# A Final Report on DeepNewton: Physical Simulations Using Deep Networks

Burak Çatalbaş and Yarkın Deniz Çetin
Bilkent University, Dep. of Electrical and Electronics Eng., Dep. of Computer Engineering

***Abstract*—The object interactions are generally computed using physical equations governing them. The current physical simulators and engines use the simple Newtonian equations to calculate objects future behavior from their current states, including its exact values of velocity, mass and friction values. For humans, similar equation solving systems does not exist explicitly, however we still observe the surrounding environment and create meaningful predictions on future states of the objects within. By using a convolutional recurrent network, we propose a model capable of similar predictions using the visual intuition. Experiments show that our model learns naïve physics concepts such as continuity of objects, their solidity, their speed and momentum and generates future physical states, and transferring information for multiple objects to single objects is possible.**

## I. Introduction

We ask the question of how physical computation of our surroundings is efficient and why we don't take exponentially more time to calculate our actions with an exponentially more cluttered scene. The interactions between the environment and us are calculated. For this reason, we provide a possible answer by describing and a training a physical model which can effectively generate future images with different number of objects with same amount of memory and computation.

In this project tried to expand the works of Fragkiadaki et al., by going to three dimensions instead of two. In this work, the physical reasoning tasks will be evaluated in space; to have a difference from work done before. In this way we expand the input space specifically, so that the original network structure is cascaded with other networks which take other input pictures with a different z coordinate. After connecting those layers with each other's, we will form our full structure.

In our final project, there are two main objectives we focused on. The first objective was training and testing a future image predictor which could generate the future images given the previous frames i.e. learning the physical interactions. Our second objective is to extend this understanding of physics into the multiple objects. One of the interesting part of this physics based model is that it is does not have an algorithmic complexity bounded by the amount of physical bodies. The multi-object model demonstrates both ability to predict in multi-object and single object environments.

## II. Related Work

To mention related works on this topic, an important work on this field is "A Compositional Object-Based Approach to Learning Physical Dynamics" by M. B. Chang and his colleagues [1]. In this article, an approach which generalizes over varying object count and different scenarios is followed. Their neural physics engine mainly uses object-based representations and function compositions, and performs prediction, generalization and inference tasks in experiments. They also made the network able to predict the masses of objects based on simple elastic collision rules with a high accuracy (around 90%) alongside those tasks. Existence of walls and different scene configurations are also taken in consideration for those experiments for 2-D images as the raw input data [2].

The raw 2-D image part draws more inspiration from the [3] where Fragkiadaki et al, demonstrates neural networks ability to predict the motions of 2D images.

In the work of Battaglia et al., "Interaction Networks for Learning about Objects, Relations and Physics" we see a Relation Discovery-based approach for forming predictors to find out object movements in various scenarios. Here, explicit reasoning about objects and relations are aimed and an unexpected success on physical prediction is acquired as it can be seen from MSE results. This is a hugely different approach than [4] and ours since it uses graphs as the inputs of the system rather than the raw images.

Another paper on this field is "Dynamic Modeling Based on Fuzzy Neural Network for a Billiard Robot" by Gao et al. which directly aims on billiard ball motion prediction. By forming the 'Billiard Robot', they aim to decrease the angular error rate as much as possible, and for this purpose they employ Back Propagation Neural Network (BPNN) method to create a fuzzy neural network which predict the position of billiard balls [5]. They used Monte-Carlo method for training with samples of billiard scenes and managed to reach successful results, as the predicted ball fell close enough to actual position of the billiard ball. In summary, this is a specific paper on billiard ball prediction on 2D scenes which extensively requires physical relations' learning by neural structure and prediction of their movements to make successful simulations by the neural networks.

The last example of the related works is the "NeuroAnimator" from Grzeszczuk et al. Which is a fast neural network emulation and control of physics-based models as they present. In this paper, computation expensiveness is target by usage of neural structures while computing the physical relations [9]. By employing an approximation by learning algorithm, thier neural network decrases computational complexity of nonlinear mappings and equations. Using hierarchical network types and aiming on the deformable models, the model achieves significant performance and practicality at the end of training for several examples.

## III. Architecture

### A. *Deep Newton v1 (Progress Report Phase)*

We propose a combined system of encoder-decoder and LSTM to predict and generate future frames according to physical laws. Our network will predict the future states of the interacting objects and generate the image representing the change in speed and direction.

This combined system is composed of 2 main components. An encoder-decoder part which handles the input frames and extracts descriptive features of the scene and an LSTM part which calculates the future states of the scene and tries to construct a physical understanding. We first learn a which maps 128x128 input frames to 128x128 output frames. The codec inherently learns two functions amd . We use the former function to generate LSTM training codes.

In Fig. 1, we give the fundamental architecture of our deep-learning model.

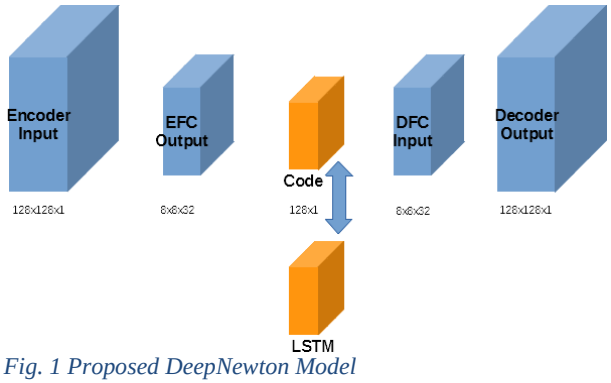



*Fig. 1 Proposed DeepNewton Model*

For our architecture, we train the encoder-decoder and LSTM parts separately, as we believe that the LSTM only needs *a representation* of the scene rather than its full contents.

### B. *Core LSTM*

LSTM networks [2] can attend to objects in a sequential manner as given in Eslami et al. [6]. In our model, we use an LSTM network as its core predictive module. The LSTM accepts the visual encodings of the system for steps and tries to create the visual code of the next step. The model infers the object interactions and passive forces (such as gravity) from the training set given.

For our current model, we use an LSTM and a fully connected layer to represent the physical state of the scene. Both LSTM and our FC layer uses 128 neurons which is our coding size described below.

### C. *Encoder-Decoder Scheme*

For tuning and cross-validating our LSTM, we believe the dense vectors representing the frames was needed. For this reason, we use a similar encoding scheme used in [3] however we do not separate the force-field vector and scene as different inputs as we expect the whole physical inference will be done through looking at the images.

The architecture consists of convolutional ReLU's followed by max-pooling. The last two layers generating the code are fully connected and scales down the 128x128 images into 128x1 code vectors. In Fig 3 you can see the implementation details

| Layer | Size | Layer | Size |
|---|---|---|---|
| **Input** | 128x128x1 | **Output** | 128x128x1 |
| **Conv1** | 3x3x4 | **Conv1** | 3x3x4 |
| **ReLU1** | 128x128x4 | **ReLu1** | 128x128x4 |
| **MaxPool1** | 64x64x4 | **Upsample1** | 64x64x4 |
| **Conv2** | 3x3x8 | **Conv2** | 3x3x8 |
| **ReLU2** | 64x64x8 | **ReLu2** | 64x64x8 |
| **MaxPool2** | 32x32x8 | **Upsample12** | 32x32x8 |
| **Conv3** | 3x3x16 | **Conv3** | 3x3x16 |
| **ReLU3** | 32x32x16 | **ReLu3** | 32x32x16 |
| **MaxPool3** | 16x16x8 | **Upsample13** | 16x16x8 |
| **Conv4** | 3x3x16 | **Conv4** | 3x3x16 |
| **ReLU4** | 16x16x16 | **ReLu4** | 16x16x16 |
| **MaxPool4** | 8x8x16 | **Upsample14** | 8x8x16 |
| **Conv5** | 3x3x32 | **Conv5** | 3x3x32 |
| **ReLU5** | 8x8x32 | **ReLu5** | 8x8x32 |
| **FC1(ReLU)** | 2048 | **FC1(ReLu)** | 2048 |
| **FC2(ReLU)** | | 128(Code) | |

*Table I. Proposed Encoder-Decoder Model, while the encoder and decoder architecture is similar, the weights are not shared*

### D. *Deep Newton v2 (Final Report Phase)*

Further experimentation revealed us that using non-spatial coding made the training problem very hard as the model had to learn the global positions and interactions separately for nearly each individual location. To solve this problem, we come up with a model which is based on Convolutional LSTMs described in [7]. The model uses a convolutional code rather than the original vector coding. This

model however, does not use LSTMs but custom designed additive RNN's for learning the time-dependencies. Our model gives a good performance and has less calculations than a full-fledge LSTM.

Another problem we have addressed is the error metrics for the single-frame prediction mode. In the initial model since we predicted the fifth frame given the first four frames, it was easy to for network to reduce the error by just imitating the last given frame. Because of temporal continuity the frames 4 and 5 were very similar and we feared this could be abused by our model. Our new model tries to address this problem by predicting the 8th frame given the first 4 frames. Using 8th frame as the training labels, we forced network to genuinely predict the future frame.

### *E. Core RNN (Final Report Phase)*

The RNN we use is inspired from the Convolutional LSTM. The RNN uses additive hidden-states as the normal LSTM networks however, the multiplication of the weights is substituted with convolutions of fixed size.

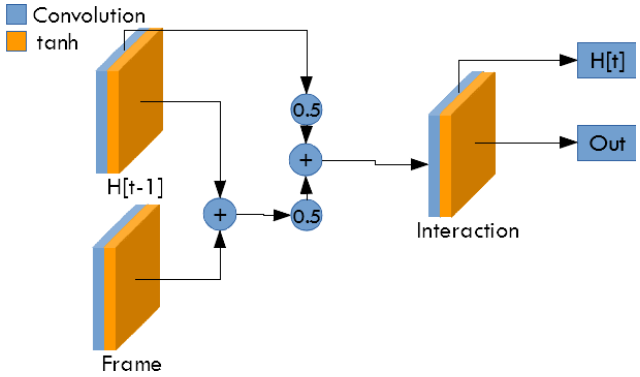



*Fig. 2 The proposed RNN cell with tanh activations and Conv. layers*

The RNN cells use a constant decay of 0.5 to act as a forget gate. In the output, we convolve the hidden state with a filter to resolve spatial dependencies of different objects. The hidden state is the "non-activated" version of the output state since the hidden state gets activated in the next cell.

The complete RNN-model uses 8 of these basic cells with minor differences in the input layer.

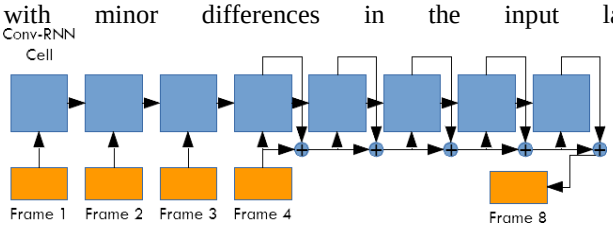



*Fig. 3 Complete RNN-model of the network. The input and outputs are frame codes*

We use the output of the previous cells as input to the next cells, but we also use a skip-connection layer to increase gradient flow to the first four items.

| Layer | Size | Layer | Size |
|---|---|---|---|
| **Input** | 128x128x1 | **Output** | 128x128x1 |
| **Conv1** | 3x3x4 | **Conv1** | 3x3x4 |
| **ReLU1** | 128x128x4 | **ReLu1** | 128x128x4 |
| **MaxPool1** | 64x64x4 | **Upsample1** | 64x64x4 |
| **Conv2** | 3x3x8 | **Conv2** | 3x3x8 |
| **ReLU2** | 64x64x8 | **ReLu2** | 64x64x8 |
| **MaxPool2** | 32x32x8 | **Upsample12** | 32x32x8 |
| **Conv3** | 3x3x16 | **Conv3** | 3x3x16 |
| **ReLU3** | 32x32x16 | **ReLu3** | 32x32x16 |
| **MaxPool3** | 16x16x8 | **Upsample13** | 16x16x8 |
| **Conv4** | 3x3x16 | **Conv4** | 3x3x16 |
| **ReLU4** | 16x16x16 | **ReLu4** | 16x16x16 |
| **8x8x16** | | **8x8x16** | |

*Table II. Final Encoder-Decoder Model, while the encoder and decoder architecture is similar, the weights are not shared*

### *F. Encoder-Decoder Scheme (Final Report Phase)*

The encoder-decoder model we used in the first version proved setbacks in physics generalization as mentioned in the introduction. For our new RNN model with convolutional layers, we introduced spatial coding for our model.

Another improvement over the first iteration is that we used a Gaussian noise in the input of the encoder, forcing decoder to generate sharp images even with noisy spatial codes. This, we believe transfers some of the learning load from RNNs making training easier. It is similar to VAE in this sense however, we omit many features from VAE such as KL divergence.

Lastly, we tried to improve the code locality by introducing a code similarity loss to our AE model. This loss forced network to create more similar codes for the consecutive frames and prevented it from code-space fragmentation.

The finalized configuration of the network layers and total neural structure can be seen in Table II.

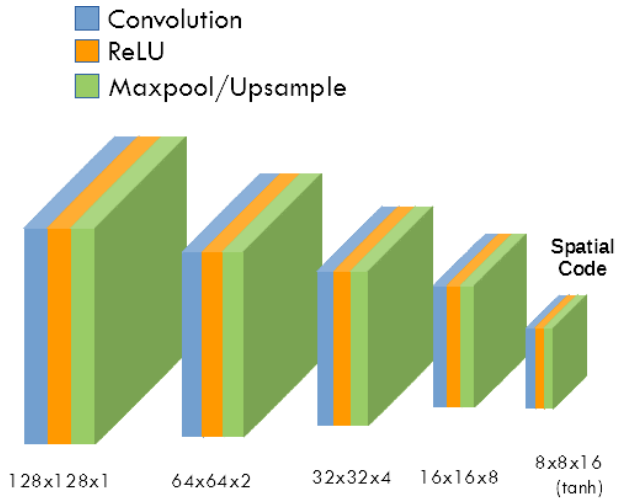



*Fig 4 Autoencoder model with Gaussian noise*

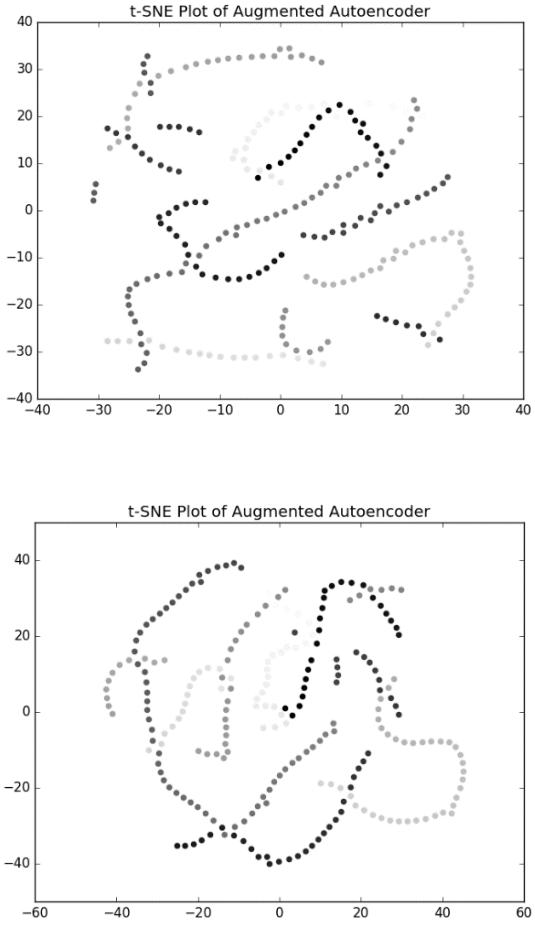



*Fig. 5 The t-SNE plots of the test-frames. The non-augmented t-SNE shows more breaks in the lines contrasted with the augmented encoder.*

Above you can see the t-SNE plots of non-augmented and augmented versions of the generated code images. It can be observed that introducing this loss created a much more continuous t-SNE "strands".

Both image-reconstruction and code-similarity errors are MSE with equal weights in our implementation.

### *G. Model Training (Final Report Phase)*

The first model is trained in two steps. The first step was the Encoder-Decoder training, which was done by using shuffled images from the dataset. The network tried to reconstruct the original image. The training was done by Adam with a learning rate of 0.0001 and 0.001. The network trained stochastically using 1 image instead of mini-batch training. We observed that using 1 image per training "batch" increased the generalization of the images when generating codes. We used 1000 epochs for the encoder-decoder.

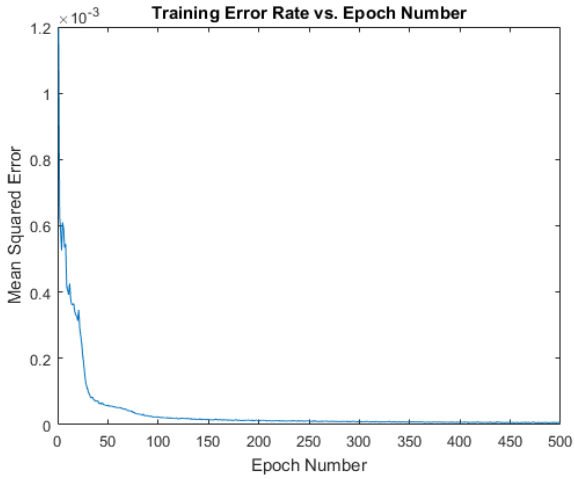


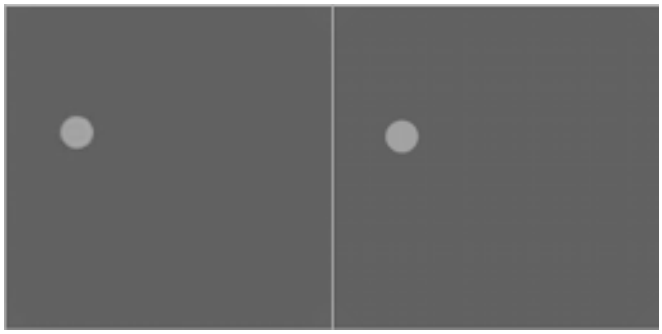

*Fig. 6 Training error over epochs for encoder-decoder*

For the LSTM part, we use the same optimizer parameters. Our loss functions for both encoder-decoder and LSTM is MSE.

During LSTM training, we observed an interesting behavior where while our training error did not decrease, additional epochs increased our test-time error visually.

For the second model, we saw improvements in prediction accuracy when trained end-to-end so we utilized a combined model of encoder-RNN-decoder. We also decided to adopt rmsProp as our learning algorithms as it seemed to improve recurrent performance as mentioned in [8]. We first trained the auto-encoder with 1000 epochs and then performed an end-to-end training of 2000 epochs.

## IV. Dataset Generation



For generating dataset, we considered two techniques, first is utilizing an existing physics engine implementation, Unity3D for our case, to create and simulate artificial environments and record the frames as our dataset. Unity3D has nice features for handling the collision detection with varying precisions. For our proof-of-concept dataset, we imagined a 2D square "pool" table, where gravity and friction is ignored. Only one ball with unit mass is considered. The walls of the scene were considered static.

The training, validation and test datasets are captured for 20, 10 and 10 seconds with 40 fps respectively and generated 800, 400 and 400 frames each. In Fig. 3 you can see 6 sample images from our training dataset.

For training the borders of the images are cropped and scaled down to 128x128 pixels only showing the background and the ball. The desired frame rate within a reasonable interval. Also, we are able to pass the pictures of 3D volume with desired elevation angle, azimuth angle and zoom factor to feel the 3D notion of simulation. This simulator can produce pictures of the plot with desired frame rate within a reasonable interval. Also, we are able to pass the pictures of 3D volume with desired elevation angle, azimuth angle and zoom factor to feel the 3D notion of simulation.

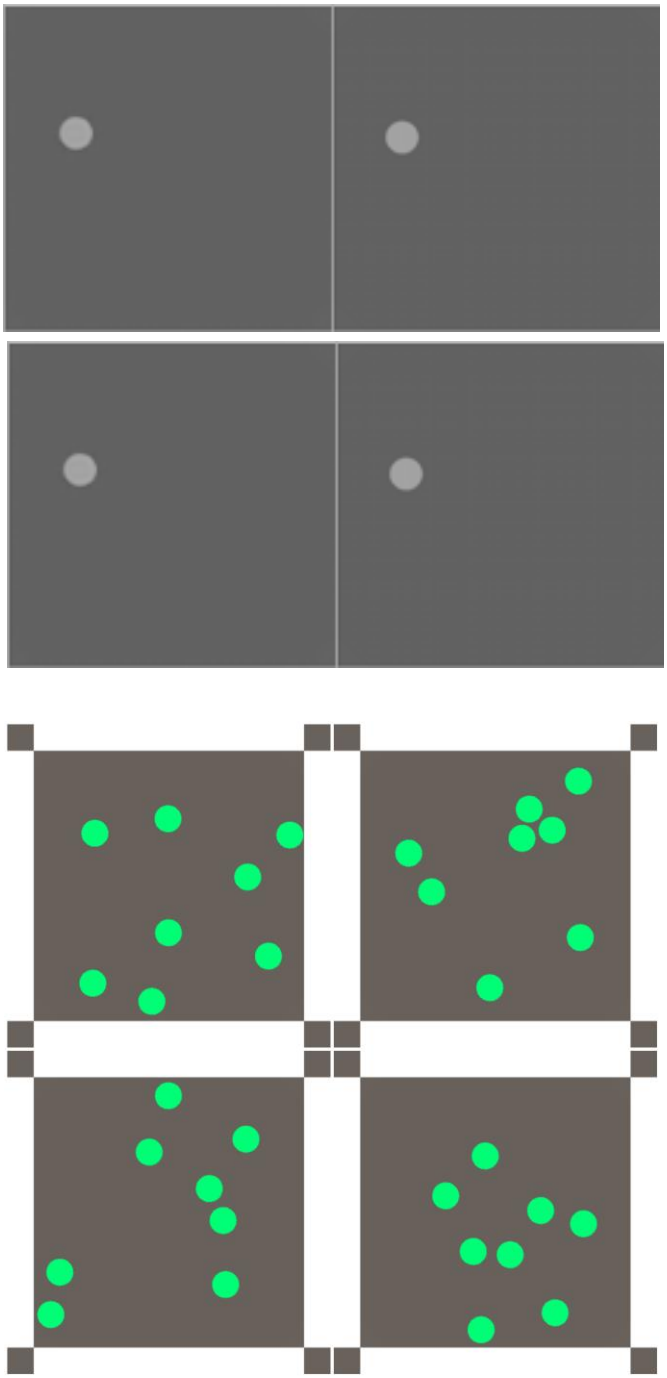


*Fig. 7a Sample Images from our model(left) and the ground truth(right)*

## V. EXPERIMENTS

### *A. Predicting 4-Frame into Future*

In the first experiment, we feed our network with 4 frames of with consecutive time-steps and predict the next frame. Then we measure the output image's MSE with the ground truth future image which is the 8$^{th}$ frame.

The MSE of this experiment is 9e-5 for this experiment. However, we note that all frames share a lot of features in the first place so we expect low MSEs for even random image pairs. To address this, we use a version of normalized MSE where we measure the base MSE, from two frames with no ball intersection and scale our error according to this value.

The MSE of non-intersecting balls in a 128x128 image is 9.6e-4. So, our scaled version of MSE is still significantly close to ground truth.

Our model predicts the 8th frame with accuracy for one object case, we also trained our model using the multiple objects dataset we have generated. Results have lost quality as you see however we still have a sense of continuity in the frames, meaning our model actually learns multiple object interaction.

*Fig*

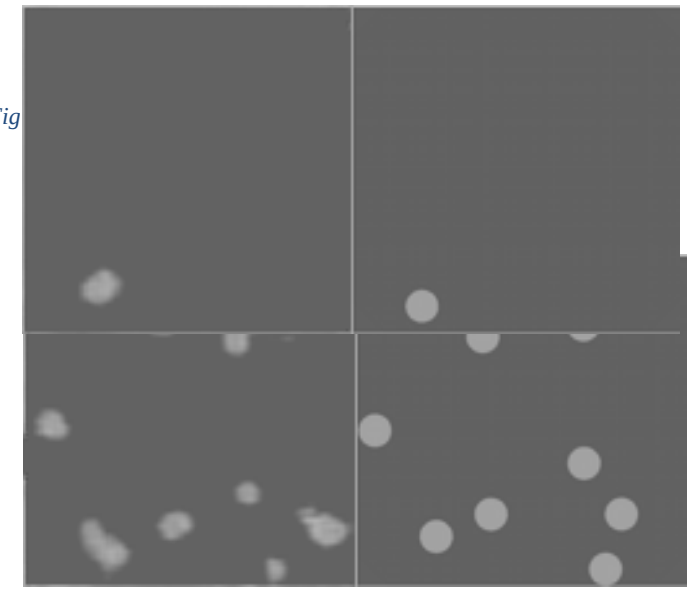


*Fig. 8 Sample images from our model trained on multiple objects, predicting multiple objects(top) predicting a single object (bottom)*

### *B. Predicting Many Frames into Future*

To test our true capabilities of our model, we designed an experiment where we give "seed" frames and use model to predict all consecutive frames in the dataset. However, our model in our current state, fails for future prediction. We observe that when our LSTM fed with generated codes, artifacts start to appear on the generated frames and the ball eventually vanished. We do observe some understanding of the physics where the artifacts appearing bounces from the walls. The test is conducted by giving 4 seed frames
and generating an output frame. This frame then added to our input code vector and the last code from the code vector is discarded, signifying a step forward in the time.
This sort of into-the-future type of experiments failed with our model. The new codes generated from the output image is not error-free enough for our model to compensate.

## VI. Discussion and Future Work

In this final report, we designed our physical model as proof of concept and showed that, it can generate future frames for artificial 2D environment. The network while demonstrating prediction over the frames, we sadly note that the model is constrained into an artificially created very simple environment. Even though it shows ability to transfer knowledge beyond its trained dataset, it does so only this environment. This might imply that our model will fail to transfer its knowledge over the real-life image sequences depicting physical interaction. The model still cannot create multi-frame predicted images, as it does not generate images with enough accuracy. Hence, the error over frames propagate over the network and result in images full of artifacts and eventual collapse of the physical state into nothingness.

While we believe that this problem can be alleviated by creating deeper convolutional layers for encoding-decoding network, the ability to generate real-life images from nearly real-world scene configurations seems slim.

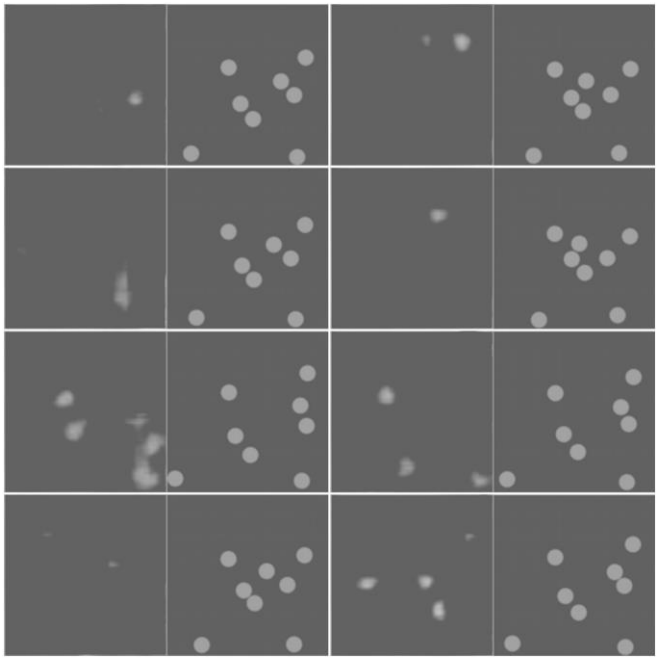


*Fig 9 Sample failure cases for the multi-frame prediction*

Additionally, our network could not incorporate the aforementioned physical features such as gravity, friction and bounciness of objects. There were no stationary objects except for the balls themselves.

Another interesting question we ask is whether we needed a recurrence predicting the future frames, as we know from our world, there is generally spatial continuity between frames which share temporal continuity i.e. consecutive frames have too many redundant information which can be used to identify them (which also substitutes for the hidden state). For this reason, using a feed-forward only convolutional network fed with temporally concatenated frames might perform better in terms of training.

Future directions of this project thus, include the use of non-recurrence as a future scene generation method, also testing the method on the real-scenery.

## VII. REFERENCES


[1] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A Compositional Object-Based Approach To Learning Physical Dynamics. arXiv preprint arXiv: 1612.00341, 2017

[2] S. Hochrieter, and J. Schmidhuber. Long Short-Term Memory. Neural Computation 9(8):1735-1780, 1997.

[3] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. arXiv preprint arXiv:1511.07404, 2015b.

[4] Battaglia et al. Interaction Networks for Learning about Objects, Relations and Physics arXiv:1612.00222, 2016

[5] Gao, Jiaying et al. "Dynamic Modeling Based On Fuzzy Neural Network For A Billiard Robot". 2016 IEEE 13th International Conference on Networking, Sensing, and Control (ICNSC) (2016):

[6] S. Eslami, N. Heess, T. Weber, Y. Tassa, K. Kavukcuoglu, and G. E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. arXiv preprint arXiv:1603.08575, 2016.

[7] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, W Woo, Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting

[8] Lecture on Deep Learning, http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[9] Grzeszczuk, Radek, Demetri Terzopoulos, and Geoffrey Hinton. "Neuroanimator". Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98 (1998): n. pag. Web. 24 May 2017.