A Appendix

A.1 Detailed descriptions of hGRU and cLSTM

Formulation of hGRU The hGRU cell and the following formulation are adapted from [1, 2] for the setting of RecSlowFast. The hGRU unit has hidden states $\mathbf{H}_t^n \in \mathbb{R}^{h\times w\times c}$ where t is the input timestep, n denotes the n_{th} recurrent step and $h \times w \times c$ give the height, width, and channel dimensions. $\mathbf{X}_t \in \mathbb{R}^{h\times w\times c}$ is the input feature to this hGRU unit. There are two gates $\mathbf{G}^S$ and $\mathbf{G}^F$, $\mathbf{G}^S$ is for suppression and $\mathbf{G}^F$ is for facilitation/excitation.

For the suppression stage, the hGRU does the following operations:

$$\mathbf{G}^S = \sigma(\mathbf{U}^S * \mathbf{H}_t^{n-1}) \tag{6}$$

$$\mathbf{C}^S = BatchNorm(\mathbf{W}^S * (\mathbf{H}_t^{n-1} \odot \mathbf{G}^S)) \tag{7}$$

$$\mathbf{S} = \left[\mathbf{X}_t - [\mathbf{C}^S \odot (\alpha \mathbf{H}_t^{n-1} + \mu)]_+\right]_+ \tag{8}$$

$$\tag{9}$$

For the facilitation/excitation stage:

$$\mathbf{G}^F = \sigma(\mathbf{U}^F * \mathbf{S}) \tag{10}$$

$$\mathbf{C}^F = BatchNorm(\mathbf{W}^F * \mathbf{S}) \tag{11}$$

$$\tilde{\mathbf{H}} = [\kappa \mathbf{S} + \gamma \mathbf{C}^F + \omega(\mathbf{C}^F \odot \mathbf{S})]_+ \tag{12}$$

$$\mathbf{H}_t^n = [(1 - \mathbf{G}^F) \odot \mathbf{H}_t^{n-1} + \mathbf{G}^F \odot \tilde{\mathbf{H}}]_+ \tag{13}$$

Convolution is denoted by *, $\odot$ denotes the Hadamard product, $[\cdot]_+$ denotes the softplus pointwise nonlinearities and σ is the sigmoid function. Detailed designed choices for the hGRU cell are in [1, 2].

The RecSlowFast-hGRU instantiation begins with a convolutional layer with kernel size 7 mapping the greyscale input to 25 channels. The resulting feature map is squared then followed by one layer of hGRU cell, which is the recurrent part of the network. The output of the hGRU is then batch normalized and fed into a convolutional layer with kernel size 1 and mapping 25 channels to 2 channels. For processing the next input feature $\mathbf{X}_{t+1}$, RecSlowFast-hGRU will forward the last hidden states $\mathbf{H}_t^{N(t)}$ as the initial hidden states for $t + 1$, namely $\mathbf{H}_{t+1}^0 = \mathbf{H}_t^{N(t)}$.

Formulation of cLSTM We follow the cLSTM used in [2, 3], each convolutional LSTM cell is as follow:

$$i = \sigma(W_{xi} * x_t + W_{hi} * h_t^{n-1}) \tag{14}$$

$$f = \sigma(W_{xf} * x_t + W_{hf} * h_t^{n-1}) \tag{15}$$

$$o = \sigma(W_{xo} * x_t + W_{ho} * h_t^{n-1}) \tag{16}$$

$$c_t^n = f \odot c_t^{n-1} + i \odot \tanh(W_{xc} * x_t + W_{hc} * h_t^{n-1}) \tag{17}$$

$$h_t^n = o \odot \tanh(c_t^n) \tag{18}$$

The RecSlowFast-cLSTM instantiation also begins with a convolutional layer with kernel size 7 mapping the greyscale input to 25 channels. The resulting feature map is squared then followed by one layer of cLSTM cell, which is the recurrent part of the network. The output of the hGRU is then batch normalized and fed into a tail convolutional layer with kernel size 1 and mapping 25 channels to 2 channels. Apart from the recurrent cell (cLSTM vs. hGRU), the two instantiations are having the exact same structure. The cLSTM cell states c_t^n and hidden states h_t^n are considered together as hidden states in the RecSlowFast framework and get passed between input timesteps, namely $(c_{t+1}^0, h_{t+1}^0) = (c_t^{N(t)}, h_t^{N(t)})$. The other notations for arithmetic operations are consistent with the hGRU formulation.

A.2 Detailed descriptions of baselines

FF-* networks FF-* stands for normal 2D convolutional feedforward networks. Every FF-* network has one head and one tail convolution layer. The head convolution layer has 25 kernels with kernel size 7, followed by batch normalization. The tail convolution has 2 kernels with kernel size 1, mapping multiple channels into 2 class feature maps, which will used for the calculation of the loss together with the ground truth.

The FF-6 and FF-8 networks consist of 6 and 8 convolutional layers between the head and tail. Each layer has 25 kernels with size 15. Each convolution layer is followed by a batch normalization and ReLU activation function. The FF-res-6 and FF-res-8 networks also have 6 and 8 convolution layers between the head and trail, each with 25 kernels with size 15. However, every two convolution layers are organized as a residual block. The input to the residual block is added to the output of the batch normalization output of the second convolution layer in the residual block, before the ReLU activation. All kernel sizes are matched up closely with other architectures in the experiments.

Conv3D-* networks The Conv3D-* baselines consist of three 3D-convolution layers as the head followed by five 2D-convolution layers. We control the spatial kernel size in each layer similar to the rest of all T-Pathfinder experiments (7 for the first layer, 15 for the rest). All 3D convolution layers use a kernel size 2 in the depth dimension. The two Conv3D variants, i.e., Conv3D-S and Conv3D-M net, only differ in the number of channels in the three 3D convolution layers ([8, 16, 16] vs [25, 25, 25]). The inputs to the network are not single frames but the concatenation of the last 4 frames, thus the model is given the ability to use past information.

A.3 More results on *T-Pathfinder*

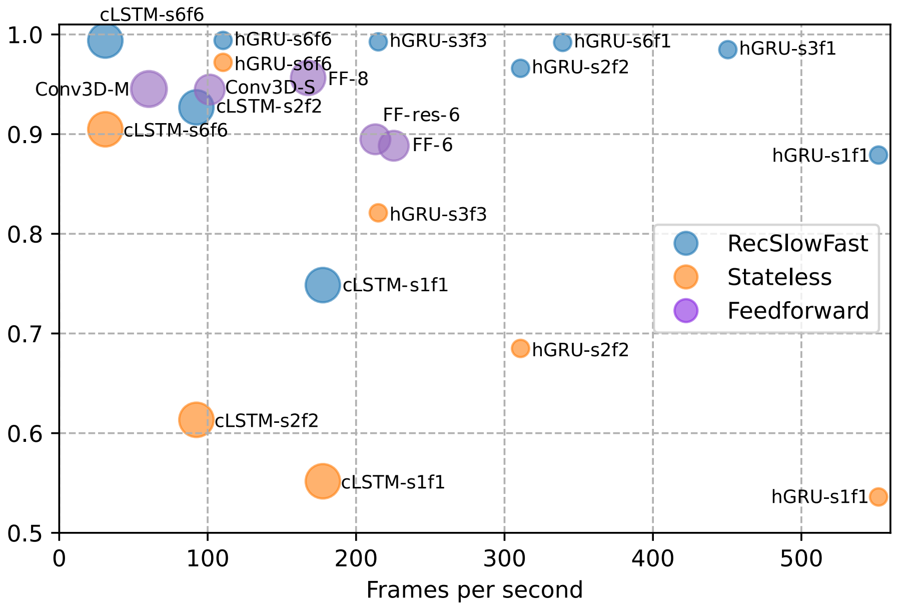

Figure 6: Different RecSlowFast schedules with hGRU and cLSTM instantiations, compared with their Stateless counterparts and feedforward CNN baselines. Circle sizes represent the number of parameters. Detailed descriptions for feedforward baselines FF-* and Conv3D-* are in Appendix A.2. The dataset studied is *T-Pathfinder-Easy*.

T-Pathfinder-Easy												
hGRU	s6f6			s4f4			s2f2			s1f1		
	mean	std.	FPS	mean	std.	FPS	mean	std.	FPS	mean	std.	FPS
RecSlowFast	.994	.001	110	.994	.002	164	.966	.003	315	.879	.011	561
Stateless	.972	.002		.919	.010		.685	.008		.536	.004	
hGRU	s6f6			s6f4			s6f2			s6f1		
	mean	std.	FPS	mean	std.	FPS	mean	std.	FPS	mean	std.	FPS
RecSlowFast	.994	.001	110	.993	.001	152	.994	.002	241	.992	.002	342

Table 3: Comparison between RecSlowFast-hGRU and Stateless-hGRU with the same schedule and the effect of reducing recurrent steps on consecutive frames when hidden states are recycled.

More comparisons on *T-Pathfinder-Easy* Figure 6 and Table 3 give a more detailed comparison among RecSlowFast, Stateless, and feedforward CNN baselines. The trend of RecSlowFast always outperforming the Stateless version with the same train/inference schedule is shown for both hGRU and cLSTM instantiations. And reducing the number of fast steps for RecSlowFast down to s6f1 well preserved the task accuracy and boosted the inference speed. In addition to having the number of slow steps with 6, we also experimented s3f1 schedule, which maintained a good mIoU while further reducing the computation cost.

T-Pathfinder-Easy						
	RecSlowFast-hGRU s6f1	FF-8	Conv3D-S	DEQ-hGRU		
				$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
mIoU	.992	.956	.947	.955	.955	.955
FPS	342	168	105	140	123	65
parameters	284k	1127k	827k	284k	284k	284k

Table 4: Comparison between RecSlowFast-hGRU, DEQ-hGRU and other baselines.

Comparison with deep equilibrium models We have implemented a baseline using the DEQ layer (DEQ-hGRU) based on the publicly available code from [4]. For DEQ-hGRU, the recurrent hGRU cell is replaced with an implicit DEQ layer. In order to maintain training stability, we had to modify the architecture by adding an additional batch normalization layer before the DEQ layer. We tried both Broyden and Anderson solvers [4] and used the relative difference of hidden states between two iterations of the fixed point solver as stopping criteria (threshold θ). The Broyden solver performed worse than the Anderson solver so we omitted the results here. We use maximum 10 iterations and $\theta = 0.01$ in training. In both training and testing, state reuse was used. A preliminary study with different inference θ (θ affects the tradeoff between inference speed and accuracy) is shown in Table 4. From the preliminary results on the T-Pathfinder-Easy dataset, the RecSlowFast-hGRU-s6f1 performs the best in both mIoU and FPS.

The root-finder solvers for the deep equilibrium models require a typically quadratic time and memory complexity w.r.t the dimension of the hidden state, making it not very compatible with layers resulting in large feature map dimensions (such as convolution). In contrast, RecSlowFast does not incur the additional budget from using a fixed point solver. This gives us more flexibility in the choice of class of layers and better compatibility with modern neural network hardware accelerators.

A.4 PCA details

For creating the PCA trajectory in Figure 4(a), we adapted the PCA analysis in [2] and implemented the following steps: (1) We first fetch the best-performing checkpoint of RecSlowFast-hGRU-s8f8 trained on *T-Pathfinder-Hard*, then extract hidden states on the test split also using the schedule of s8f8. Thus, for each input test frame, there will be 8 hidden states. Together with the initialization, there are 65 hidden states in total for every sequence, each of size $128 \times 128 \times 25$, where 25 is the number of channels and 128×128 is the spatial size. (2) All hidden states are then averaged along the channel axis and flattened, resulting in 128×128 dimensional vectors. (3) The dimension is reduced with PCA to 2 dimensional for visualization.

A.5 DRU network for CamVid semantic segmentation

For the DRU experiment, we use a U-Net architecture with a convolutional DRU network same as in [5]. The DRU network has a U-Net-like bottleneck structure. The encoding stage consists of 4 convolutional layers followed by batch normalization and maxpooling. The four convolutional layers have 16, 32, 64, and 128 filters respectively. The convDRU unit, which works as the bottleneck block, maps the features from 128 channels to 256 channels. The decoder part of the network reduces the channels with 4 transposed convolutional layers which maps the number of channels to 128, 64, 32, and 16 consecutively. The decoder is then followed by a convolutional layer that maps the 16 channels to the number of semantic classes of channels. We denote the output of the last layer as S_t^n in each recurrent step of input timestep t. The input to the entire DRU network is the original input frame concatenated with the output S_t^n, namely $[I_t, S_t^n]$.

The formulation of the convDRU cell, used as the bottleneck, adapted from [5] is as follows:

$$r = \sigma(W_r * x_t^{n-1}) \tag{19}$$

$$z = \sigma(W_z * x_t) \tag{20}$$

$$\tilde{h} = \tanh(W_t * (r \odot x_t^{n-1})) \tag{21}$$

$$h_t^n = z \odot h_t^{n-1} + (1 - z) \odot \tilde{h} \tag{22}$$

where W_r, W_z, W_t are convolutional kernels. x_t^{n-1} is the input features to the convDRU cell. Notice x_t^{n-1} comes with a recurrent step in the superscript, since the input feature is also dependent on the output of the entire network S_t^n. In the RecSlowFast-DRU network, the states reused cross input timesteps are h_t^n and S_t^n.

A.6 T-Pathfiner examples

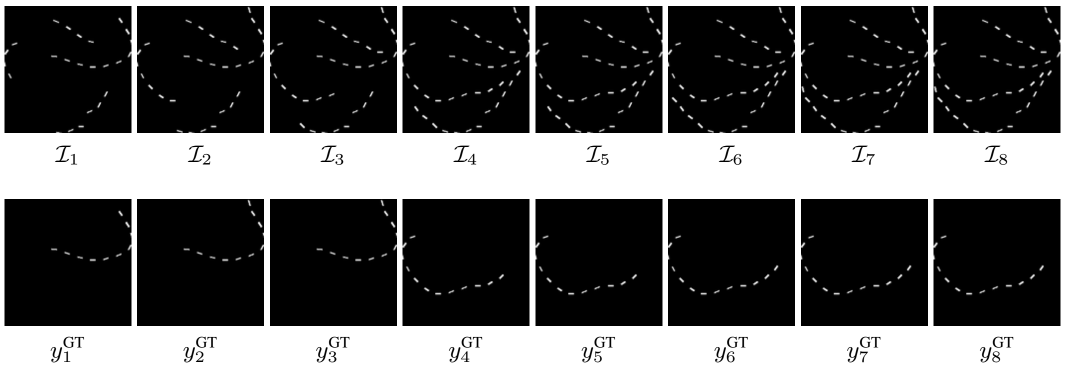

Figure 7: *T-Pathfinder-Hard* example used in Figure 2.

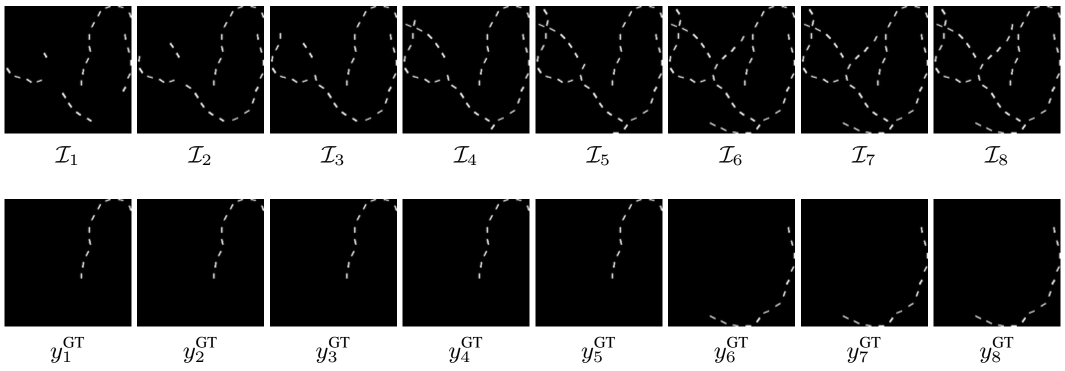

Figure 8: *T-Pathfinder-Hard* example used in Figure 4(b). Notice the lower right two distractors visually merged starting from $\mathcal{I}_2$. Stateless models are likely to misinterpret them as from the same contour.

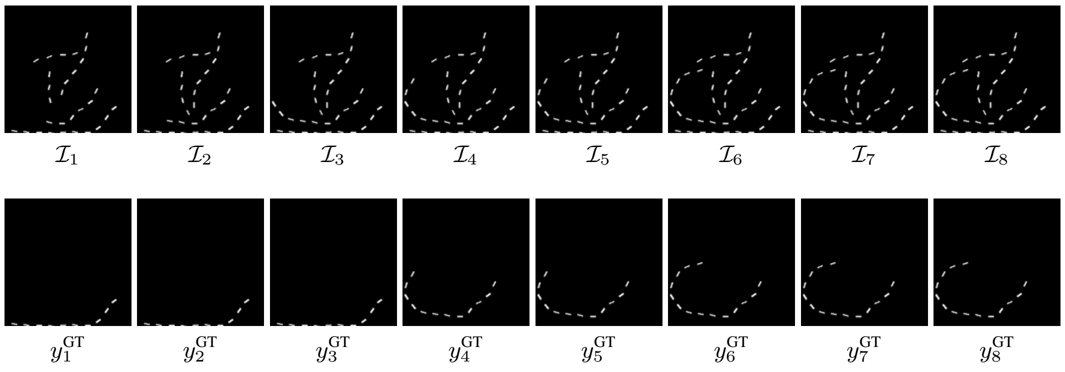

Figure 9: *T-Pathfinder-Hard* example.

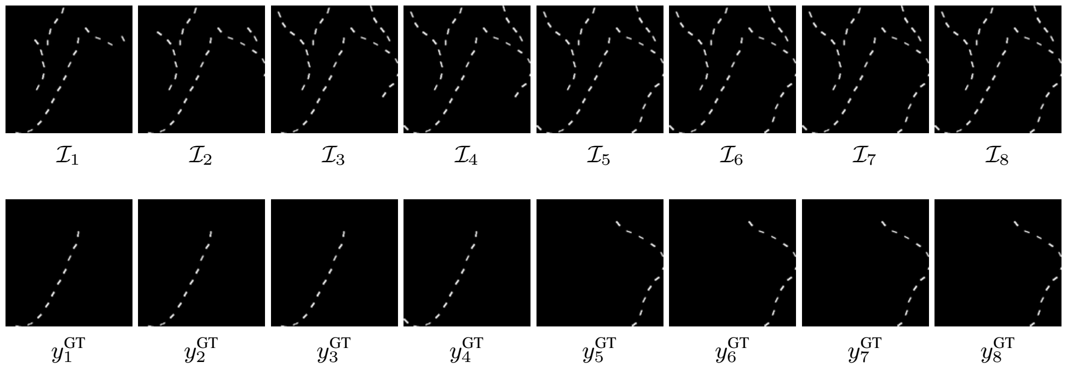
$\mathcal{I}_1$ $\mathcal{I}_2$ $\mathcal{I}_3$ $\mathcal{I}_4$ $\mathcal{I}_5$ $\mathcal{I}_6$ $\mathcal{I}_7$ $\mathcal{I}_8$
y_1^{GT} y_2^{GT} y_3^{GT} y_4^{GT} y_5^{GT} y_6^{GT} y_7^{GT} y_8^{GT}

Figure 10: *T-Pathfinder-Hard* example.

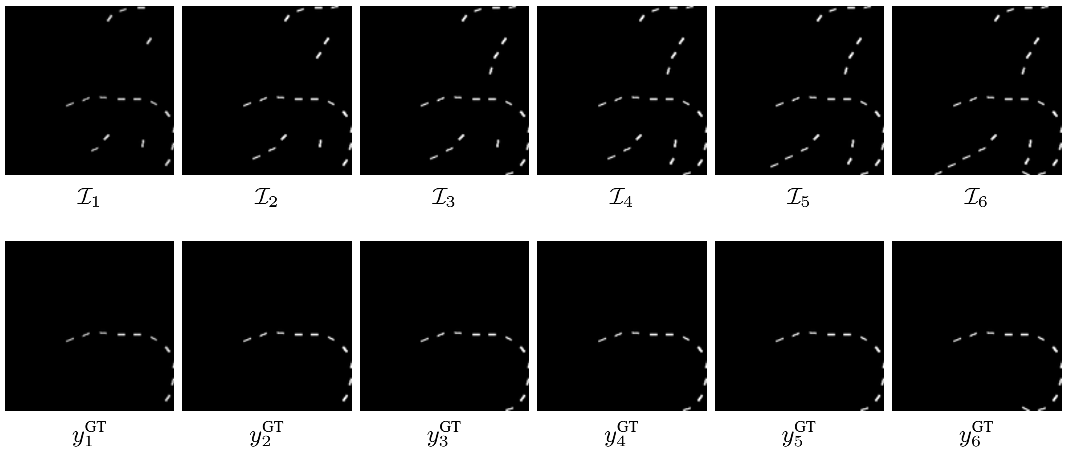
$\mathcal{I}_1$ $\mathcal{I}_2$ $\mathcal{I}_3$ $\mathcal{I}_4$ $\mathcal{I}_5$ $\mathcal{I}_6$
y_1^{GT} y_2^{GT} y_3^{GT} y_4^{GT} y_5^{GT} y_6^{GT}

Figure 11: *T-Pathfinder-Easy* example used in Figure 2.

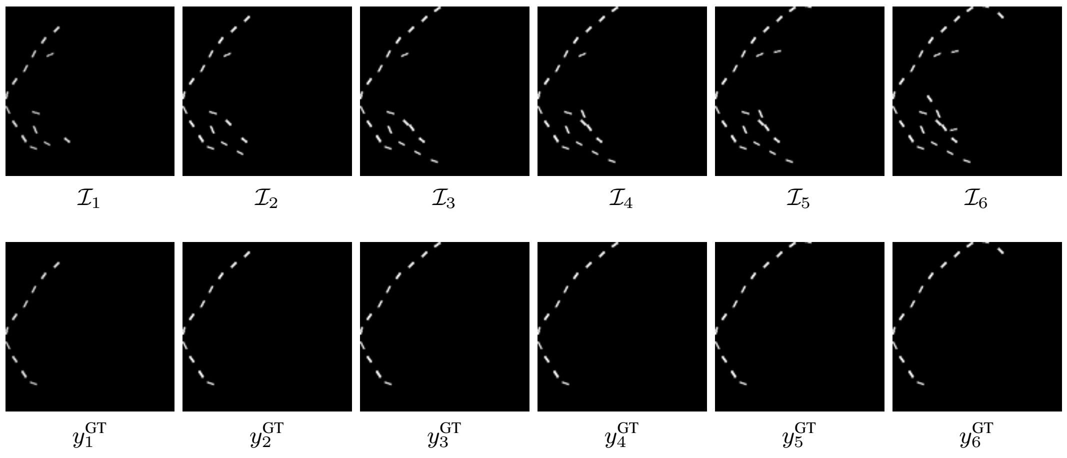
$\mathcal{I}_1$ $\mathcal{I}_2$ $\mathcal{I}_3$ $\mathcal{I}_4$ $\mathcal{I}_5$ $\mathcal{I}_6$
y_1^{GT} y_2^{GT} y_3^{GT} y_4^{GT} y_5^{GT} y_6^{GT}

Figure 12: *T-Pathfinder-Easy* example.

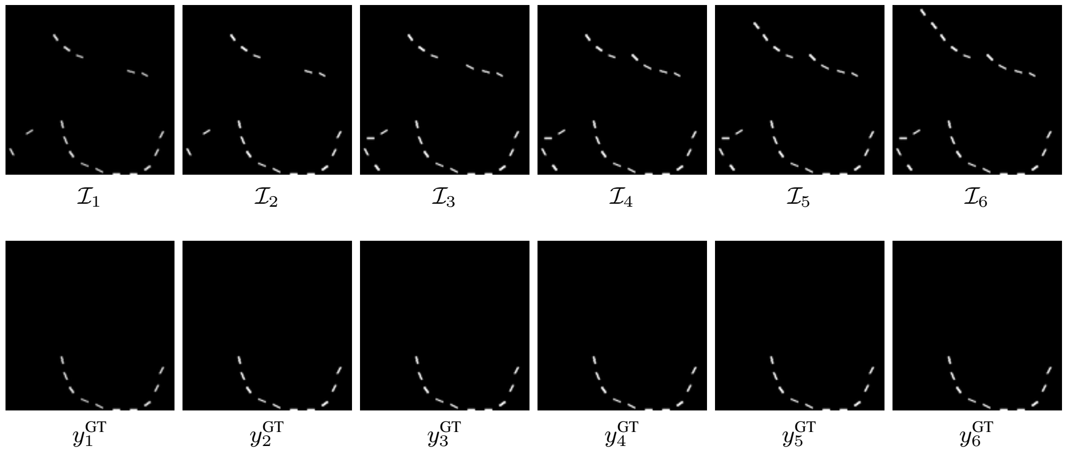

Figure 13: *T-Pathfinder-Easy* example.

References

[1] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/ec8956637a99787bd197eacd77acce5e-Paper.pdf`.

[2] Drew Linsley, Alekh Karkada Ashok, Lakshmi Narasimhan Govindarajan, Rex Liu, and Thomas Serre. Stable and expressive recurrent vision models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 10456–10467. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/766d856ef1a6b02f93d894415e6bfa0e-Paper.pdf`.

[3] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=B1ewdt9xe`.

[4] P. Micaelli, A. Vahdat, H. Yin, J. Kautz, and P. Molchanov. Recurrence without recurrence: Stable video landmark detection with deep equilibrium models. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22814–22825, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society. doi: 10.1109/CVPR52729.2023.02185. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR52729.2023.02185`.

[5] Wei Wang, Kaicheng Yu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. Recurrent u-net for resource-constrained segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2142–2151, 2019. URL `https://ieeexplore.ieee.org/document/9010910`.