE would be integrated for each frame. Where the state of any given ODE block would be the result of the prior frame’s integration.

The dynamics used in each ODE block were modeled after the “Bottle Neck Dynamics” of large CNNs like Resnet 50. In order to mix information from the current state and input, a 3x3 convolution between bottlenecked features of both states is used.

In the Diagram



Currently the design of these dynamics are to use static weights. In the [cite ContinuosNets] paper, the Authors use time varying dynamics, which allow them to increase their effective parameter count. We also attempt something similar.

Results:

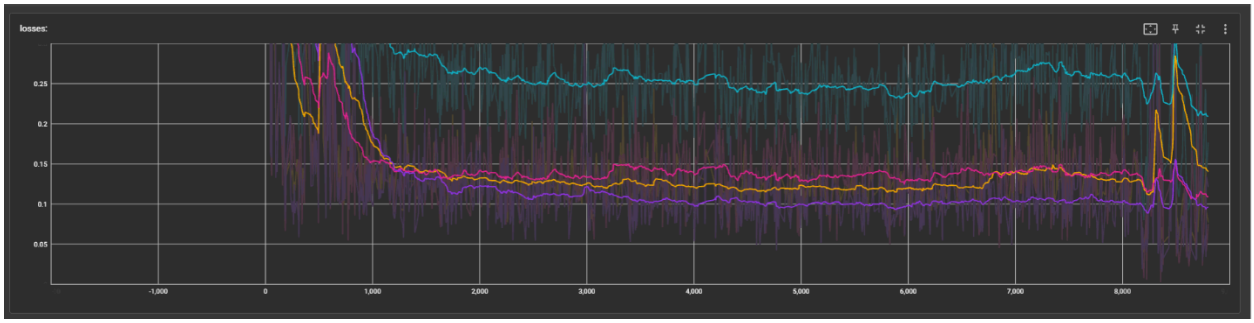
8-step, 4-step, and 2-step, refer to the number of evaluations used to integrate the dynamics for each ODE-block. Euler and RK4 refer to the integration methods used.

Approach	**Computation Time Per Frame**
Resnet 50	**0.0088s**
ODE-8-Step-Euler	**0.0102s**
ODE-4-Step-Euler	**0.0077s**
ODE-2-Step-Euler	**0.0059s**
ODE-4-Step-Rk4	**0.0174s**

Table 1 - As can be seen here, only certain integration parameters necessarily achieve a faster computational speed than a standard Resnet 50.

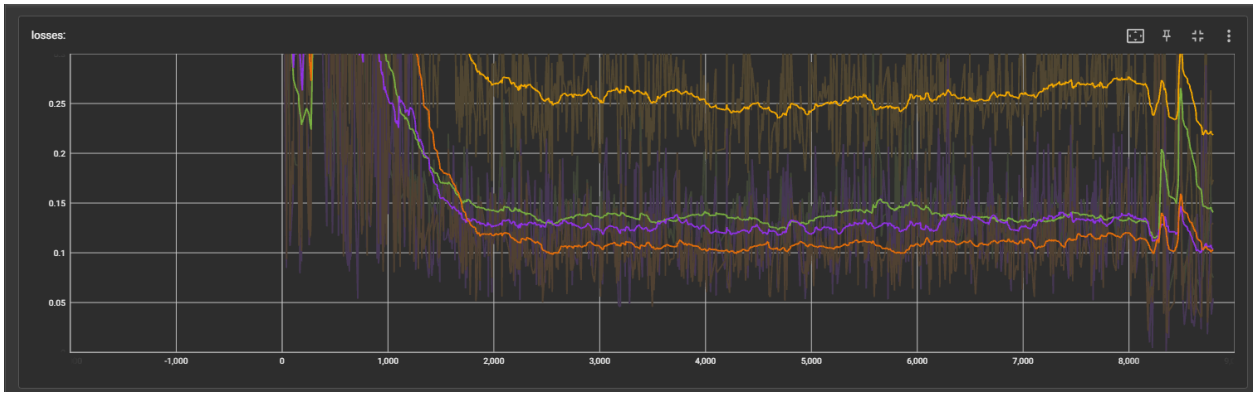
8-step-loss

Light Blue – Classifier Loss
Pink – Box Regression Loss
Yellow – RPN Classification Loss
Purple – RPN Box Regression Loss



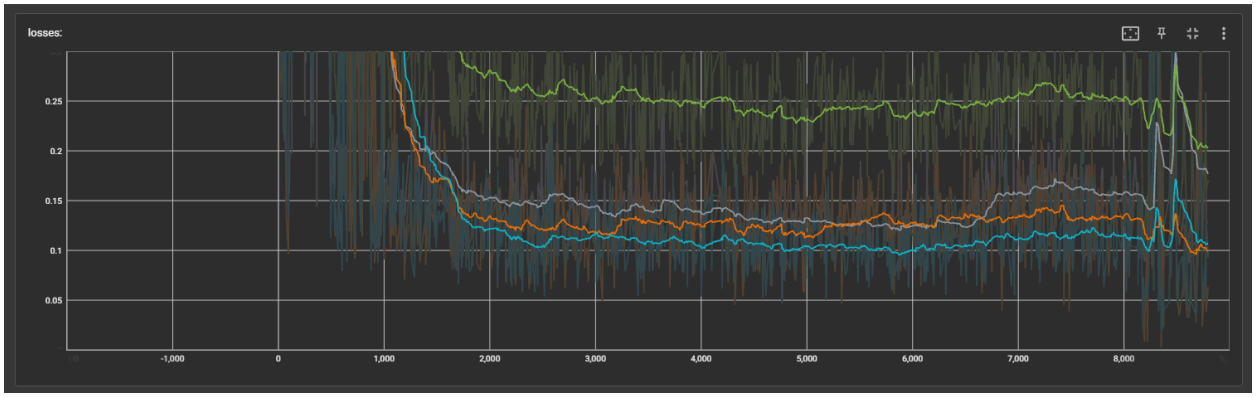
4 step loss

Yellow – Classifier Loss
Green – Box Regression Loss
Purple – RPN Classification Loss
Orange – RPN Box Regression Loss



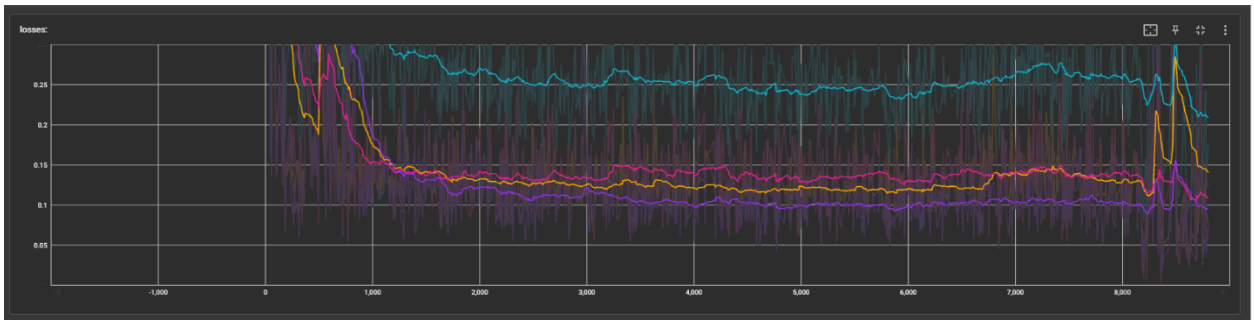
2-step loss:

Green – Classifier Loss
Gray – Box Regression Loss
Orange – RPN Classification Loss
Blue – RPN Box Regression Loss



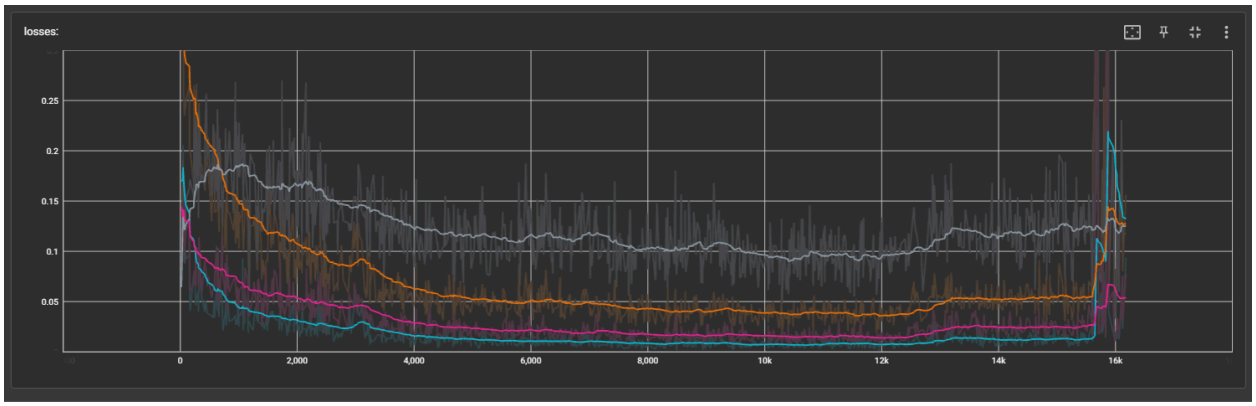
RK-4 loss:

Blue – Classifier Loss
Pink – Box Regression Loss
Orange – RPN Classification Loss
Purple – RPN Box Regression Loss



Resnet 50 Control:

Orange – Classifier Loss
Gray – Box Regression Loss
Blue – RPN Classification Loss
Pink – RPN Box Regression Loss



Time Varying (4 step Euler)

Purple – Classifier Loss
Yellow – Box Regression Loss
Orange – RPN Classification Loss
Green – RPN Box Regression Loss

As you can see, the Neural ODE backbones perform significantly worse (with about twice the loss) when compared to the Resnet 50 control. There is little variation in the actual losses or the convergence rate when changing the precision of the Neural ODE integration or the method of integration. This implies that the use of an ODE at all might be unnecessary, and that simple Independent Recurrence [cite IND RNN] may be enough. Our implementation of Time Varying Dynamics caused the loss to diverge for reasons not yet known. The fact that the 2-step Euler and 4-step Euler are faster than the baseline, shows the fundamental promise of faster (and thus more accurate in deployment) detection in streaming perception using ODE Feature Recurrence.

Methods:

The Nero Optimizer was used to optimize in all cases. Each model was trained for 4 epochs, with a learning rate of .025 which halved each epoch. Training was done on the Caltech HPC.

Discussion:

There are still further avenues for research in the realm of applying Neural ODE's for recurrence in streaming detection. For one thing, Faster RCNN is no longer state of the art. Currently the state of the art in image detection is called YOLO. This is a single stage detector, rather than the two stage detector of Faster RCNN. YOLO also uses a probability map of object classes that seems like it would fit rather well with the slow smooth variation of a Neural ODE. Another idea we had was to try to implement Neural ODE recurrence in the later parts of that Faster RCNN detector, specifically the ROI-Head. Another useful avenue of research could be to move beyond the use of ODEs entirely and focus on an independently recurrent approach [cite independently recurrent Neural Network].

Conclusion:

The Use of Neural ODEs for latent feature recurrence in streaming detection has a strong theoretical grounding in the fact that much of the information extracted from one frame to the next is redundant. Neural ODEs seem apt to take advantage of this proximity in state space to allow recurrence of these features, thus reducing the overall size and increasing the speed of the CNN backbone. In practice our results show that the proposed Neural ODE backbone detectors have significantly worse accuracy than the control, but make inferences at faster speeds. Further research is needed in this area to determine whether the accuracy can be improved and Neural ODEs can be applied to streaming perception.

Works Cited

Chen, Ricky T. Q., et al. “Neural Ordinary Differential Equations.” *ArXiv.org*, 14 Dec. 2019, https://arxiv.org/abs/1806.07366.

He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” *ArXiv.org*, 10 Dec. 2015, https://arxiv.org/abs/1512.03385.

Queiruga, Alejandro F., et al. “Continuous-in-Depth Neural Networks.” *ArXiv.org*, 5 Aug. 2020, https://arxiv.org/abs/2008.02389.

Rodriguez, Ivan Dario Jimenez, et al. “LyaNet: A Lyapunov Framework for Training Neural Odes.” *ArXiv.org*, 5 Feb. 2022, https://arxiv.org/abs/2202.02526.