Video Prediction

Independent Work Report (MAE 340, Spring 2021)
Advisor: Professor Olga Russakovsky

Matthew Coleman

April 26, 2022

This project represents my own work,
in accordance with the University regulations.
/s/Matthew Coleman

Contents

1 Introduction

While humans cannot perfectly predict the future, they are indeed capable of inferring a great deal of information about near events in the future, and this knowledge greatly aids them in planning out their actions, such as which movements to take to reach a goal. This ability to forecast the future is a direct result of an understanding of causality that is learned through observation and interaction [3].

Many human predictions are erroneous in major respects, but even humans least adept at inferring far-off outcomes and consequences still are masters of learning very near-term ones. For example, people generally have a good sense for where a car will move in the street, or which direction a pedestrian may continue walking. Even a young child can predict where to toss a football to a moving receiver, and even this small knowledge reveals an infinite wisdom compared to the most advanced video prediction methods.

The task of video prediction is comprised of several open challenges in computer vision; it uses some of the most recent machine learning model architectures that have been developed and it even contends directly with an impossible task altogether, which is to predict the future. Although it is a particularly confusing task, it also has the potential for immense impact and immediate practical applications, such as in autonomous driving [12], video interpolation [15] and most interesting in the context of this report, robotic control systems [9].

This project will examine the current state-of-the-art in video prediction machine learning models, report on several experiments carried out by implementing and testing such a model on various existing datasets with varying hyperparameters, and attempt to make meaningful conclusions about video prediction based off of these results.

2 The Task of Video Prediction

The task of video prediction is to construct an approximation for the completion of a sequence of frames using only the initial sequence of frames. Formally, given an ordered set of n image frames $\boldsymbol{X} = (X_1, X_2, X_3 \cdots X_n)$, the task is to predict the immediately following m frames of the sequence $\boldsymbol{Y} = (Y_1, Y_2, Y_3 \cdots Y_m)$, each of which having the same dimensions, for example with c channels, height h, and width w. At each inference in training, the model predictions $\hat{\boldsymbol{Y}} = \left(\hat{Y}_1, \hat{Y}_2, \hat{Y}_3 \cdots \hat{Y}_n\right)$, with the same dimensions as the inputs, are conditioned on the input sequence $\boldsymbol{X}$, and the model weights are updated typically by the gradient of a loss function computed between the predictions and ground truth sequence $\boldsymbol{Y}$ directly. Critically, since there is no human intervention or labeling required for the model to do this, and models typically are able to learn from the implicit temporal organiziation of the video data, video prediction is a self-supervised task [16].

3 Families of Prediction Models

Modern video prediction models tend to adopt several canonical architectures, which are normally simple and easily generalizable for specific tasks, such that they can be used as building blocks or blueprints within larger architectures. Understanding what each architecture seeks to do on a high level is imperative to understanding what kind of output one should expect from the model or how to debug and improve it, and research implementations make use of these basic concepts in different ways.

For example, RNNs and generative networks are each successful and well-tested paradigms used in many other machine learning domains outside of computer vision, but they are especially utilized within video prediction because of their key properties and strengths, particularly of RNNs to work on time-sequential data and of generative networks to "imagine" new data within a distribution. These paradigms are used extensively in Convolutional LSTM (ConvLSTM), FutureGAN [2], and SAVP models [14] for video prediction, as well as many other models in cutting-edge research. A short description of each family and its relevance to video prediction is given below:

3.1 Recurrent Neural Networks

Formally, a Recurrent Neural Network (RNN) is the end-result of a mathematical analysis of a nonlinear first-order non-homogeneous ordinary differential equation describing the evolution of a state signal s as a function of time, along with an input signal x. The canonical statement of an RNN is given below in the form of a system of discrete Delay Differential Equations (DDE) [20]:

$$\begin{aligned} s_t &= W_s s_{t-1} + W_r r_{t-1} + W_x x_t + \theta_s \\ r_t &= G(s_t) \end{aligned} \tag{1}$$

With W_s, W_r, and W_x as weights which are either multiplied or convolved with their respective signals, and θ_s as a bias term which is typically added in element-wise fashion to s. This equation also includes the read-out or output signal r, which is the activation $G(z)$ of the state signal, and can be seen as the "output" of the network. All together, this equation describes all of the moving parts of an RNN.

As a toy example (and ignoring some technical tricks that would be required to make this actually happen), a network of this type should be capable of transcribing seqeunces of a lecture by evaluating discrete audio samples recorded from the event and outputting the spoken words in text form [20]. Another extremely common application of vanilla RNNs is machine translation, in which the input sequence is text in one language and the output is meant to be the translation of that text into another language.

A more easily understandable depiction of the RNN described in Equation 1 can be found in Figure 1. This figure also shows the "unfolding" or "unrolling" of the model, which is just a way of seeing each time-step of the state signal s_t, input signal x_t, and ouput signal r_t (here replaced by $\hat{y}_t$, denoting the network inferences) as they are produced over time.

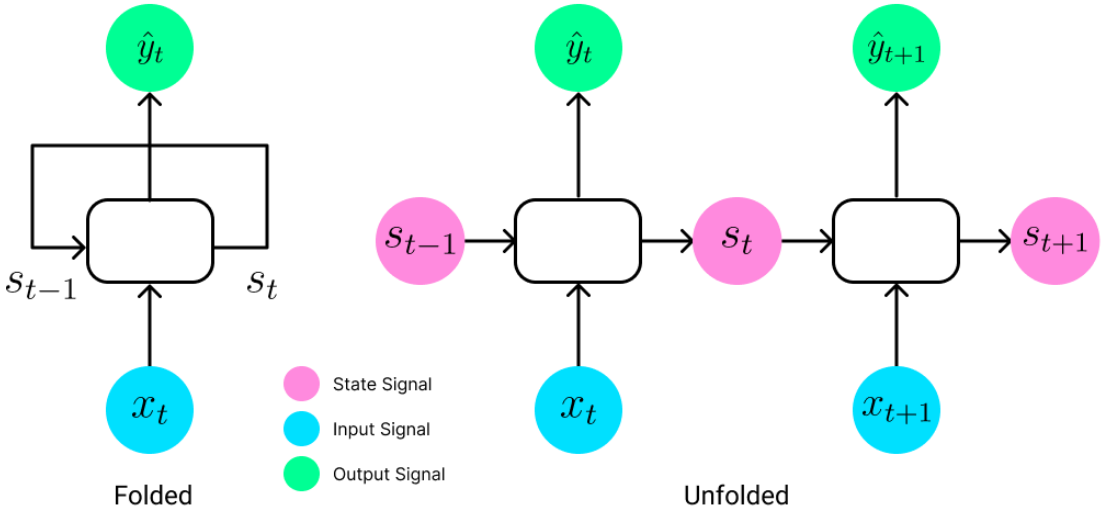

Figure 1: RNN Architecture

RNNs are generally trained using a techinque called Backpropagation Through Time (BPTT), in which the gradient of a loss function taken between model outputs and ground truth labels at a certain instance in time is used to update the model parameters recursively through the history of the model, using the chain rule. In this way, RNN models trained over long sequences have their parameters updated by the product of many Jacobian matrices, which, just like a product of many real numbers, can vanish or explode very easily [17]. For this reason, along with several others having to do with the long-term stability of RNNs, the Long Short-Term Memory (LSTM) cell was developed with nonlinear, data-dependent controls in the form of several "gates" that control input to and output from the cell's state [20].

3.2 Long Short-Term Memory (LSTM) Cell

The changes described in the following section take the form of a modified system of equations [23]:

$$\begin{aligned} f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f) \\ i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \\ \tilde{c}_t &= \tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}) \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{2}$$

The key differences in the LSTM are the four gates denoted by f, i, $\tilde{c}$, and o. Other than these, the cell state c and output h are analogous to the original RNN definition.

Now, the input and output gates are able to "learn" how to add information to the cell state or take information for inferences, and the forget gate is able to remove information from the cell state entirely. To see how this is done, recall that the sigmoid funtion $\sigma(z)$ has an output range of $(0, 1)$, meaning that element-wise multiplication by embeddings activated by sigmoid can minimize and essentially nullify the cell state. In the input gate, this same technique is applied for i onto $\tilde{c}$, and in the output gate it is applied by o onto c, meaning that i decides what from h_{t-1} is allowed into the cell state and o decides what from c_{t-1} is allowed out of the cell state into the inference. A diagram of these connections is shown in Figure 2.

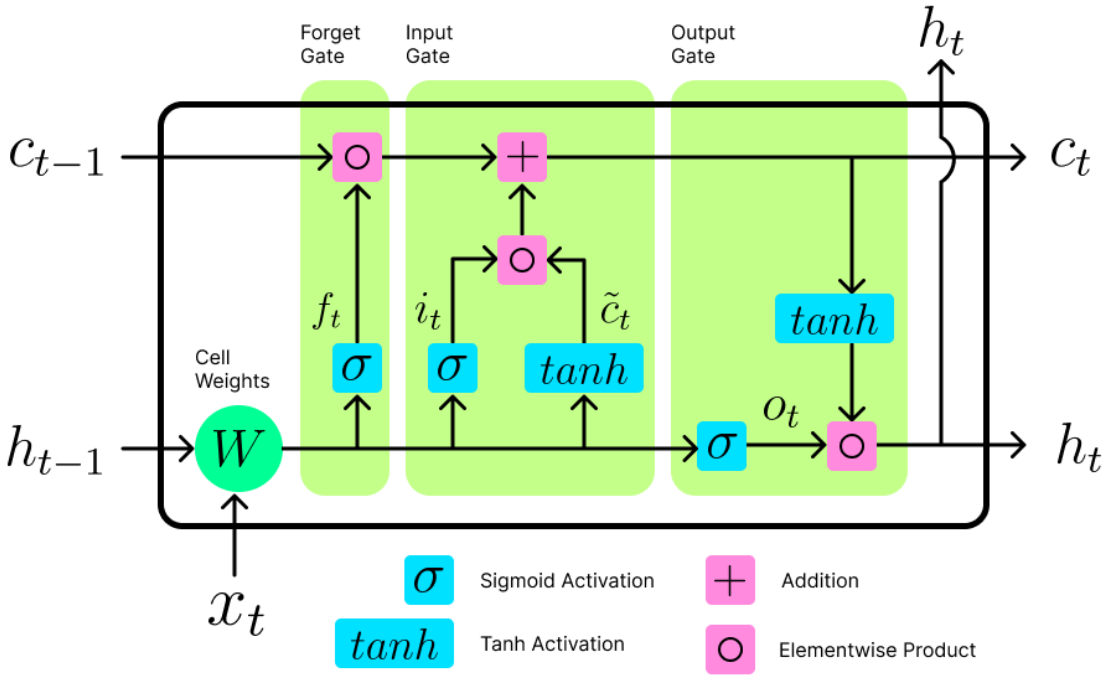

Figure 2: LSTM Cell Architecture

In this way, LSTM cells are capable of mitigating some of the drawbacks of RNNs such as vanishing and exploding gradients, and they are capable of learning how to make good inferences for very different sequences. In other words, they are better able to approximate both $y(t_1)$ and $y(t_2 \gg t_1)$ using the same parameters.

Now that some of the general architectures and methodologies are laid out, however, the question still remains of how these models actually predict the continuation $\boldsymbol{Y}$ of a sequence of frames $\boldsymbol{X}$ with any sort of accuracy. In order to do this, one final methodology must be examined, namely Sequence to Sequence (Seq2Seq) learning, which will lead to the final implementation of a Convolutional LSTM used for experimentation in this project.

3.3 Sequence To Sequence Learning

Seq2Seq learning is only a slight modification to the original usage of RNNs, but they represent an elegant solution to a classical shortcoming of most Deep Neural Networks (DNNs), which is that they can only be applied to problems in which inputs and outputs can fit into fixed, discretized, vectors, and therefore must be of a certain length or size. Seq2Seq learning solves this problem for video prediction as well as many other tasks by using 2 LSTMs: an encoder to read the sequence and learn to create an embedding vector, and a decoder, which is passed the embedding vector and learns to generate the predicted sequence [22]. A diagram of this procedure is shown in Figure 3.

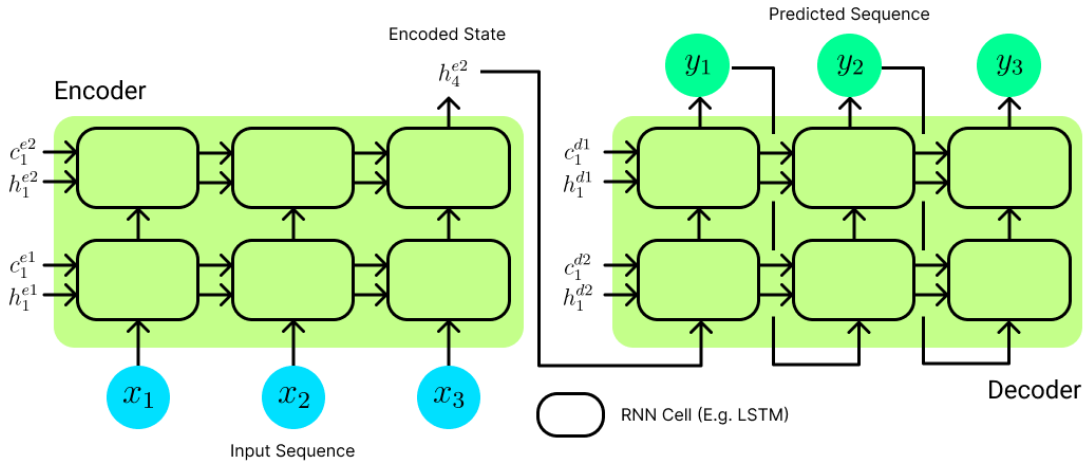

Figure 3: Seq2Seq Architecture

The diagram shows how the input sequence x is fed into the encoder LSTM for 3 steps (in practice this can be any length), how the encoder generates an embedding vector h_4 which is passed to the decoder as an input, and how the decoder generates the predicted sequence by passing the predicted output $\hat{y}_t$ back into itself at the next time step as an input, continuing until all predicted frames are generated.

This diagram also shows two layered LSTM cells, which is another key modification used in this project's implementation. Note that each layer has it's own states h and c, and at each time step, the cells pass their h embedding up to the next cell as an input. Deep, multilayered LSTMs have been shown to significantly outperform shallow LSTMs for language translation [22], however, this project will also analyze the effect of depth on LSTM inferences for video prediction.

In sum, these are the main methods used in this report's implementation of an LSTM model for video prediction, which is discussed in further detail in the following section.

4 Implementation and Training Details

Implementation of all model architecture discussed in this report (which can be found in This GitHub repository) was carried out using PyTorch [18], with other utility code for model training, saving, and organization done using PyTorch-Lightning [8]. Code for the each video dataset `DataLoader` class was sourced from the GitHub repository for the implementation of the paper Stochastic Video Generation with a Learned Prior [6]. (This was for convenience and to ensure that data sampled in this project would be the same as the data used in common video prediction implementations.) The three `DataLoader` instances used in the linear sequence prediction experiments are described in the following section.

The implementation the LSTM cell matches the details described in the previous sections, with the addition of a final layer which acts on the collected predictions $\hat{\boldsymbol{Y}}$ in order to map them back into the original input channel dimension. For the LSTM with linear layers, this was another linear layer with an input dimension of 64 (the number of hidden channels used in all models) and an output dimension of 1. For the ConvLSTM, this was a 3D convolution with a kernel size of $(3, 3, 3)$, 'same' padding, and an output channel dimension of 1 or 3, depending on whether the dataset used was black-and-white or in RGB color.

Each model was trained using Stochastic Gradient Descent (SGD) with a learning rate of 0.001 on batches of size 4 for the video datasets and 64 for the sequence datasets. The input sequence length and predicted sequence lengths for the video and sequence datasets were 10 and 10 frames, and 25 and 25 points, respectively.

5 Datasets

5.1 Sequence Datasets

The main experimental procedure carried out in this report is the training and testing of the Convolutional LSTM model on the MovingMNIST, KTH, and BAIR datasets, however, in order to more gain a more robust understanding of LSTM features and limitations, as well as to debug the training mechanisms in a much simpler setting, it was useful to first test a Linear LSTM model on one-dimensional sequential data before moving fully to video prediction. Three datasets of this type were implemented: a dataset consisting of generated sin waves with random frequency and phase offset, a dataset consisting of NASDAQ close price data from 8 tech companies, and a dataset consisting of randomly generated points. The choice of these datasets were intended to each test the model with a varying degree of stochasticity, the sin waves being the most deterministic, or predictable, the random points being the most stochastic, or random, and the stock price data hopefully being somewhere in between. Some specific details about these datasets are described in the following sections.

5.1.1 GeneratedSins

This dataset was generated by simply plotting a sinusoid function:

$$\boldsymbol{X}(t) = \boldsymbol{Y}(t) = \frac{1}{2}(\sin(\alpha t + \beta) + 1) \tag{3}$$

With random α and β. This results in a sin wave with a random frequency and phase offset plotted within the range $[0, 1]$. Although this dataset has random features, once the sin wave is detected and learned by the model, it should be relatively easy to predict, since sin waves are perfectly periodic. $\boldsymbol{X}$ consists of 25 data points evaluated from $[0, 0.5)$, and likewise, $\boldsymbol{Y}$ consists of 25 data points from $(0.5, 1]$. In sum, the train split consists of 22000 samples, and the test set consists of 7000.

5.1.2 GeneratedNoise

This dataset was generated by using pytorch's random tensor function `torch.rand()`, which generates a tensor of specified dimensions, where each entry is a float uniformly distributed within $(0, 1)$. To make this data a little easier to view, this data was then smoothed using `scipy.ndimage.filters.gaussian_filter1d()` function, which convolves the signal with a 1-dimensional gaussian kernel with $\sigma = 1$. The train split consists of 22000 samples, and the test set consists of 7000.

5.1.3 Stocks

This dataset was sourced using Alpha Vantage [1], a free online stock price API. Close price data is provided from the past two years for the NASDAQ tickers 'GOOGL', 'MSFT', 'TSLA', 'AAPL', 'AMZN', 'NVDA', 'FB', and 'AMD'. This data is normalized

by subtracting the minimum and dividing by the maximum for each sequence individually, such that each sequence pulled from the dataset lies within $(0, 1)$, and is therefore comparable to the other datasets. The train split consists of 22277 samples, and the test set consists of 7312 samples.

5.2 Video Datasets

The video datasets chosen in this report are MovingMNIST [21], KTH [19], and BAIR [7]. All datasets were downsampled to 64×64 frames, with MovingMNIST and KTH having one channel dimension and BAIR having three. Each sequence sampled from these datasets was trimmed to 20 frames total, with the first 10 being the input to the model and the last 10 being the ground truth label, representing the duration the model was tasked to predict.

The MovingMNIST consists of a pair of white MNIST [5] bouncing around over a pure black background. The digits in the train split of MovingMNIST are chosen randomly from the training MNIST digits split, and likewise for the testing split.

The KTH dataset consists of one person (or zero) against a mostly solid background (either indoors or outdoors) performing one of six actions (walking, jogging, running, boxing, hand waving and hand clapping).

The BAIR dataset consists of a robot arm pushing around a collection of children's toys across a table top.

6 Experiments

When researching and implementing a prediction model of any sort, it is important to consider that certain sequences are simply impossible to predict, and that even if the model were "perfect" by human standards, it would still fail to perform this task perfectly. The interesting question here is not whether the model will be able to achieve a certain accuracy in predicting the sequences but rather where or how exactly will it fail? And where it does fail, which features of the model's architecture and methodology are to blame?

All these questions lead directly into one of the major concerns in video prediction research, which is that the task is inherently hard to judge [16], particularly that pixel-wise loss functions, such as the Mean Squared Error (MSE) function used to train this model, cause models to prefer blurry results that average out multiple possibilities for the next time step rather than a single, clearer image that could be very wrong using pixel-wise metrics. Models trained in this way are not directly learning to produce images but merely to appease the loss function, and these results may not be as clear or useful to humans, and are therefore not preferred in research discussion.

6.1 Sequence Prediction

The MSE loss was recorded as the model was trained, and a plot of this is shown in Figure 4. In addition, after training was complete, a plot was generated of the model's average loss on the test split as the predicted sequence progresses, which is shown in Figure 5. The average loss over the whole test split is shown in Table 1.

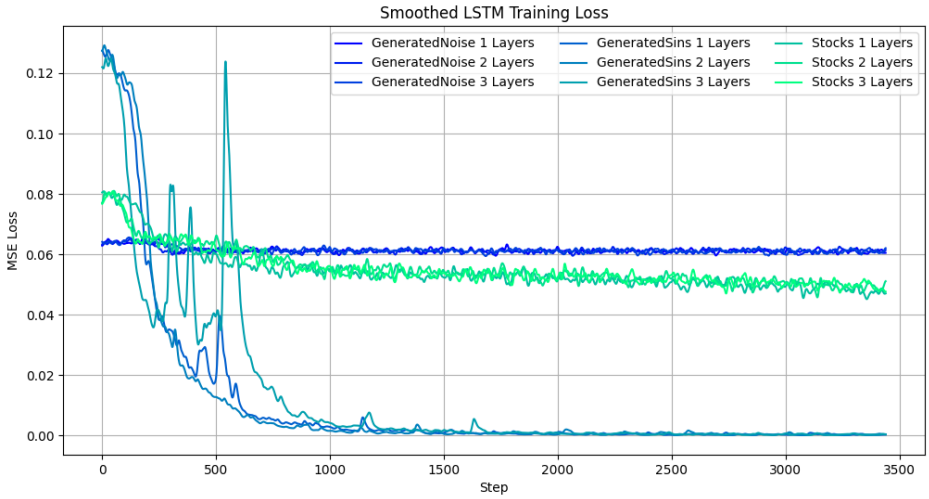

Figure 4: LSTM Training Loss on GeneratedSins, GeneratedNoise, and Stocks Datasets for Models with Varying Number of Cell Layers

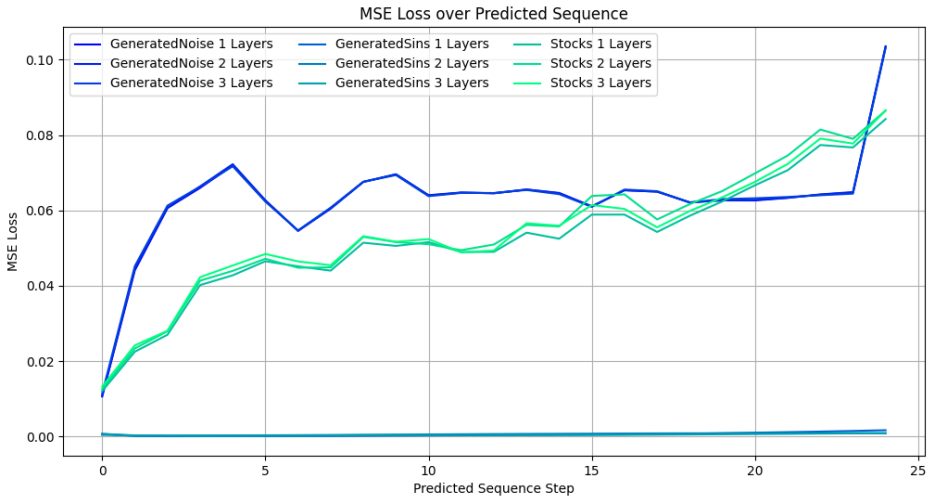

Figure 5: LSTM Test Loss over Predicted Sequence Steps for GeneratedSins, GeneratedNoise, and Stocks Datasets for Models with Varying Number of Cell Layers

Table 1: LSTM Average Test Losses vs. Number of Layers

Layers	**GeneratedNoise**	**GeneratedSins**	**Stocks**
1	5.964×10^{-2}	4.950×10^{-4}	5.227×10^{-2}
2	5.967×10^{-2}	6.298×10^{-4}	5.431×10^{-2}
3	5.961×10^{-2}	4.804×10^{-4}	5.381×10^{-2}

Additional test inferences for each dataset are shown in the Appendix Section 9.1.

6.1.1 GeneratedSins

The results shown in the previous figures confirm this dataset to be the most predictable of the three. This can be observed in the plot of training losses in Figure 5, which shows that models trained on GeneratedSins data generally decreased the most over training out of the three, and in the plot of average test losses over the predicted sequence in Figure 4, which shows that GeneratedSins led to the most accurate predictions over longer sequences by far. Additionally, Table 1 shows that the average test loss from this dataset was only 10^{-2} times the average losses of the others. A Test inference of a 2-layer LSTM model trained on this dataset with the ground truth sequence underlaid is shown in Figure 6.

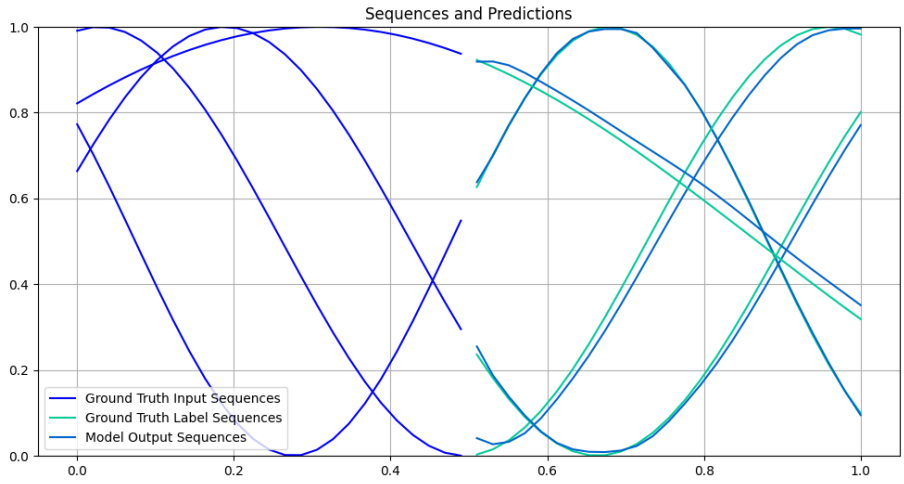

Figure 6: 4 LSTM Inferences on Test Split of GeneratedSins Dataset

As clean and impressive as these predictions are, however, they are not perfect, and still decrease in accuracy as the sequence progresses. While the errors are minor in this demonstration, the shortcoming reveals a pretty major property of the Seq2Seq architecture (and all sequence prediction architectures, not just Seq2Seq), which is that in general, predictions will tend to decrease in accuracy as they become longer. In most situations, this is because real data is at least some part stochastic, meaning that, as time goes on, a sequence can have many possible outcomes, and the state of the sequences at any far-off point in time is a composition of all the previous random events that take place. In this situation, however, a feature of the Seq2Seq modeling is most likely the culprit (excluding possible training and implementation errors), since there is *no* inherent stochasticity of a sin wave, that is, it is not a random process at all and once it is established at any point in time it will continue in that way forever.

In this situation, when the model feeds each LSTM cell's output directly back into itself as an input for the next time step in order to generate the full sequence (See Figure 3), if any prediction is off by a tiny amount, then the next prediction will likely share and even increase that error. While it is likely that this is the phenomenon causing the majority of the error in this situation, the small initial errors are most likely due to incomplete convergence of the model, since the training and test set should theoretically approach the same distribution as more samples are drawn.

6.1.2 GeneratedNoise

This dataset, in contrast, was nearly impossible for the model to learn satisfactorily, as evidenced by the training loss plot shown in Figure 4 which is essentially flat. Additionally, over the test sequences on average in Figure 5, the model increases most rapidly in error as the prediction increases in duration. While this result is somewhat expected for this

dataset, it is still concerning that the model is capable of predicting any amount of a random signal for any duration. The answer to this question lies in the details of the dataset implementation, specifically that the points are not purely random after all; they become slightly cross-correlated after being smoothed by the gaussian filter, due to the convolution action which sums neighboring points. A batch of test inferences showcasing this phenomenon is shown in Figure 7.

As for the model implementation and architecture, one important aspect of the results on this dataset stands out, which is that the model predicts nothing but a flat line centered at 0.5 after these initial predictions. This may seem like an uninteresting result, but what it shows is fundamentally the same issue that will cause blurry predictions in actual video predictions, and this is one of the current challenges discussed in video prediction research [16].

To analyze the issue here, it is necessary to closely evaluate the loss function used in training this model:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4}$$

The issue with this loss function is that it assumes that the underlying data distribution is Gaussian. Here, it is nearly the case that $X, Y \sim U(0,1)$, and the expected value of the uniform distribution $\mu = \frac{1}{2}(1-0) = 0.5$ will minimize the MSE loss, even if μ itself has very low probability of occurring on its own right. This feature of pixel-wise loss functions has negative side-effects for video prediction, since the mean prediction does not usually yield very realistic frames, and these types of predictions are not useful to humans.

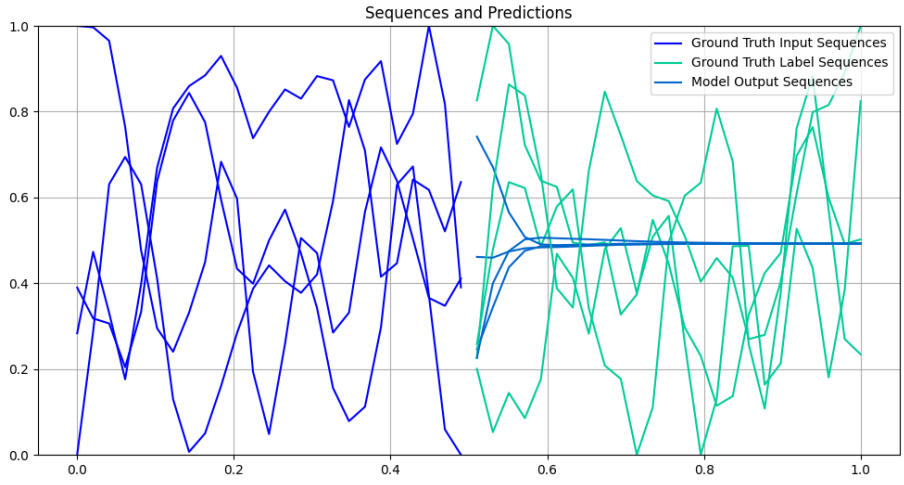

Figure 7: 4 LSTM Inferences on Test Split of GeneratedNoise Dataset

6.1.3 Stocks

Prior to performing this experiment, it was difficult to hypothesize how the LSTM model would perform on the stock price dataset, partly because the author knows very little about finance in general, but also because if it could be done reliably, then it would not be hard to make a lot of money very easily! However, from the previous experimentation with the GeneratedNoise dataset, it was reasonable that stock data points should be more cross-correlated between neighboring points, since stock prices should have some underlying predictability, even if it is minor.

In practice, the results of testing the model on this dataset showed some interesting phenomenon and differences when compared to GeneratedNoise. Firstly, the model does not just predict the mean for every sequence with this dataset, instead, it attempts to fit each line individually, as can be seen in Figure 8. In addition, the average test loss over this dataset shown in Figure 5 reveals that the prediction error does not decrease as quickly as with GeneratedNoise, and the training loss plot shown in Figure 4 shows that the overall decrease in loss over the stocks dataset was greater than for GeneratedNoise. These results show that the model does learn something from the stocks sequences, even though they are, at first glance, just as random and unpredictable as the random noise.

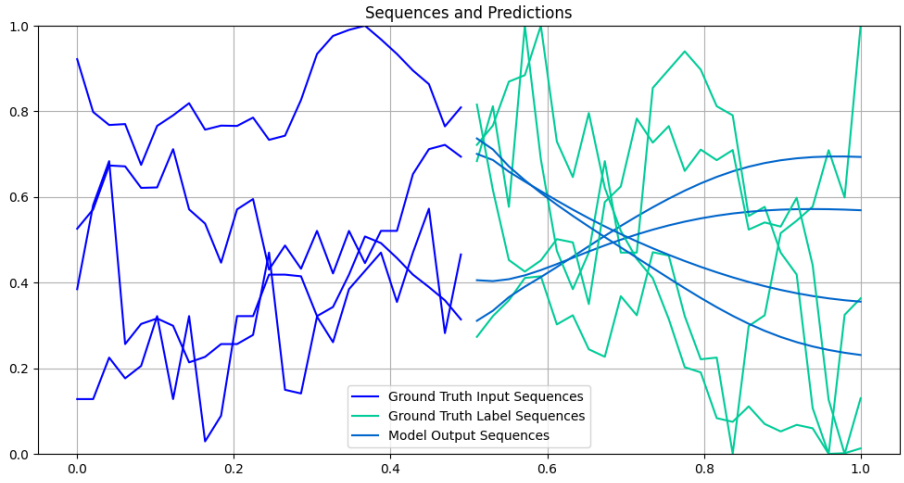

Figure 8: 4 LSTM Inferences on Test Split of Stocks Dataset

While some of the predictions from this dataset are impressive, however, and seem to almost approximate the trajectory of the stock price, a good amount of them are also mostly wrong, and even seem to diverge from the ground truth label completely. From a qualitative perspective, these predictions do not seem accurate enough to make any serious decisions off of, however, it would be very interesting to test the performance of this model in making trading decisions as a future experiment.

6.2 Video Prediction

Now on to the focus of this project, which is to use a variation of the previously-used model equipped with convolution in order to predict the continuation of video sequences.

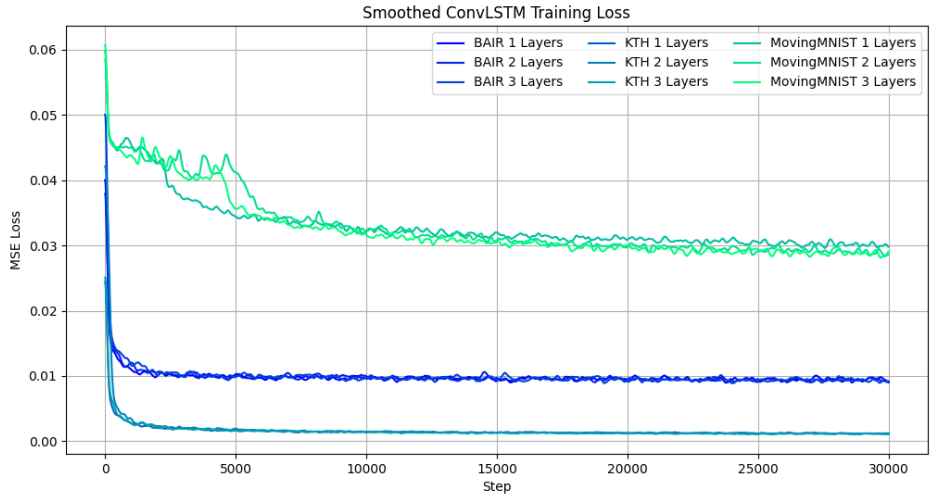

Figure 9: ConvLSTM Training Loss on MovingMNIST, KTH, and BAIR Datasets for Models with Varying Number of Cell Layers

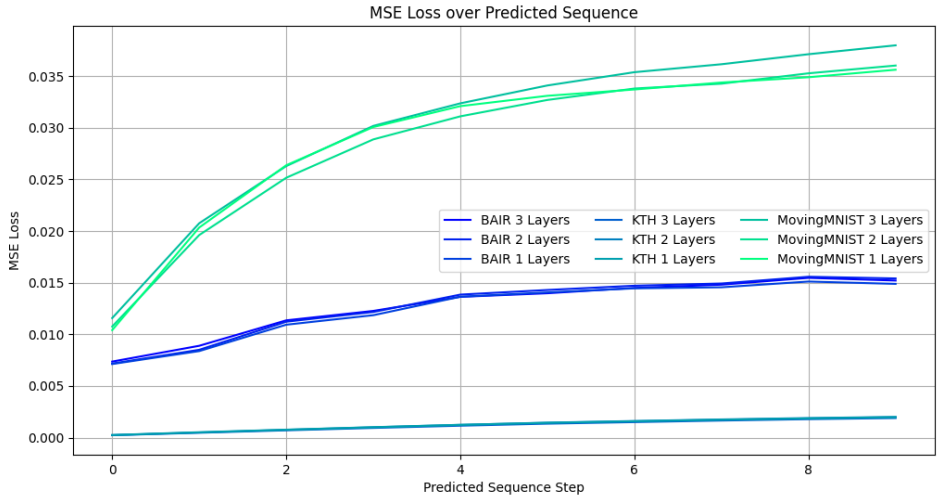

Figure 10: ConvLSTM Test Loss over Predicted Sequence Steps on MovingMNIST, KTH, and BAIR Datasets for Models with Varying Number of Cell Layers

As before, this model's training loss over the three video datasets MovingMNIST, KTH, and BAIR are shown in Figure 9, and the average test loss plotted over the prediction sequence steps for each dataset and number of cell layers is shown in Figure 10. Additionally, the average test loss is shown in Table 2.

Table 2: ConvLSTM Average Test Losses For Each Dataset and Number of Layers

Layers	**MovingMNIST**	**KTH**	**BAIR**
1	1.249×10^{-2}	1.233×10^{-3}	2.910×10^{-2}
2	1.278×10^{-2}	1.239×10^{-3}	2.876×10^{-2}
3	1.274×10^{-2}	1.163×10^{-3}	3.019×10^{-2}

Additional test inferences for each dataset are shown in the Appendix Section 9.2.

6.2.1 Moving MNIST

The model's inferences on the MovingMNIST were annoyingly mixed, with some results being quite clear, such as the one shown in 11, and others being much more muddled and blurry, such as the one shown in 12. Since the MovingMNIST dataset has elements of predictability (for example, digits bounce off the walls in a predictable way and move at a constant velocity) the expectation was that this dataset would be easy for the ConvLSTM model to learn, however, the training loss plot in Figure 9 and the average test sequence loss in Figure 10 show that this dataset was the most difficult of the three.

As to why this is the case, it could be a combination of several factors: poor bias learning, occlusion, or model depth and training time limitations.

For the first reason, it is important to look at the ways in which MovingMNIST is unique to the other datasets, for example, it is extremely bimodal, and pixel intensity values densely cluster around 1 and 0, or white and black, as they appear in images. Although the LSTM architecture (Figure 2) includes a bias term in its weights, the bias needed here is not a simple pixelwise correction, it is a correction term needed for each frame as the digits bounce around. While a bias term such as the one included in the vanilla LSTM cell implementation is capable of correcting the image predictions to be mostly black, for example, it is not capable of learning that the pixels in each digit as they bounce around should be white, since the bias terms are constant over the entire prediction. Thus, the model struggles to produce such black-and-white predictions.

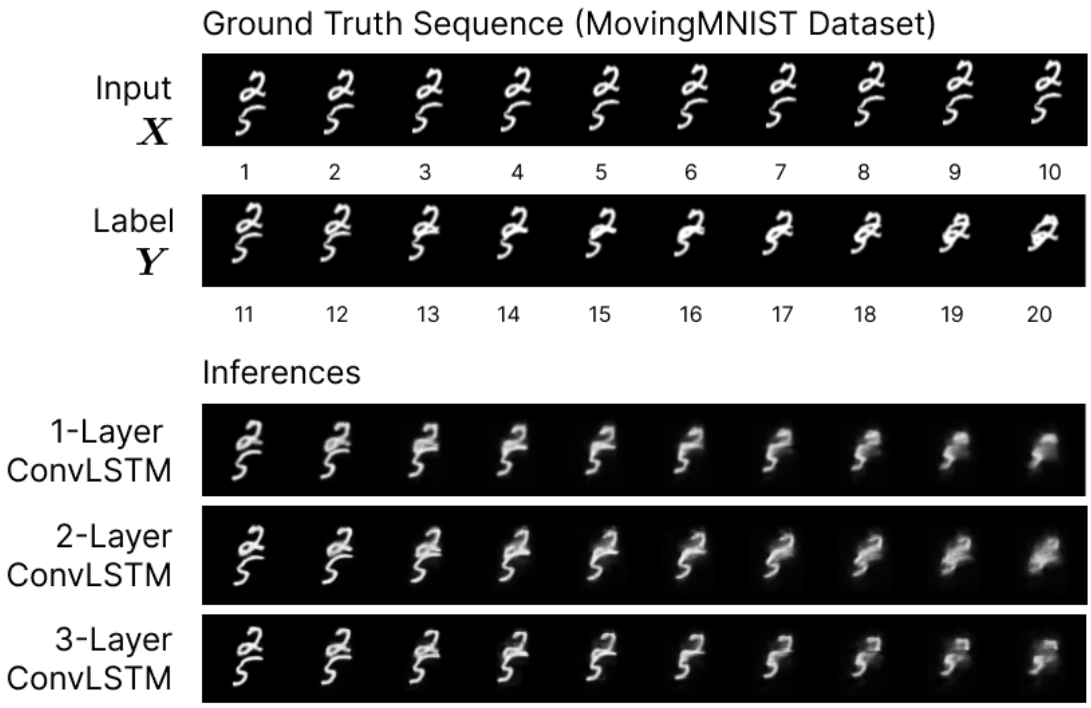

Figure 11: Convolutional LSTM Inference on MovingMNIST dataset

While the second reason doesn't account for the high training loss directly, since occlusion is only present in about half of the sequences and even then isn't a major factor in most cases, it does increase the underlying complexity of the dataset to some degree. This, combined with the third reason, could also be major contributors to the poor prediction accuracy in this case, since the model was only able to train over 10 epochs due to time and computing power constraints. Additionally, the dataset is generated on the fly from the MNIST digits dataset, so it would be easily possible to train a deeper model along with many more generated sequences in future work.

The dataset creators were able to achieve a much better accuracy using a different model architecture, an LSTM Autoencoder [21].

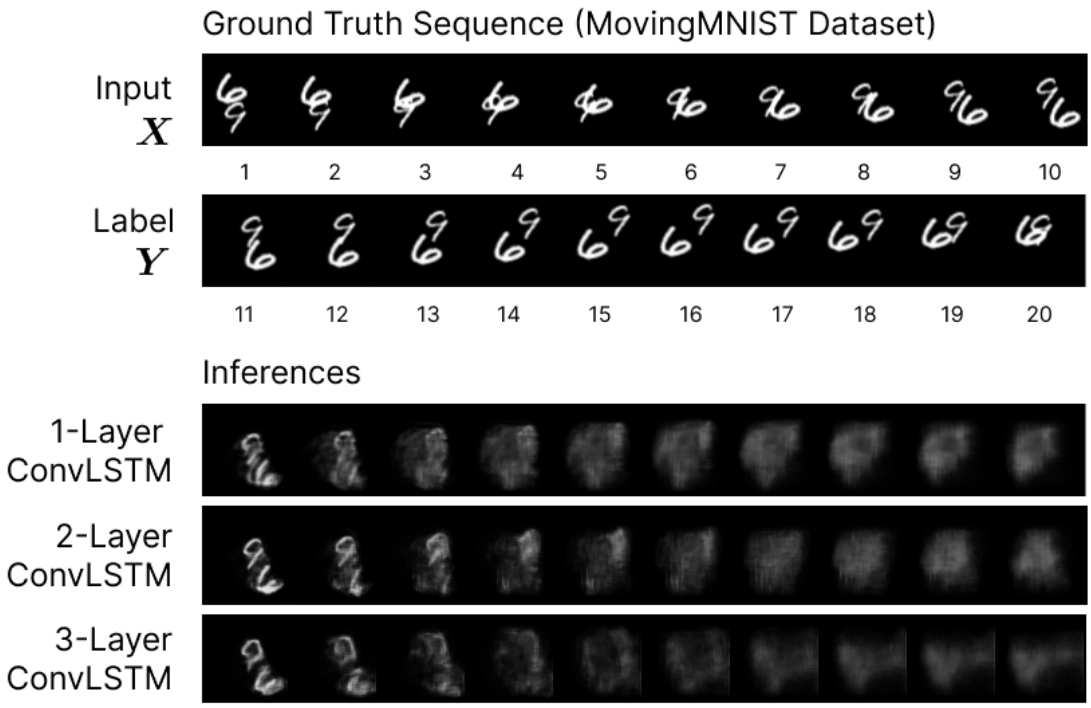

Figure 12: Convolutional LSTM Inference on MovingMNIST dataset

6.2.2 KTH

The ConvLSTM model trained and tested best on the KTH dataset out of the three, as the loss plots in Figures 9 and 10 confirm, and this success led to some of the most impressive and satisfying predictions in this project. Additionally, the models trained on the KTH dataset achieved an average test MSE loss of about 10 times less than the other two.

The inferences shown in Figure 13 show a collection of test predictions done by the model as it trains, and as the model progresses, it seems to be able to reduce blurring as well as increase prediction accuracy persistence. Although the train loss does not decrease very dramatically after the initial decrease in Figure 9, the predictions become much more clear, showing that the difference in MSE loss between a blurry image and a clear image that both roughly approximate the correct prediction is very small.

From a qualitative perspective, this inference is very exciting as it seems to correctly predict the footsteps of the walking man even during the difficult stage of occlusion when the legs are superimposed, and for the most part, other inferences taken from the KTH dataset were similarly detailed. Figure 14 shows another inference of similar quality on a video of a man waving his arms.

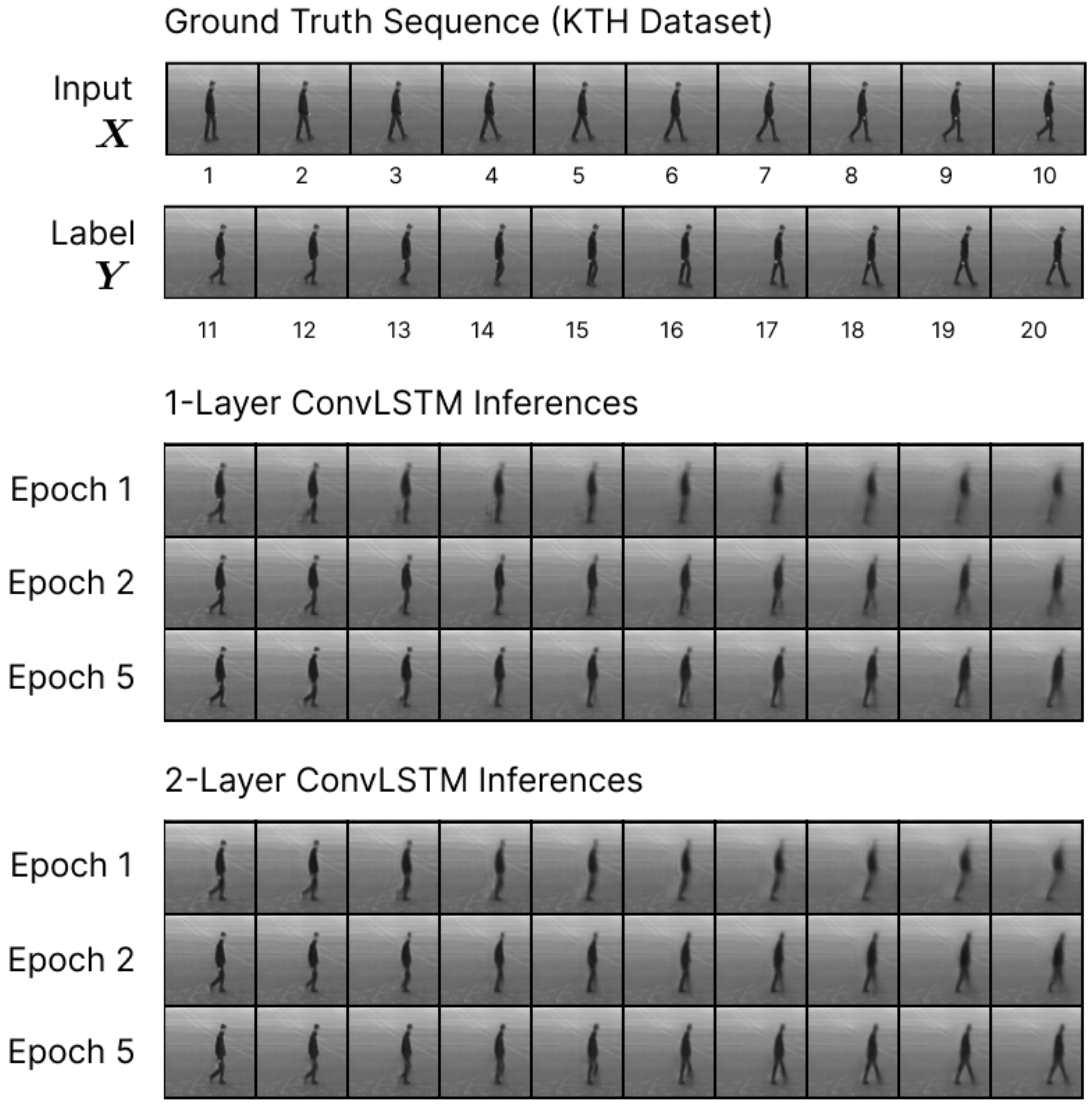

Figure 13: Convolutional LSTM Inference on KTH dataset (Sample 10)

This inference (as well as the inferences from the MovingMNIST dataset in Figures 11 and 12) also seek to compare the effect of the number of LSTM cell layers used in the model on the prediction quality, and this effect seems to be made clear in this example. Compared to the model with only one layer, the models with 2 and 3 layers are generally able to produce clearer, more accurate results for this dataset. This detail is also confirmed by the table of average test losses in Figure 2, which shows that the 3-layer model had a slightly lower average loss on the KTH dataset.

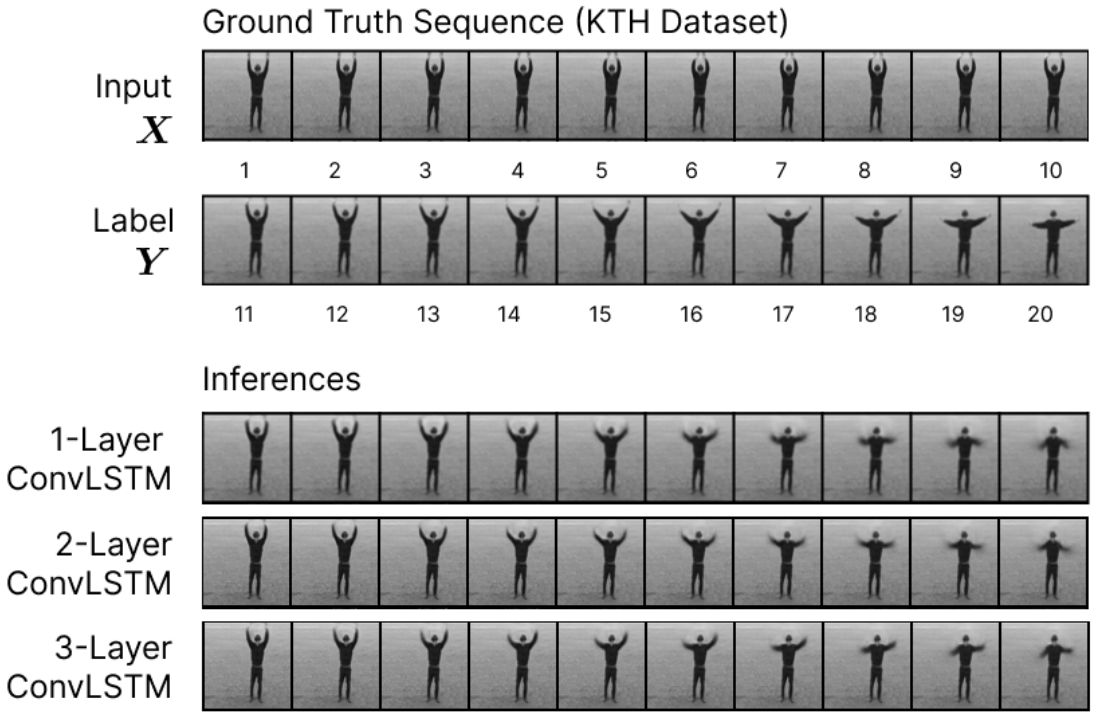

Figure 14: Convolutional LSTM Inference on KTH dataset (Sample 50)

Although the KTH dataset was the only video prediction dataset in this project which showed a beneficial effect of adding cell layers, several papers concluded that more layers should have a positive effect [21, 22], and common sense dictates that a model with more parameters should, in general, be capable of learning a higher-level representation of training data. Some reasons for why the results in this paper do not match up with those expectations, therefore, could be that the models trained were altogether much too small, meaning that there would be little difference between models with 1, 2, or 3 layers, or that the data used except for KTH is too complex for such a simple model to predict well at all, which is not an unlikely conclusion. The most successful predictions seems to come from adversarial models which incorporate a discrimator in conjunction with a generator (The Seq2Seq model, for example is a generator) and therefore avoid blurry predictions as well as some of the other shortcomings of this ConvLSTM setup (see Section 7).

In short, the reason why adding more layers to the model doesn't simply increase the accuracy of predictions in this case is most likely because there are bigger problems at play having to do with fundamental shortcomings of vanilla LSTMs or severe implementation defects. Therefore section 7 will discuss some popular alternatives and improvements that could be made to this project's implementation.

6.2.3 BAIR

For engineers, one of the most interesting and immediate applications of video prediction is to better control robotic movement. While most of this report discusses video prediction purely from a computer science perspective, it may actually turn out to be necessary to incorporate research from other domains such as robotics in order to make meaningful improvements to current computer science methodologies—after all, as discussed in the introduction, humans mostly have their own awareness of what will happen in the near future as a result of observation *and* interaction [3]. This motivation may be partly what is behind a growing interconnection between robotics research and computer vision, as evidenced by growing research in the field as well as the existence of this dataset and its big brother, RoboNet [4], another dataset from Berkeley Artificial Intelligence Research (BAIR) which contains even more robotic movement videos.

The ConvLSTM model's performance on the BAIR robot pushing dataset used in this project, however, is mixed. The models trained on this dataset achieved a lower training loss and test sequence loss than on the MovingMNIST dataset (Figures 9, 10), but higher than the KTH dataset. This could be because the content of the BAIR dataset is more random than the KTH dataset and the backgrounds are far less uniform, however it is less directly bimodal, as discussed in Section 6.2.1 about the MovingMNIST sequences.

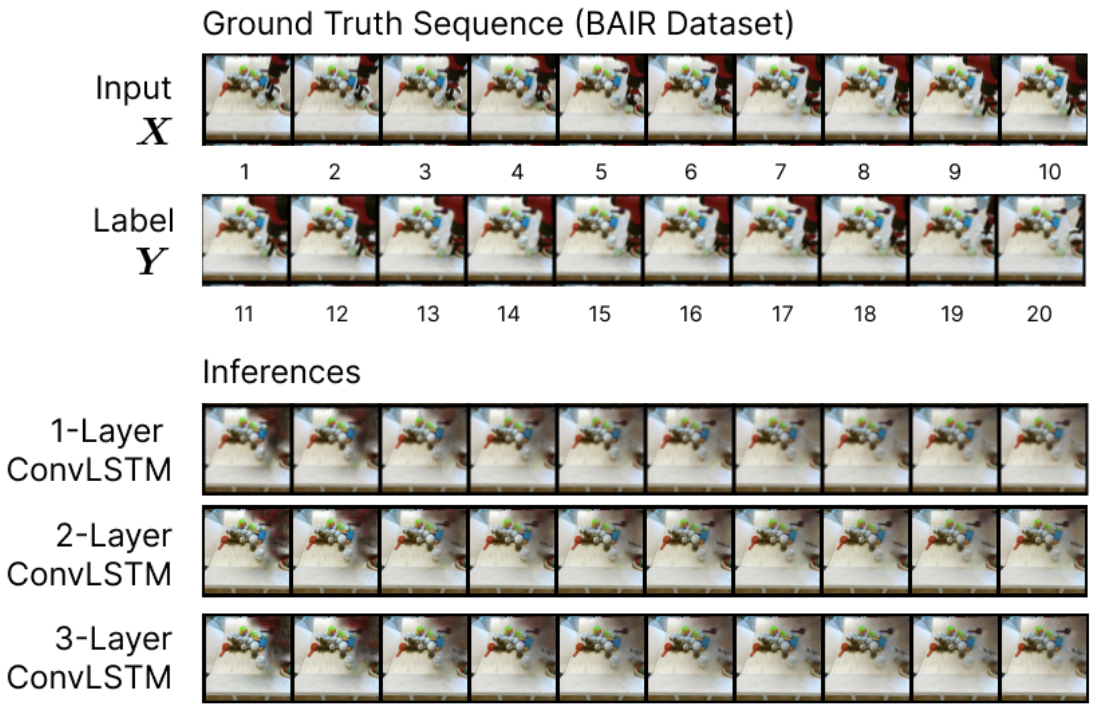

Figure 15: Convolutional LSTM Inference on BAIR dataset

The inference of the ConvLSTM model shown in Figure 15 seems to correctly predict the general location of the robot arm as it moves across the frame, however it is so blurry that it is hard to tell exactly what the model is fundamentally predicting. While most of this blurriness likely comes from pixel-wise loss effects (see Section 6.1.2), some of it could also be due to the fact that the robot arm moves very quickly across the frame, and this can be seen in the ground truth label for this prediction, where the arm disappears from the frame very suddenly between images 18 and 19. In general, the robot arm in this dataset does not seem to move at a constant velocity, and it makes many changes to its trajectory that are hard for a human to predict.

An additional inference is shown in Figure 16, in which it is a little easier to see how the model predicts the general motion of the arm as it moves to the left edge of the frame. While the BAIR dataset contains around 40,000 sequences, it also has much more complexity than the other datasets, since the robot arm is constantly pushing numerous objects around the frame, and there are countless possible configurations of the arm. This reason could also help to explain why sequences appear so blurry.

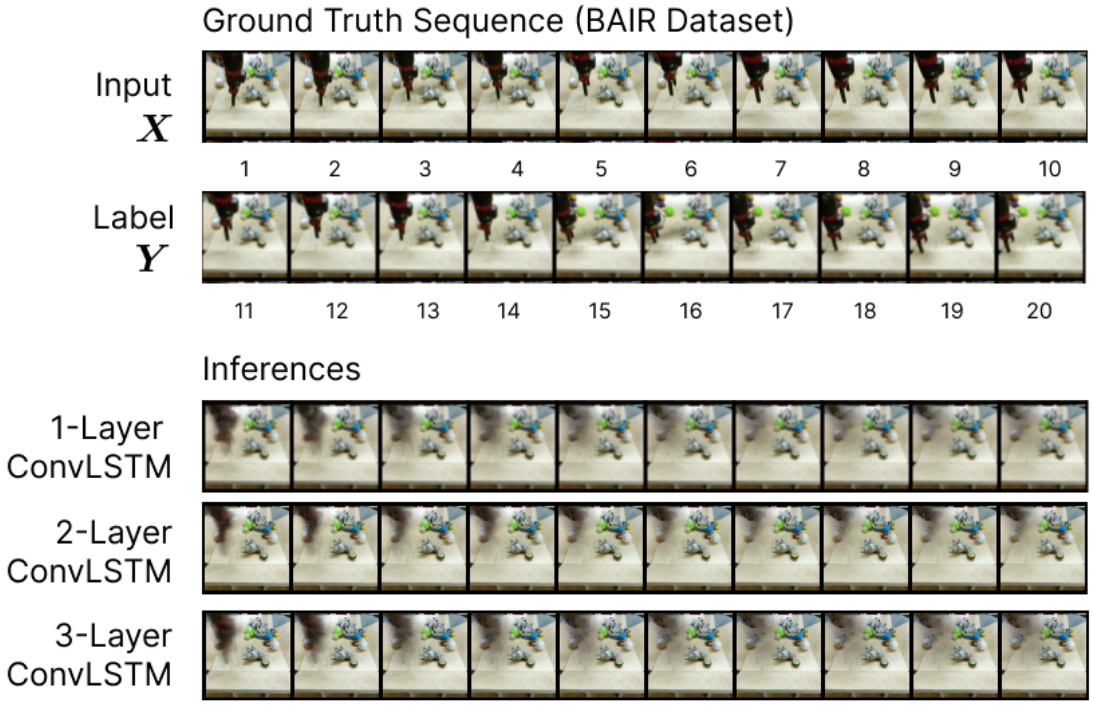

Figure 16: Convolutional LSTM Inference on BAIR dataset

7 Future Work

In video prediction research, there are numerous different approaches to video prediction in addition to the ConvLSTM implementation used here, and some of them are able to drastically improve on some of the shortcomings encountered in this project. Batch Normalization, for example, is a well-researched addition to many computer vision models which can potentially increase LSTM performance, however, is very difficult to implement with RNNs [13]. Peephole connections are an easier improvement to implement, and are described along with the original presentation of LSTMs [11]. One of the most popular and successful alternatives, however, is a Generative Adversarial Network (GAN), which consists of a generator and a discriminator. A diagram of the general architecture is shown in Figure 17.

A generator used in a GAN consists of mostly the same architecture as a discriminative neural net, such as the one which might be used for image classification. While discriminative nets seek to learn the conditional probability $p(y \mid x)$ of an input x belonging to a particular class y; however, generative nets seek to learn the conditional probability distribution $p(x \mid y)$ of an input data given the output label, allowing them to make inferences in the form of "imagined" data that might belong to the same distribution as x [10]. In short, discriminative models would look at many Van Gogh paintings and fakes in order to learn to differentiate between them, and generative models would look at many Van Gogh paintings in order to learn how to paint like Van Gogh.

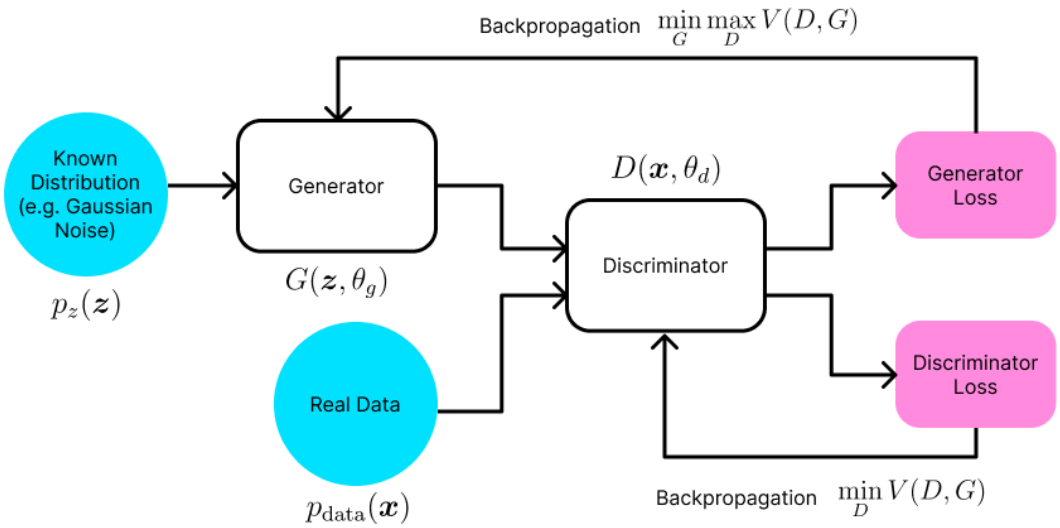

Figure 17: FutureGAN Architecture

In practice, GANs train a generator $G(\boldsymbol{z}, \theta_g)$ which seeks to learn a mapping from an input noise distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ to the targeted data distribution p_g over data $\boldsymbol{x}$, and a discriminator $D(\boldsymbol{x}, \theta_d)$, which learns to discriminate between real training samples and

generated samples from G. The value function $V(G, D)$, describes the minimax game played between G and D:

$$\min_G \max_D V(G, D) = E_{\boldsymbol{x} \sim p_{data}(x)}[\log(D(x))] + E_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[log(1 - D(G(x)))] \tag{5}$$

The result is a model which is capable of not only generating very good predictions to minimize pixel-wise loss functions, but also of generating more predictions which are more likely to occur on their own right. Both FutureGAN [2] and SAVP [14] as well as many other video prediction models which are much more successful than the ConvLSTM model used in this project use GANs, and so the most clear path for future work in this field is to implement and test a GAN model and compare its results with the ConvLSTM.

For additional enrichment on the exciting topic of GANs, and as a window into how the previously described architecture is implemented in practice, a figure taken from the FutureGAN implementation is shown in Figure 18.

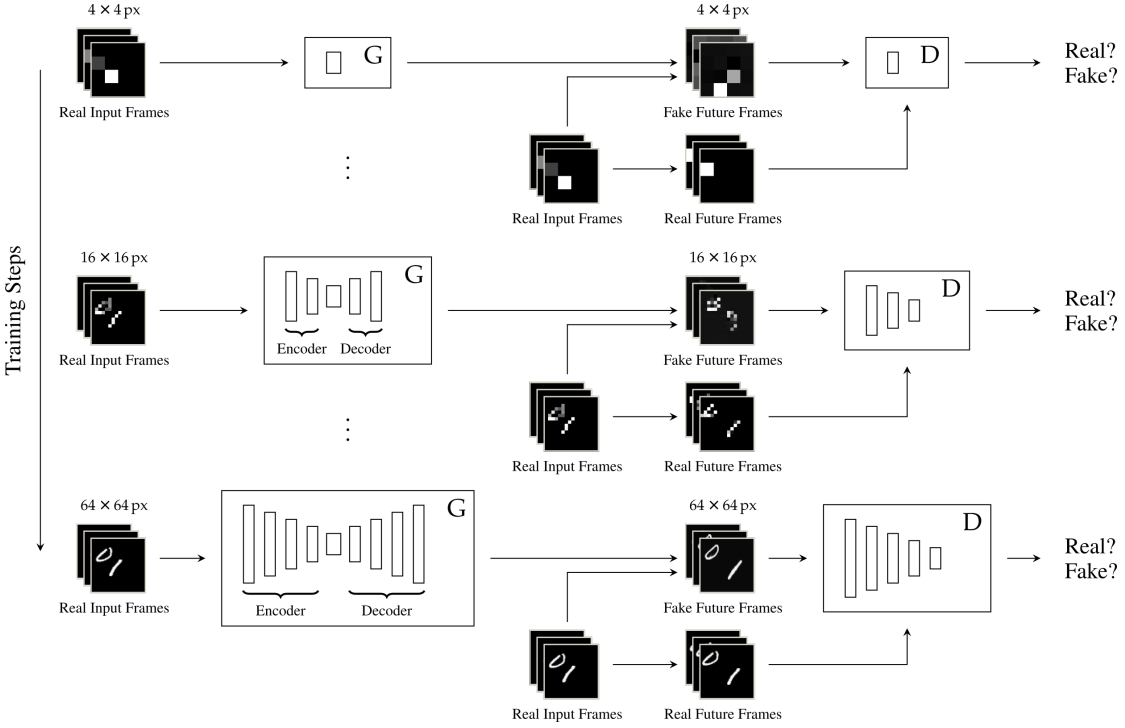

Figure 18: General GAN Architecture

Note the G and D symbols denoting the model generator and discriminator, and interestingly, how this model grows in prediction resolution and model parameters as it trains. FutureGAN was originally singled out as a potential avenue for future work in this project, but it was discontinued due to time constraints.

8 Conclusion

In conclusion, this project represents the successful implementation and testing of a Seq2Seq LSTM model on several datasets for sequence and video prediction. In addition, models with varying numbers of cell layers were tested to determine whether more layers would result in greater prediction accuracy.

In general, the model equipped with linear layers performed extremely well on the dataset consisting of generated sin waves and was able to predict them with very high fidelity, although it struggled greatly on the dataset consisting of generated noise. In addition, it achieved a slight improvement (compared to the generated noise) on predicting the stock price dataset. The hypothesis that more cell layers would positively impact model performance held most clearly on these datasets, as Table 1 shows that the models with 3 layers outperformed the models with 1 layer for both the GeneratedNoise and GeneratedSins datasets. The quantitative results of these experiments are discussed in Section 6.1.

Furthermore, the model equipped with convolutional layers produced mixed results on the BAIR and MovingMNIST datasets, but performed exceptionally well on the KTH dataset. One highlight of the experimental results was that each of the ConvLSTM models with 1, 2, and 3 layers were seemingly capable of predicting a man's footsteps as he walks through the frame in a sequence belonging to the KTH dataset. Another promising result from the KTH dataset was that it was the only dataset of the three for video prediction which confirmed that more cell layers would equate a greater model performance.

While the results for different numbers of cell layers showed mixed results for this hypothesis all considered, many of the model's failures as they are described in each section may contribute to the inconclusiveness of this testing, such that a dataset was too complex or the model was already too disadvantaged in predicting samples from a dataset that more layers would not help in every case. The two datasets which were most successful, however, GeneratedSins and KTH, both showed improvements with more layers. Since multiple sources [21, 22] observed that deeper LSTM networks generally perform better, more exploration is required to examine the results of this model.

References

[1] Alpha vantage, 4 2022. URL https://www.alphavantage.co/.

[2] S. Aigner and M. Körner. Futuregan: Anticipating the future frames of video sequences using spatio-temporal 3d convolutions in progressively growing gans, 2018. URL https://arxiv.org/abs/1810.01325.

[3] A. Cleeremans and J. Mcclelland. Learning the structure of event sequences. *Journal of experimental psychology. General*, 120:235–53, 10 1991. doi: 10.1037//0096-3445.120.3.235.

[4] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning, 2019. URL https://arxiv.org/abs/1910.11215.

[5] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[6] E. Denton and R. Fergus. Stochastic video generation with a learned prior. *CoRR*, abs/1802.07687, 2018. URL http://arxiv.org/abs/1802.07687.

[7] F. Ebert, C. Finn, A. X. Lee, and S. Levine. Self-supervised visual planning with temporal skip connections, 2017. URL https://arxiv.org/abs/1710.05268.

[8] W. Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning Cited by*, 3, 2019.

[9] C. Finn and S. Levine. Deep visual foresight for planning robot motion, 2016. URL https://arxiv.org/abs/1610.00696.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[12] A. Hu, F. Cotter, N. Mohan, C. Gurau, and A. Kendall. Probabilistic future prediction for video scene understanding, 2020. URL https://arxiv.org/abs/2003.06409.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

[14] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine. Stochastic adversarial video prediction. *CoRR*, abs/1804.01523, 2018. URL http://arxiv.org/abs/1804.01523.

[15] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow, 2017. URL https://arxiv.org/abs/1702.02463.

[16] S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros. A review on deep learning techniques for video prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2020. ISSN 1939-3539. doi: 10.1109/tpami.2020.3045007. URL http://dx.doi.org/10.1109/TPAMI.2020.3045007.

[17] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2012. URL https://arxiv.org/abs/1211.5063.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[19] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local svm approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 32–36 Vol.3, 2004. doi: 10.1109/ICPR.2004.1334462.

[20] A. Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL http://arxiv.org/abs/1808.03314.

[21] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms, 2015. URL https://arxiv.org/abs/1502.04681.

[22] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.

[23] Y. Yu, X. Si, C. Hu, and J. Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, 07 2019. ISSN 0899-7667. doi: 10.1162/neco_a_01199. URL https://doi.org/10.1162/neco_a_01199.

9 Appendix

9.1 Additional Sequence Prediction Samples

9.1.1 GeneratedSins

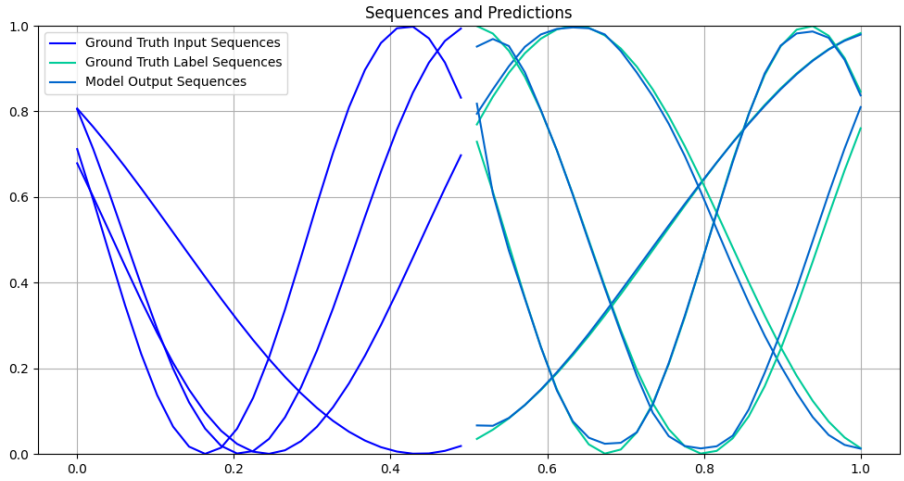

Figure 19: 4 1-Layer LSTM Inferences on Test Split of GeneratedSins Dataset

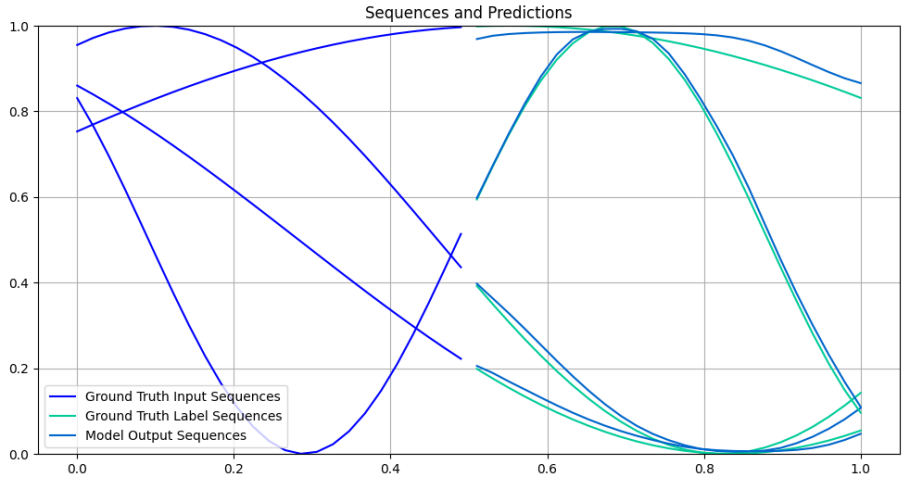

Figure 20: 4 2-Layer LSTM Inferences on Test Split of GeneratedSins Dataset

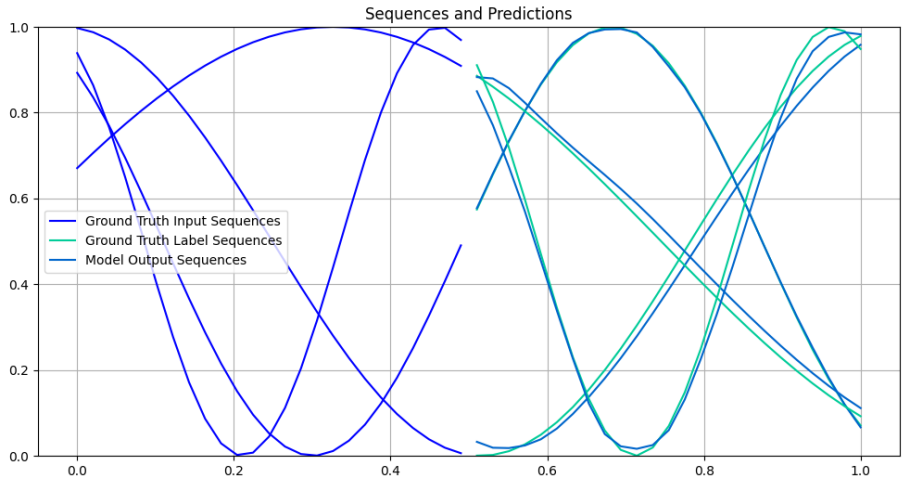

Figure 21: 4 3-Layer LSTM Inferences on Test Split of GeneratedSins Dataset

9.1.2 GeneratedNoise

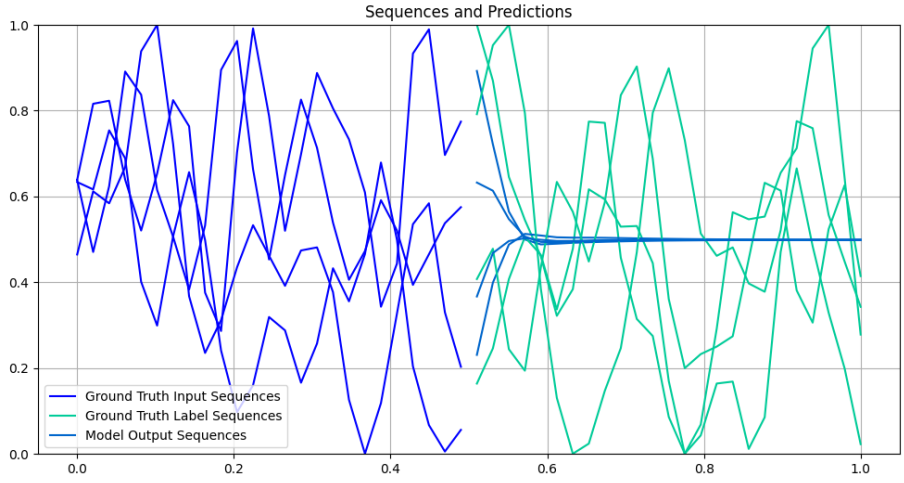

Figure 22: 4 1-Layer LSTM Inferences on Test Split of GeneratedNoise Dataset

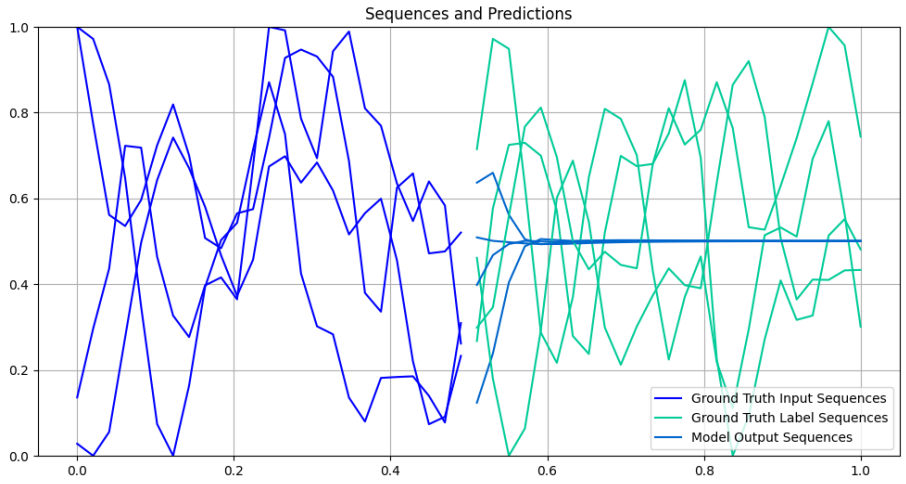

Figure 23: 4 2-Layer LSTM Inferences on Test Split of GeneratedNoise Dataset

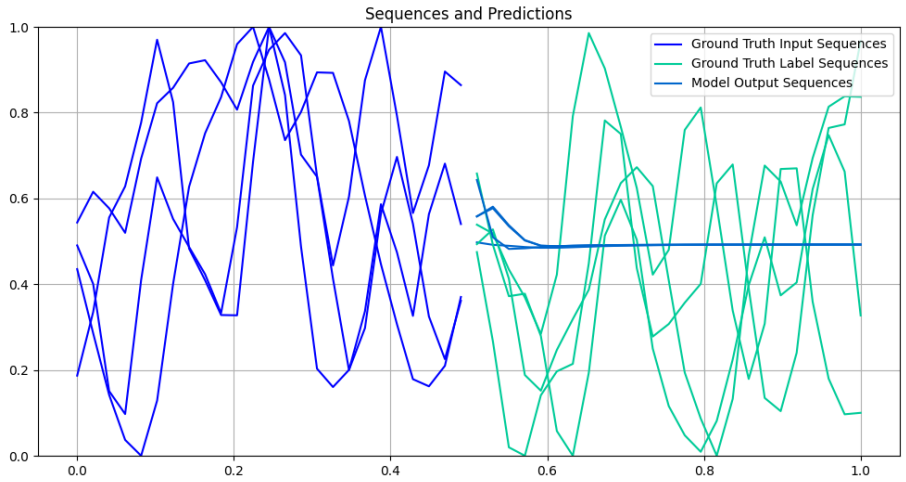

Figure 24: 4 3-Layer LSTM Inferences on Test Split of GeneratedNoise Dataset

9.1.3 Stocks

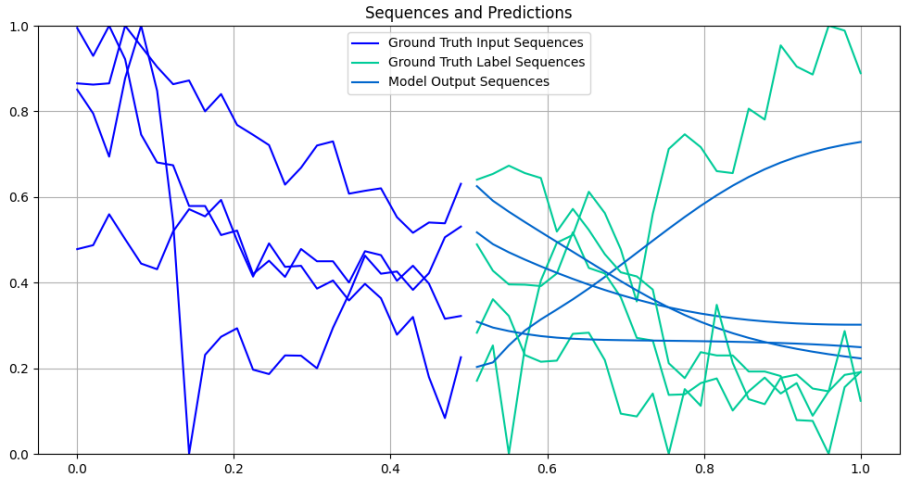

Figure 25: 4 1-Layer LSTM Inferences on Test Split of Stocks Dataset

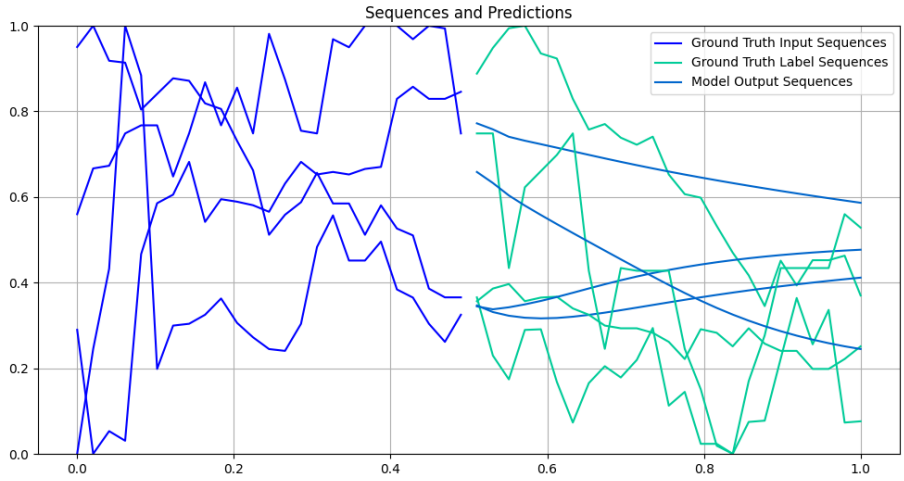

Figure 26: 4 2-Layer LSTM Inferences on Test Split of Stocks Dataset

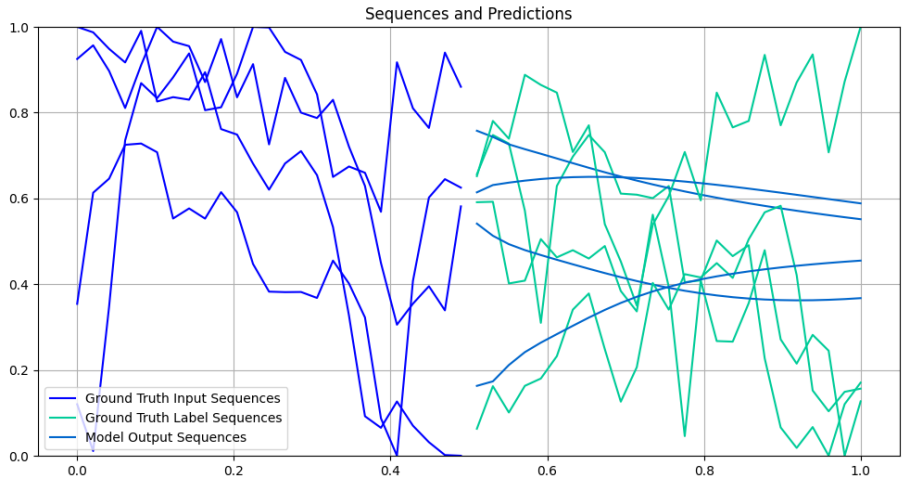

Figure 27: 4 3-Layer LSTM Inferences on Test Split of Stocks Dataset

9.2 Additional Video Prediction Samples

For each sample shown below, the first row of images is the ground truth input and label, the second row is the model's prediction, and the third row is the absolute difference between the label and prediction.

9.2.1 MovingMNIST

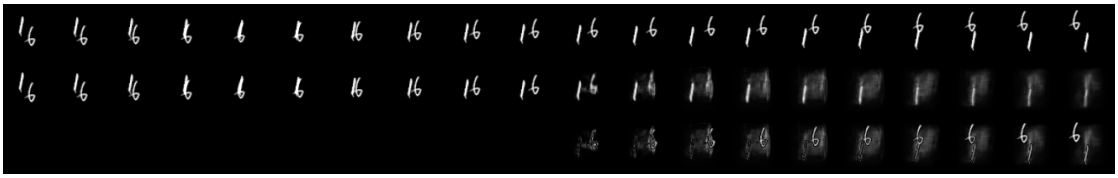

Figure 28: 1-Layer ConvLSTM Inference on Test Split of MovingMNIST Dataset

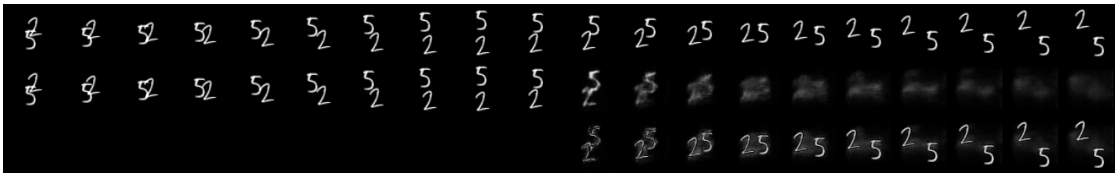

Figure 29: 2-Layer ConvLSTM Inference on Test Split of MovingMNIST Dataset

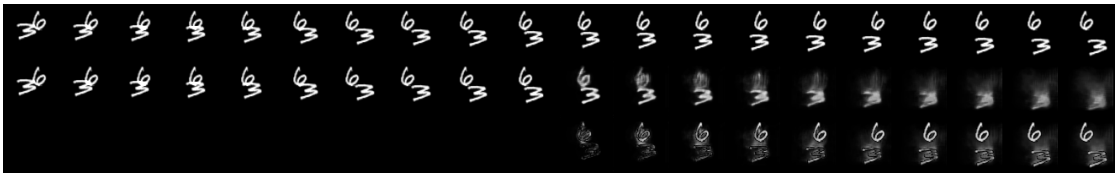

Figure 30: 3-Layer ConvLSTM Inference on Test Split of MovingMNIST Dataset

9.2.2 KTH

Figure 31: 1-Layer ConvLSTM Inference on Test Split of KTH Dataset

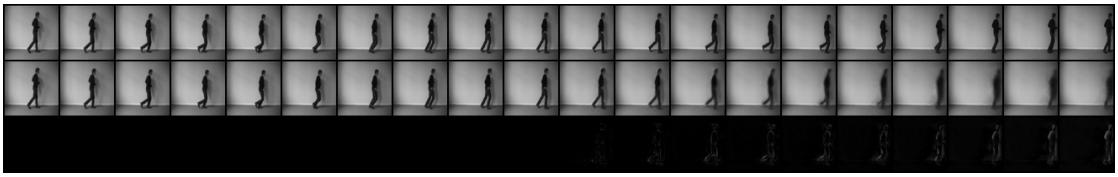

Figure 32: 2-Layer ConvLSTM Inference on Test Split of KTH Dataset

Figure 33: 3-Layer ConvLSTM Inference on Test Split of KTH Dataset

9.2.3 BAIR

Figure 34: 1-Layer ConvLSTM Inference on Test Split of BAIR Dataset

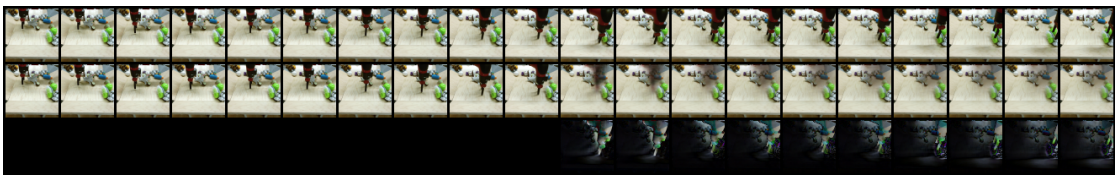

Figure 35: 2-Layer ConvLSTM Inference on Test Split of BAIR Dataset

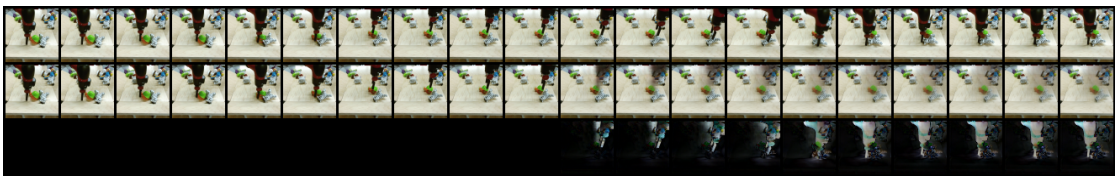

Figure 36: 3-Layer ConvLSTM Inference on Test Split of BAIR Dataset