Simulating Proto Planetary Disks with Deep Networks

by

Haotian Shen
B.Sc., University of Victoria, 2018

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Simulating Proto Planetary Disks with Deep Networks

by

Haotian Shen
B.Sc., University of Victoria, 2018

Supervisory Committee

Dr. Kwang Moo Yi, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-supervisor
(Department of Computer Science)

ABSTRACT

Proto planetary disks provide hints to how planets are formed. To understand them, it is necessary to run multiple simulations with various hyperparameters through trial-and-error. This procedure is typically time consuming as each simulation is expensive. In this project, we aim to solve this problem by learning a deep network that interpolate and extrapolate, given two simulation outcomes. We then iteratively apply this network to extrapolate simulations. Specifically, as proto planetary disks are circular, we make use of the log-polar representation of these disks and apply a circular 1D convolution on it. We empirically motivate our design choices via ablation study. Our experimental results show encouraging outcomes on approximating proto planetary simulations through extrapolation.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Kwang Moo Yi, for the guidance throughout the project.

Dr. Ganti Sudhakar and Wendy, for support, advising, and their patience in solving problems that I have encountered.

Dr. Ruobing Dong, Dr. Sébastien Fabbro and Dr. Karun Thanjavur, for providing the simulation data and their help in the astronomy related field.

Chapter 1

Introduction

In recent years, proto planetary disk has gained great interests in researches on planet formation. As a result of molecular cloud collapsing, a proto planetary disk is a rotating circumstellar disk in which planets are formed following physical laws. Most of the collapsing mass is collected at the center, forming a star, while the rest flattened into the proto planetary disk. Since the molecular cloud condition is non-trivial, proto planetary disks could form different number of planets, natural satellites and asteroids. For example, the Solar system is one final product of such proto planetary disk.

Real proto planetary disk data can be extremely expensive and time consuming to collect through existing astronomical instruments. Alternatively, computer simulation is a perfect tool to help researchers understanding the planet forming process better. Aside from this, simulation comes with consistent high precision data flow which is impossible to be gathered through the real world observation. Another advantage of computer simulation is that it provides astronomers a reliable way to verify their hypothesis. But a simulation run could still take several minutes to hours (depending on parameters like scene resolution) to generate a single snapshot of the evolving proto planetary disk, such as one shown in Figure 1.1. This makes the hypothesis verification process absurdly overlong, especially when the amount of simulation hyperparameters is large and can only be determined manually.

One way to address this is to provide simulators a video preview of the simulation outcome. This helps them to walk through ill hyperparameter combinations without waiting for the simulation to finish. To produce such preview, we first train a ResNet [1] based extrapolation network that takes two reference frames as inputs. We then produce the frame sequence by applying this network iteratively, with the previous

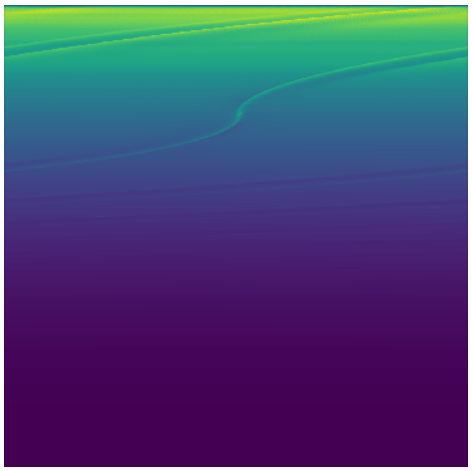
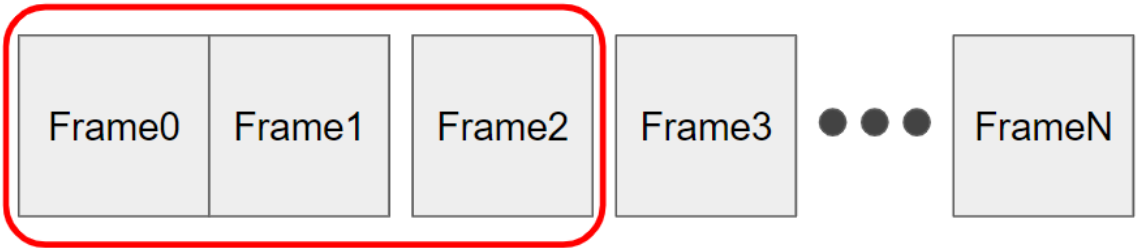


Figure 1.1: Simulation snapshot. The original data contains 4 channels representing particles of different sizes, our visualizations are based on the second channel to avoid confusion.

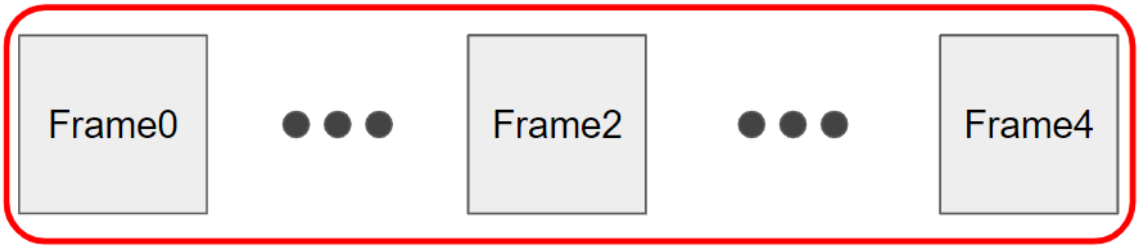
extrapolated frames as new input (Figure 1.2a). At a higher level view, our approach works by replacing the iteration function representing the planet formation process with a deep network. Benefit from the problem formulation, our networks also support frame interpolation by simply change the input frame order (Figure 1.2b).

The project accompanying this paper is broken into two major parts. The first part explains the proposed approach for frame synthesis, including a data pipeline that incorporates variable input sizes and two network models. We then visualize the leaning outcome of our current best models in the results section.

This report covers the design of our deep networks, an ablation study on network performance with different parameter setting and conclusions drawn from the experiments about the feasibility of using a single deep network to perform high quality frame synthesis on the simulation data.



(a) Iterative Frame Extrapolation



(b) Recursive Frame Interpolation

Figure 1.2: Extrapolation and Interpolation used to produce video preview. a) Iterative Frame Extrapolation. Given ground truth frame0 and frame1, the network predicts frame2. Using frame1 and the previous result frame2, the network predicts frame3, and so on. b) Recursive Frame Interpolation. To get frame1, the network produces frame2 first using ground truth frame0 and frame4. It then take frame0 and frame2 as inputs to produce frame1. So, our network need to perform single step extrapolation or interpolation shown in the red box only.

Chapter 2

Related Work

We first give a brief introduction of the simulation work. We then introduce heuristics that have been proposed for frame synthesis and review several deep learning based approaches in this field.

2.1 Proto Planetary Disk simulation

Proto planetary disks have been studied by astronomers to decipher the birth of planets [2, 3, 4, 5]. To obtain accurate analysis, effort has been made to develop better simulators. The original two dimensional dust radiation transfer code [2, 3, 4] was used to classify evolutionary stages on many other data collected by the Spitzer Space Telescope. Data provided by several observatories have revealed new 3D geometric structures and physical processes that were not included in the earlier codes [5], Whitney et al. [5] developed the three dimensional radiation transfer code to simulate the three dimensional structure, the Polycyclic Aromatic Hydrocarbon (PAH) molecule and the very small grain (VSG) emission. To resolve the inner edge of the disk in optical depth [2], Whitney et al. [2, 3, 4, 5] define the dust radiation transfer codes in the log-polar space.

In human vision system, the real world projected in the retinas of the eyes are reconfigured onto the cortex by a process similar to log-polar mapping [6]. And in computer vision, log-polar transformation has been shown useful to study the optical flow properties [7] and to improve pattern matching robustness against rotation and scaling [8]. For example, Polar Transformer Networks (PTN) [9] utilizes the log-polar transformation to provide equivalence for rotation and translation, avoiding

the fully connected layer required by the pose regression in the spatial transformer [10]. Therefore, the same log-polar transformation properties enable our networks to perform circular 1D convolution following a regular straight path.

2.2 Frame synthesis techniques

Frame synthesis is a fundamental computer vision and image processing technique. It is challenging because it involves the construction of an internal representation that models the frame evolution accurately [11]. This technique is widely applied in video compression standards (e.g. MPEG1, 2, 4) to reduce the frame temporal redundancy. One conventional way to perform frame synthesis is by doing motion estimation using two adjacent frames [12, 13]. The process of determine motion vectors involves various methods that utilizes all kinds of input frame features. Some of them are frequency domain (phase correlation [14]), corners (corner detection [15, 16]), texture & patterns (block-matching [17, 18, 19]) and the spatial-temporal image brightness variation (optical flow [20]). Usually, motion estimation is treated as a separate module in frame synthesis. Take motion estimation with optical flow as an example, the method typically requires to solve the optical flow equation

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

at each pixel location to get the velocity V at both x and y directions. And it could take an additional detector (e.g. Lucas-Kanade [21]) to find interesting features or to compute the flow vectors of the entire frame algorithms such as Farneback [22]. So, frame synthesis by traditional motion estimation method can be overly complicated.

2.3 Deep Networks

With advances of deep learning in computer vision, Convolutional Neural Networks (CNNs) has been demonstrated great potential in tasks including classification [23], semantic segmentation [24] and natural language processing [25]. Recent works [26, 11, 27, 28, 29] reveal the possibility of using deep CNNs for frame synthesis. Huaizu et.al [26] developed Super SloMo showing that state-of-the-art optical flow method [20], coupled with occlusion reasoning [30], can serve as a strong baseline for frame interpolation. Their method is composed of a bi-directional optical flow

estimation network and a flow interpolation network to predict the soft visibility map, which helps to occlude pixels that should not be seen in the final output frame. Generative CNN based approaches [11, 27] directly hallucinate RGB pixel values of the synthesized frames. Generative Adversarial Network (GAN) [28] has been shown the capability to generate high visual quality plausible images. In [11], a multi-scale GAN based network was proposed to predict the future frames of a video sequence. Zibo et. al [27] used a feature fusion network to refine the network input and use the refined input to perform frame synthesis on a second network.

One flaw of the traditional optical-flow-based methods is that they fail where flow estimation is challenging (large motions). This can be solved by computing the flow in a pyramid structure [31, 32, 33] or using the EpicFlow [34]. On the other hand, the deep-learning-based methods that hallucinate pixel values directly tends to produce blurry results [29]. Another well-performed approach Deep Voxel Flow (DVF) [29] combines the advantage of the traditional optical-flow-based methods and the deep learning approach. DVF interprets the network output as the pixel spatial translation offset (voxel flow). Then the computed voxel flow is used to calculate weights for a trilinear interpolation between sampled image pixels, and the resulting network learns to synthesize frames by flowing pixel values from existing ones.

However, most of them still require hand-crafted adjust on multiple deep networks for late frame synthesis, and these methods focus more on overall frame visual quality over predicted pixel values accuracy. In this work, the proposed approach does not involve a separate motion estimating network or occlusion reasoning process. By solely exploiting the network architecture and data representation, we are able to perform high quality frame extrapolation and interpolation within a single CNN.

CNNs may include pooling layers to reduce data dimension [35]. An inherent issue with pooling such as max pooling is the reduce of localization accuracy [36]. UNet [36] is one of such CNNs designed to keep the localization accuracy while performing pooling operations in the network. We adopted a similar UNet structure in our second model to mitigate the network performance drop caused by the variable input sizes on fixed network structure.

Chapter 3

Our Approach

Our goal is to predict a map of scalar value difference between the ground truth frame and one of the input frames at each pixel. To calculate the map accurately, we adopt a similar network architecture used in semantic segmentation but without applying softmax function to the network output.

In order to get a video preview of the proto planetary disk evolving process, the proposed method first synthesize the third frame using the given two ground truth frames. It then uses the synthesized output and the previous frame to produce subsequent frames iteratively (Figure 1.2a). Meanwhile, we apply tiling to incorporate super-high resolution inputs and thus ensure our network can be trained regardless of the input size. Together with one dimensional convolution and proper padding beforehand, our approach preserves border details completely.

We first give the formulation of the frame synthesis approach, then we justify the logic behind our network design. Finally, we present the resulting ResNet based deep network architecture for smaller inputs. In order to preserve the network performance for higher resolution input, we propose a second UNet [36] based deep network architecture.

3.1 Formulation

Denote the entire input dataset $X, X \in \mathbb{R}^{N \times H \times W \times C}$ where N, H, W, C are the number of frames, frame height, frame width and number of feature channels. To illustrate the relationship between the inputs, we use t for the index of the first input frame, i for the frame offset. Since the frames are snapshots taken from a proto planetary

disk simulation every 20.032 planet orbits, and each orbit takes the same amount of time, t and i can be understood as the simulation time and the relative time offset. The proposed approach then takes a triplet of (x_t, x_{t+i}, x_{t+2i}) as a sample. For extrapolation tasks, the network treats x_t and x_{t+i} as two reference frames, $Y = x_{t+2i}$ as the target frame. While the interpolation network takes both x_t and x_{t+2i} as the reference frames, and x_{t+i} the target frame. For ease of exposition, we only describe the formulation for extrapolation below.

Rather than hallucinate the target frame y from scratch, we consider learning the true scalar value offset D between the target frame and either one of the input frames an easier task for the proposed network. And we have:

$$D_{Y-t0} = Y - x_t$$
$$D_{Y-t1} = Y - x_{t+i}$$

where $t0, t1$ are abbreviations of the first input frame and the second input frame.

Generally, frames close to each other tend to have closer values at each pixel. Since x_t is closer to target frame Y than x_{t+i}, we expect $D_{t2-t0} > D_{t2-t1}$. We denote $\mathcal{F}(x_t, x_{t+i}; \Theta)$ the deep network which models a complex extrapolation or interpolation function. The output of F is an approximation of D that could be directly applied to one of the input frames (x_t or x_{t+i}, for extrapolation):

$$\hat{D} = \mathcal{F}(x_t, x_{t+i}; \Theta) \qquad \hat{Y} = x_{t+i} + D_{t2-t1}, \text{ since } D_{t2-t0} > D_{t2-t1}$$

Using a pixels-level weight map (computed by various methods) to perform a bilinear interpolation between two input frames is a conventional way of doing frame synthesis. Yet it does not support frame extrapolation. We use $\hat{D}$ to modify pixels scalar values on x_{t+i} directly and this design enables the network to learn both interpolation and extrapolation by changing the input frame order only.

The residual is defined as

$$R = \hat{Y} - Y = \hat{D} - D_{t2-t1}$$

which indicates area where most error occurs. We use it as a visual clue for model error distribution.

Such network $\mathcal{F}$ can be trained by minimizing the distance between the predicted frame and the ground truth Y, with $l = 1$ or $l = 2$:

$$\mathcal{L} = \|(\mathcal{F}(x_t, x_{t+i}; \Theta) + x_{t+i}) - Y\|^l$$

3.2 Data Pipeline

As our formulation requires three frames of equal frame gap i, we split the data into training, validation and test according to an odd and even splitting scheme to fully utilize the limited number of frames.

3.2.1 Dataset

The quality of dataset can directly impact the training output, especially for deep learning model. Our dataset is resulted from a proto planetary disk simulation run for 3004.5 orbits. Each one of the given 151 frames is a snapshot of the evolving system after 20.03 planet orbits which can be treated as a time unit. The original data are stored in 1024×1024 arrays, each contains 4 channels representing gas, dust of size 0.01cm, dust of size 0.1cm and dust of size 0.001cm. The number of channels varies from simulation to simulation. Showing all channels at once brings confusion and could be misleading, so we decide to use the second channel of all disk figures and training visualizations in the rest of the project.

According to our formulation, the proposed networks will take a triplet including two input frames and one target frame. For example, if we set $t = 0$ and $i = 1$, the dataset will give a total of 150 triplets. But if we define i as a range, say $1 \leq i \leq 75$, the dataset contains 5625 triplets. This is equivalent to train a network with the same video at 75 different FPS rate. However, later we found a huge network performance discrepancy between using larger and smaller i.

Comparing to action recognition dataset such as UCF101 which contains 13320 videos at 25 FPS, our dataset is extremely small and contains frame sequence from a single simulation run. This makes the proposed networks overfit easily.

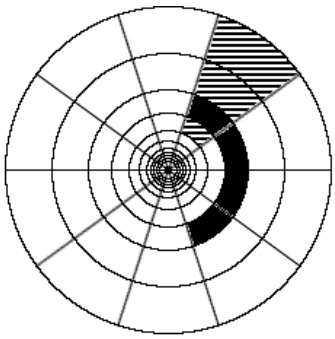
Two split schemes are used to divide the dataset into training, validation and testing set. The first scheme uses the earlier 75 frames as training data, the next 36

frames as validation data and the remaining 36 frames as test data. The second split scheme splits the data into odd and even indexed frames, we use the even frames for training, first half of the odd frames for validation and the rest for testing. These two schemes utilize the uniform frame gaps the most, we take advantage of the uniform frame gaps to maximize the number of triplets available in each dataset. Because the first split scheme contains frames next to each other ($i = 1$), we expect our models overfit the training data created by this scheme better. But, the image pattern in the validation and test split become so different from the training data, the resulting error calculation could be inaccurate for this split scheme. The second scheme avoids the previous problem. However, enforcing the odd and even criteria could introduce bias to both training and validation data, and it also doubles the frame gap i.

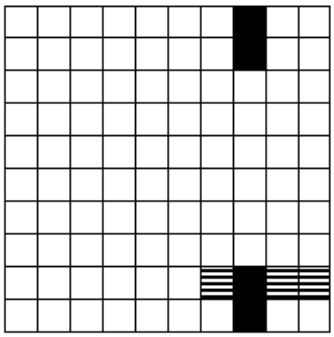
The original data has frames stored in separate binary files. We resize each frame to 32×32 before dividing them into train, valid and test set. Additionally, a log grid file is provided for reconstructing the data in polar with linear scale radius. H5py stores data in JSON-like format, and it allows parallel data read. For ease of data accessing and organization, we store the datasets into three h5py files, each with a copy of the same log grid file.

3.2.2 Log-Polar representation

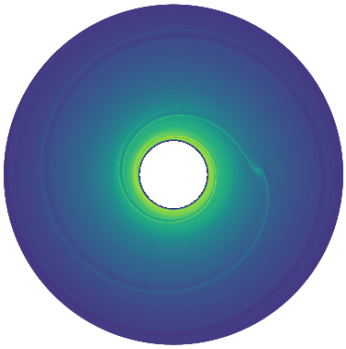
Data in log-polar representation helps our network to trace more pixel correlations. At the basic level, CNNs work by finding spatially-linked correlations like planets and their orbits. Intuitively, planets or asteroids orbiting a proto star creates a circular or ellipse shape in the Cartesian coordinates, and it is easier to find pixel correlations if our network convolute following the same path. As such, we prefer a semi-circular kernel in the Cartesian coordinate system. Since the given data is in polar with log scaled radius (X-axis), the rotation of planets are represented by translation in the input frames. Fig. 3.1 is an illustration of the mapping between the two representations. We also show the simulation data (transposed to align with the grid plot) in these two representations in Figure 3.1c and Figure 3.1d. For data in Cartesian representation, the disk center are removed for numerical stability, convolution over the excluded area in the center (shown in white) brings artifacts. On the contrary, we can pad the top and the bottom side of the log-polar representation with real values from the input itself, because both of them are naturally connected as one piece. So, we stick with the log-polar representation for network training.



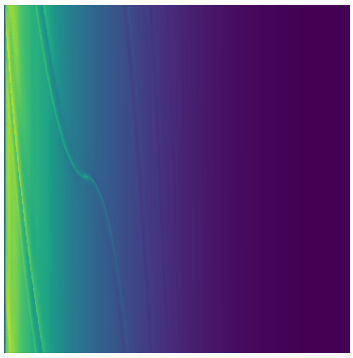
(a) Cartesian



(b) Log-Polar



(c) Data in Cartesian (Transposed)



(d) Data in Log-Polar (Transposed)

Figure 3.1: Mapping between Cartesian and Log-polar. a) Grids in Cartesian. The kernels should have a similar shape to the black band. b) Grids in log-polar. x-axis, the scaled radius. y-axis, the angle θ. Rotation in Cartesian maps to translation in log-polar along the y-axis. c) Data in Cartesian. We transfer the original data back to linear-polar using a log-grid file provided with the binary data files. Since no simulation is performed near the central star, we leave that area blank. d) Data in Log-Polar. The left side is the proto planetary disk center, where the right is facing outward. We visualize using the second channel only as the original image is multi-channel. (Images 2a and 2b are adopted from [37])

3.3 Network Design

In this section, we first discuss the feasibility of using one dimensional convolution in our network. We then justify our model implementation choices and compares the model performance under different network configurations.

3.3.1 One Dimensional Convolution

In a proto planetary disk, particles or planets usually has circular orbits. So, most of the frame features will be distributed in the angular direction in the log polar representation. The network is required to localize those features before predicting their movements, so translation variance is necessary in the angular direction. As we shown later in the results section, iterative extrapolation error increases exponentially, we must lower the error in the initial extrapolation result. However, 2D convolution with kernel height larger than 1 require extra padding, which brings artifacts at image boundary. We use one dimensional convolution to avoid padding at the frame top and bottom (Figure 3.2). By doing this, we also reduce the training time spent on network convolution in the radius direction. To switch between 1D convolution and 2D convolution quickly, we use 2D convolution layers with kernel height set to one. Then we reshape our input data and output data as needed. By using 1D convolution, we are able to achieve similar performance as using 2D convolution (Figure 3.3).

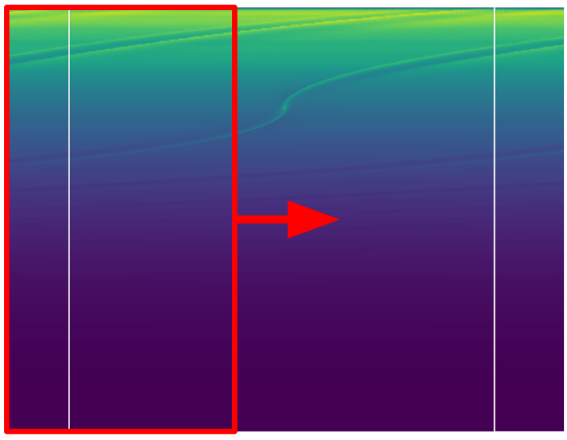


Figure 3.2: 1D Convolution. We perform the one dimensional convolution in the angular direction with kernel (the red box) that covers the entire radius direction, after applying cyclical padding.

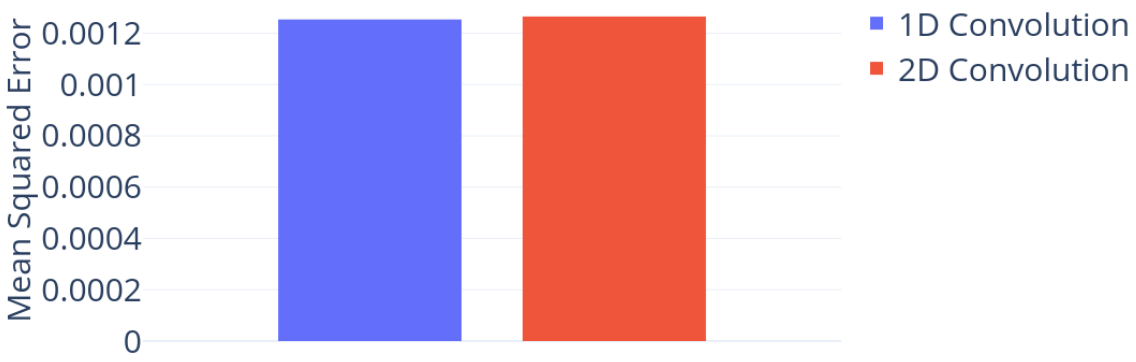


Figure 3.3: Performance comparison between one dimensional convolution and two dimensional convolution. There's no evidence showing we should use two dimensional convolution over one dimensional convolution.

3.3.2 Kernel size

Convolution kernels are filters applied to close together pixels. Those filters activate the neuron once the corresponding feature is found. With large number of training iterations, we hope the filter weights become close to their true values. Thus, they are able to capture motions or locate features such as the forming planet with high certainty.

In a high level view, kernel size determines the largest feature or motion that can be captured from the input frame. For instance, a 5×5 feature containing a human face will not activate a neuron with a 3×3 sized kernel. Even though the input frame is in log-polar, meaning that features are in a more compact and stable representation than that they are in Cartesian, we observed that our network performance is affected by the kernel size a lot. In Figure 3.4, increasing kernel size lowers the test error by a noticeably. This means that global features such as orbit location play an important role in the network decision making. However, those features scale with the input size and will lead to unstable model performance.

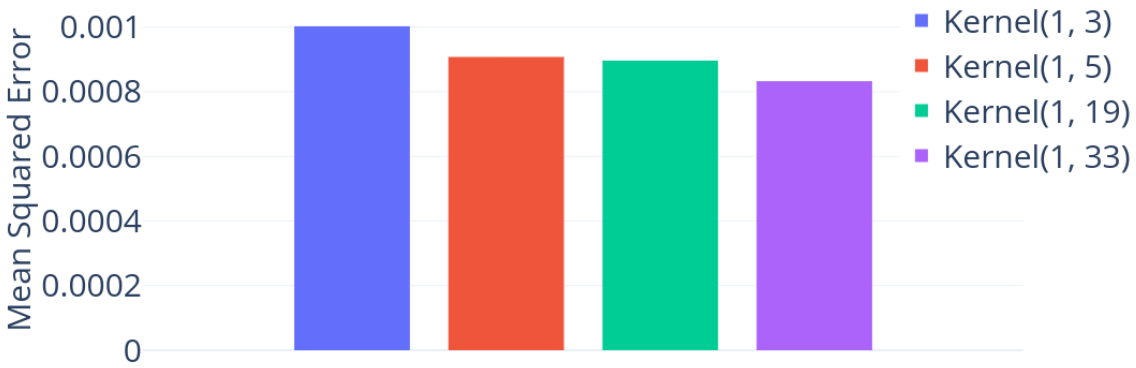


Figure 3.4: Performance comparison between smaller kernel size and larger kernel size with all other parameters fixed. Our experiments show that larger kernel brings noticeable decrease in loss using the testing dataset.

One way to overcome the problem of scaling features is by adding more layers to the network. Functionally, 5×5 kernels can be replaced by stacking two layers of 3×3 kernels next to each other. The reason is that they have the same receptive field

and larger features like faces could still be detected through a composition of smaller features (such as the nose, eyes) from earlier layers. Meanwhile, two layers of smaller kernel requires less weights (not true for our 1D convolution case) than one large kernel. In the project, a Resnet model is designed to have one input layer and one output layer along with 7 ResUnit in between, each ResUnit includes 2 convolution layers. This setup gives a total of 16 convolution layers. By using 1×3 kernels, the resulting model has receptive field covers the entire row of the 32×32 resized data.

For higher resolution inputs, the UNet model uses an encoding network to pool large features into a spatial compact representations and hence preserve the network performance.

3.3.3 Number of features

Earlier layers learn low-level features like lines while deeper layers learn much more complex features like faces. There are a lot more of those complex features we want to capture than there are low-level features. But it is rather hard to control the number of features at each layer. To avoid dimensional bottleneck, we simply choose to have 512 output features at all convolution layers.

3.4 Proposed Models

3.4.1 ResNet Model

Driven by the significance of network depth in a standard vanilla network where training is based on back propagation, adding extra layers to existing network does not necessarily improve the performance. Instead, it could lead to performance degradation caused by the notorious problem of vanishing or exploding gradients. This is addressed by adding skip lines that allows the gradient flow to pass through layers. Because of that, we start with a fully convolutional residual network[1], and we use this to perform ablations study. Fig. 3.5 illustrates the architecture of our first model, we use a 2 dimensional kernel to convolute in the frame angular direction only. This not only reduce the number of convolute operation in the network, leading to significant speed up, but also produces comparable results as the 2-dimensional convolution gives (Figure 3.3). After cyclical padding, we stack the reference frame channels and pass them to the network.

The network is composed of seven residual blocks plus one input and one output convolution layer. Additionally, a 1x1 convolution layer is added to the ResUnit to reshape the residual flow.

3.4.2 UNet Model

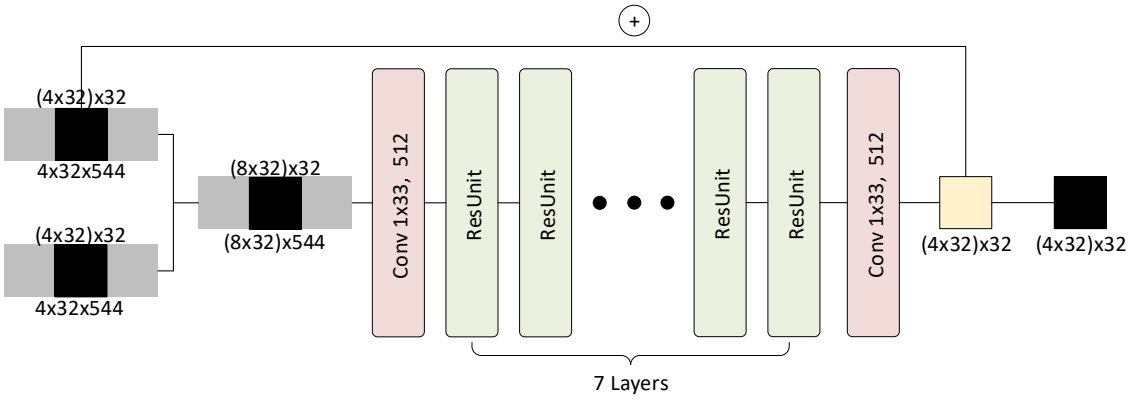
UNet is another fully convolutional neural network we explored for frame synthesis. This architecture includes an encoder network which is used to learn efficient representations of the raw input. As the name suggests, the decoder network decodes the feature representations and map them back to their original position. We choose this because the encoder reduces the raw input features spatial dimension and thus addressing the scaling problem caused by variable input sizes.

As Figure 3.6 shows, our UNet model contains three encoder layers, one bottleneck layer and three decoder layers. We make use of those max pooling layers to half the kernel sizes in the deeper layers, thus the resulting network takes exactly the same input size as the ResNet model does, so as to compare the performance of these two models. For higher resolution features, we just increase the kernel sizes at each layer of the UNet model.

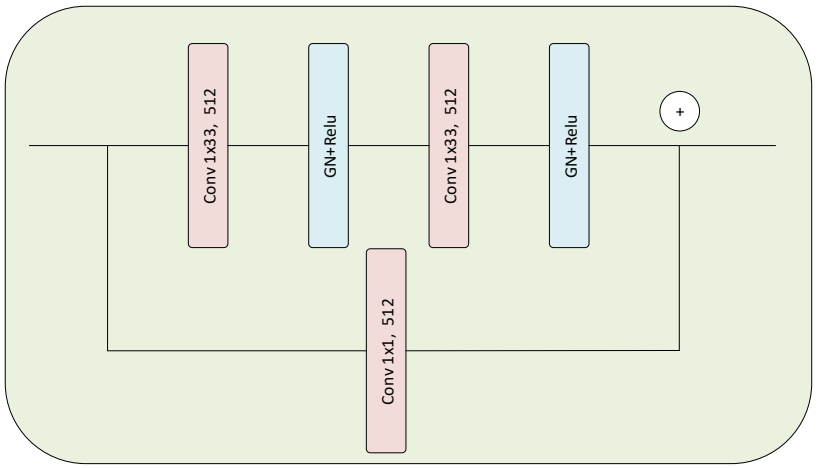
3.5 Training

All experiments in this project were ran on GPUs provided by three national clusters: Cedar (Tesla P100), Graham (Tesla P100) and Beluga (Tesla V100). The model is implemented in PyTorch with Adam optimizer, and it could reach 100% GPU utilization after code optimization. The training of our model is unsupervised and we determined 10^{-6} an appropriate learning rate to use. Since the dataset is small, we limit the batch size to 10 triplets. The training takes from 0.5 secs to 40 secs per epoch depending on the network configuration. Usually we stop the train at 100k or 200k iterations where a balance of time and stable error is reached.

During training, it is found that larger inter-frame gap i produces much higher validation loss (Fig 3.7). This is caused by the scalar value offset D being too large, and our network does not produce as accurate predictions when the two reference frames are too far away from each other: it comes naturally a harder problem to predict x_{t+100} given x_t and x_{t+50} than it is to predict x_{t+2} given x_t and x_{t+1}. Fundamentally, it is due to the decrease of transition linearity as time gap i increases.

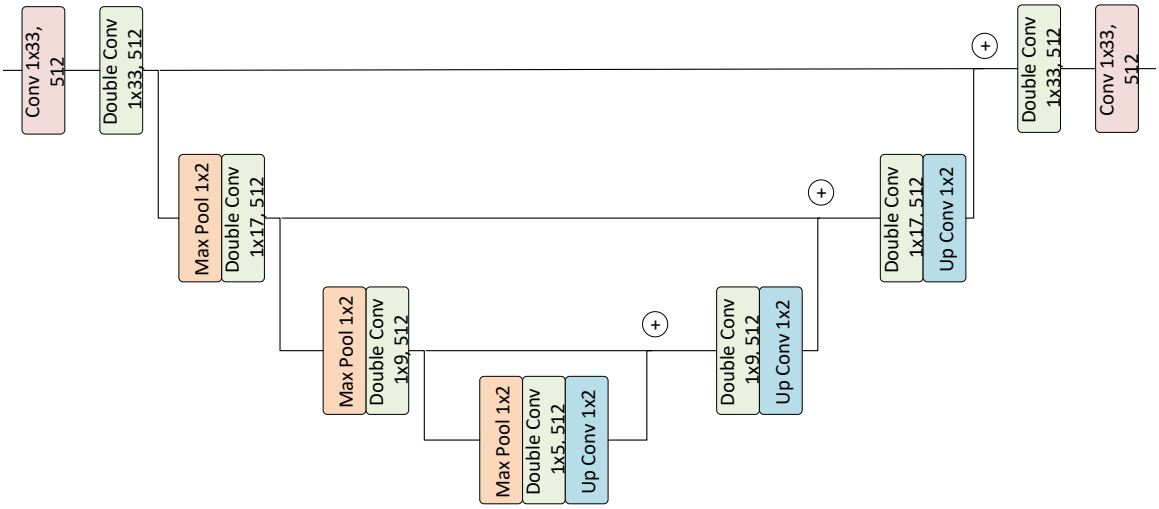


(a) Resnet model

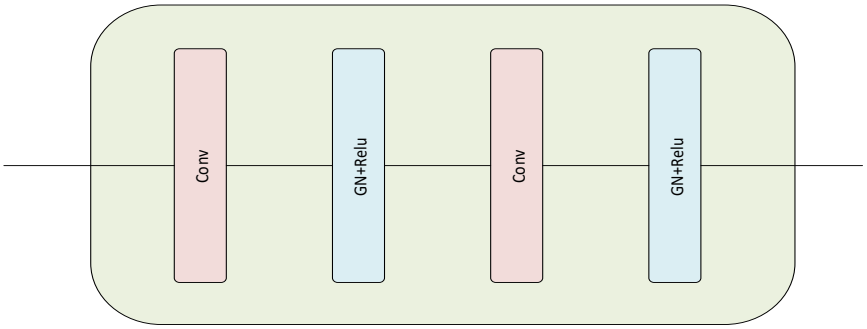


(b) ResUnit Block

Figure 3.5: ResNet Model. We removed the default padding on each of the convolution layers to reduce the approximation error at image borders. a) Architecture of the ResNet Model. The network takes a $(8 \times 32) \times 32$ image as input and output a (4x32)x32 color offset which is added to one of the input frames producing the synthesized frame $\hat{Y}$. b) ResUnit Block. We adopted a 1x1 convolution layer to reshape the residual flow. Instead of batch norm, group norm with parameter 32 is used.



(a) UNet model



(b) DoubleConv Block

Figure 3.6: The Proposed UNet model to address the feature scaling problem. a) Architecture of the UNet Model. b) DoubleConv Block. Instead of batch norm, group norm with parameter 32 is used.

Therefore, we set $i = 1$ after splitting the data to training, validation and test.

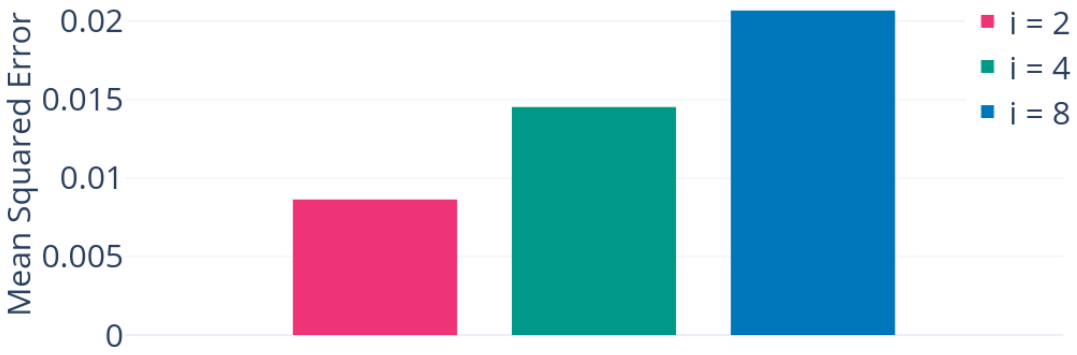


Figure 3.7: Performance comparison between large and small frame gaps. While our network overfits the training data well, the prediction performance on the validation split drops fast if the frame gap i set too large.

Chapter 4

Results

In this chapter, we provide qualitative and quantitative visualization for both frame interpolation and extrapolation. To produce visually comparable results, we use the odd & even split data.

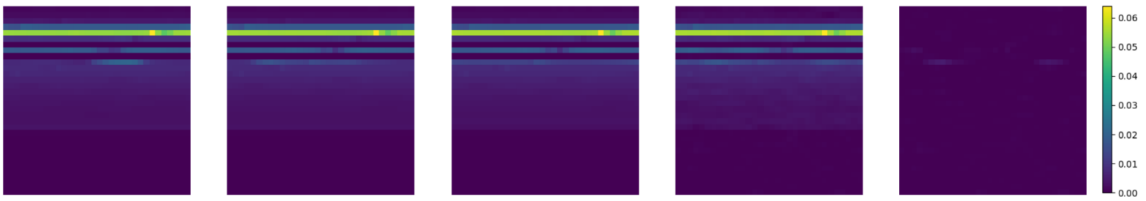
We train our models on 32 by 32 frames with maximum frame gap $i = 1$. The best ResNet model we have outperforms the proposed UNet model in both case of interpolation and extrapolation slightly. We summarize the performance of the two models in terms of Peak Signal to Noise Ratio (PSNR) using the test dataset in Table 4.1.

Metric	PSNR	
Mode	ResNet	UNet
Extrapolation	50.636	49.592
Interpolation	50.868	48.556

Table 4.1: Model Comparison in PSNR on test dataset. The ResNet model performed better than the UNet model in both interpolation and extrapolation.

4.1 Extrapolation

Our ResNet model is able to produce high quality extrapolation result as shown in Figure 4.1. One may notice several error spikes in the residual plot R (Figure 4.1a, 4.1b), which are mostly caused by the non-linear movement of features (e.g. the spiral arm) in the original dataset. To demonstrate this, we provide a video made of the original frames.

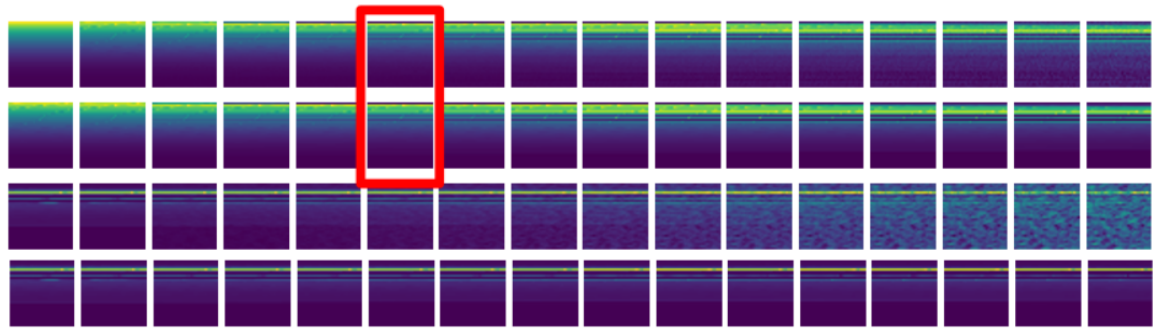


(a) ResNet based model single frame extrapolation visualization on the test data (Input1, input2, ground truth, synthesized frame, residual plot. Left to right)

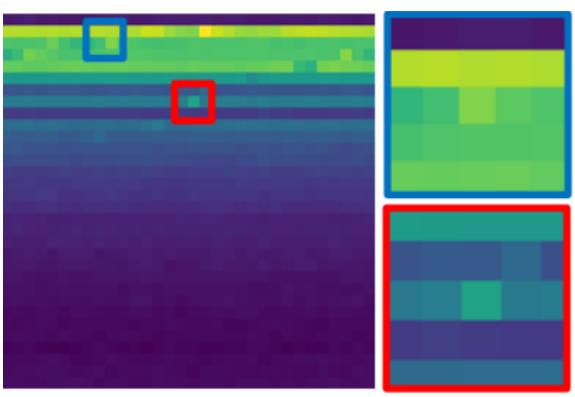
(b) UNet based model single frame extrapolation visualization on the test data (Input1, input2, ground truth, synthesized frame, residual plot. Left to right)

Figure 4.1: Extrapolation performance comparison between the proposed ResNet model and the UNet model. Visualized using the second frame channel. a) Extrapolation visualization on one of our test data triplet using the proposed ResNet. b) is the same visualization using the UNet model.

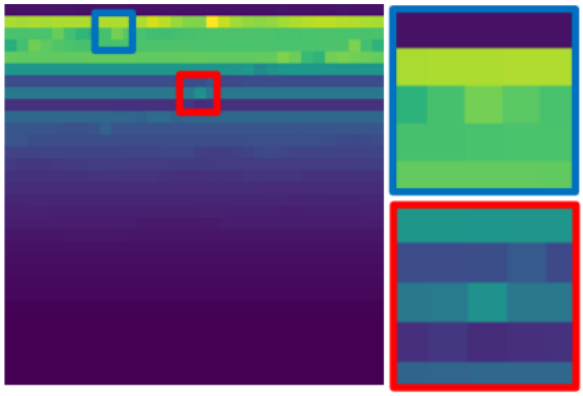
Given the first two ground truth frames from either validation data or test data, Figure 4.2 presents the frames that is iteratively extrapolated by our best ResNet model. The visualization shows that our model is able to produce visually convincing results for at least 8 frames after the ground truth. One drawback of such method is the exponential increase of cumulative error, as shown in Figure 4.3.



(a) Iterative extrapolation result



(b) Extrapolated frame



(c) Ground truth

Figure 4.2: Iterative extrapolation visualization. Visualized using the second frame channel. a) Extrapolation result using the ResNet Model. The first row is the extrapolation results based on the validation data. The second row is the validation data ground truth. The third row is the test data extrapolation results. The last row is the test data ground truth. We provide the initial two frames of the validation or test dataset to the network and let produces the rest. b) Detail at the forming planet at the 6th extrapolated frame. c) Detail of the forming planet at the 6th ground truth frame.

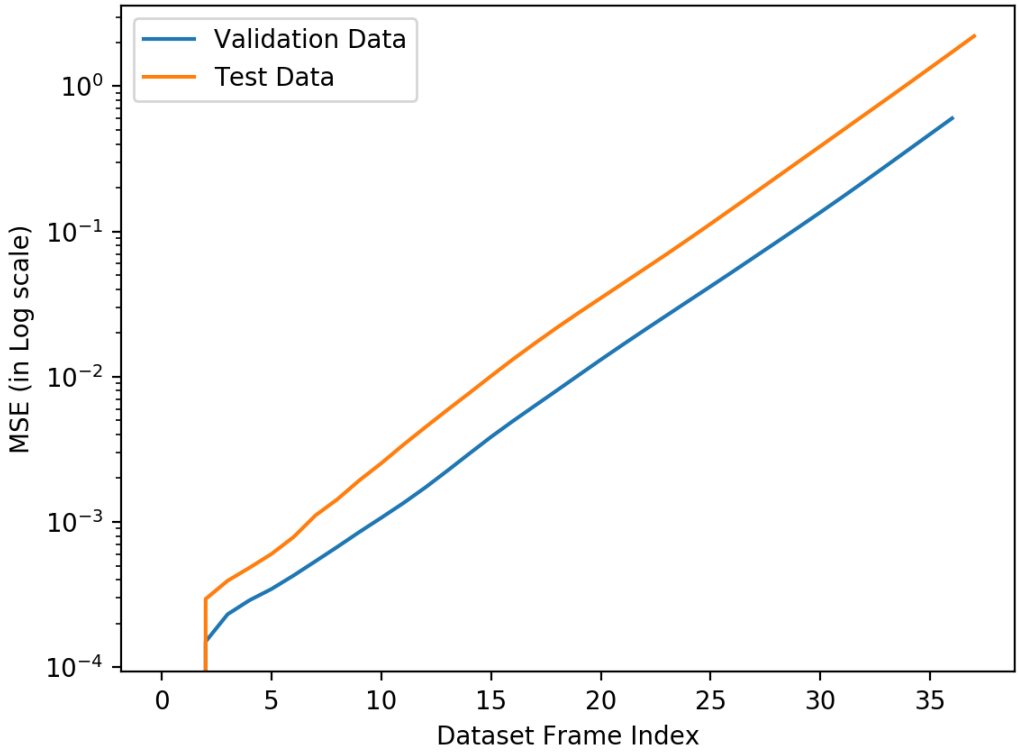
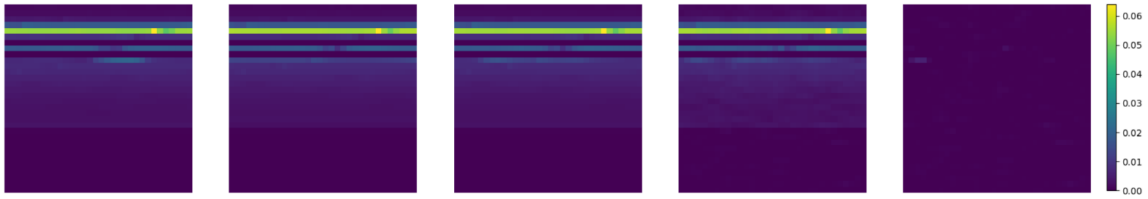


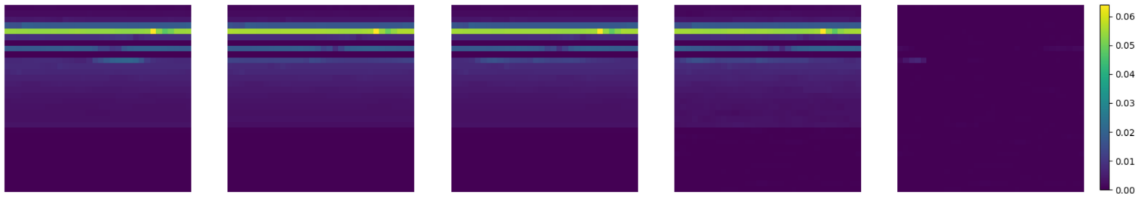
Figure 4.3: Cumulative extrapolation loss in producing a frame sequence using the validation and the test data with log scaled Y-axis, visualized using the second frame channel. The frame index is relative to the dataset being used.

4.2 Interpolation

For frame interpolation, we just change the network to take the first and the third frame of the triplet as its input. As a by-product of the proposed approach, the model interpolation performance is similar to the best we can achieve with extrapolation. In Figure 4.4, we visualize the single frame interpolation result using the same triplet as in Figure 4.1.



(a) ResNet based model single frame interpolation visualization on the test data (Input1, input2, ground truth, synthesized frame, residual plot. Left to right)



(b) UNet based model single frame interpolation visualization on the test data (Input1, input2, ground truth, synthesized frame, residual plot. Left to right)

Figure 4.4: Interpolation performance comparison between the proposed ResNet model and the UNet model, visualized using the second frame channel. a) Interpolation visualization on one of our test data triplet using the proposed ResNet. b) is the same visualization using the UNet model.

Chapter 5

Conclusion and Future Work

In the project, two deep networks are proposed to extrapolate the video frames. The resulting ResNet model learns to predict the pixel scalar value offset from two given reference frames better than our UNet model. Frame interpolation is also possible with our approach. Using the log-polar representation of the proto planetary disk simulation data, we show that one dimensional convolution with large kernel size yields performance similar to the 2d convolution counterpart and requires substantially less time to train. To reduce error in iterative extrapolation, we choose to apply cyclical padding beforehand and use 2D kernel with height set to 1. This helps to reduce the artifacts near the image boundary. Considering the frame sequence comes from a single simulation run only, our framework is highly limited by the small training dataset. We plan to expand the current dataset to include more simulation runs with different initial condition.

Since the current models are evaluated using 32 by 32 inputs, it will be interesting to see how these two models performs with higher resolution inputs. Future work will deal with the recursive interpolation performance of the two models. Another extension of this work could be the combination of the current approach and an extrapolation momentum.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," pp. 770–778, Jun 2016.

[2] B. A. Whitney, K. Wood, J. E. Bjorkman, and M. J. Wolff, "Two-dimensional radiative transfer in protostellar envelopes. i. effects of geometry on class i sources," *The Astrophysical Journal*, vol. 591, pp. 1049–1063, Jul 2003.

[3] B. A. Whitney, K. Wood, J. E. Bjorkman, and M. Cohen, "Two-dimensional radiative transfer in protostellar envelopes. ii. an evolutionary sequence," *The Astrophysical Journal*, vol. 598, no. 2, p. 1079, 2003.

[4] B. A. Whitney, R. Indebetouw, J. E. Bjorkman, and K. Wood, "Two-dimensional radiative transfer in protostellar envelopes. iii. effects of stellar temperature," *The Astrophysical Journal*, vol. 617, no. 2, p. 1177, 2004.

[5] B. A. Whitney, T. P. Robitaille, J. E. Bjorkman, R. Dong, M. Wolff, K. Wood, and J. Honor, "Three-dimensional radiation transfer in young stellar objects," *The Astrophysical Journal Supplement Series*, vol. 207, Jul 2013.

[6] E. L. Schwartz, "Anatomical and physiological correlates of visual computation from striate to infero-temporal cortex," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-14, pp. 257–271, Mar 1984.

[7] H. Araujo and J. Dias, "An introduction to the log-polar mapping," *Proc. of Second Workshop on Cybernetic Vision*, Jan 1996.

[8] T. V. Javier and P. Filiberto, "The log-polar image representation in pattern recognition tasks," in *Pattern Recognition and Image Analysis*, (Berlin, Heidelberg), pp. 1032–1040, Springer Berlin Heidelberg, 2003.

[9] C. Esteves, C. Allen-Blanchette, X. Zhou, and K. Daniilidis, "Polar transformer networks," Feb 2018.

[10] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2017–2025, Curran Associates, Inc., 2015.

[11] M. Mathieu, C. Couprie, and Y. Lecun, "Deep multi-scale video prediction beyond mean square error," Nov 2015.

[12] D. Mahajan, F.-C. Huang, W. Matusik, R. Ramamoorthi, and P. Belhumeur, "Moving gradients: a path-based method for plausible image interpolation," *ACM Transactions on Graphics*, vol. 28, no. 3, p. 42, 2009.

[13] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, pp. 1–31, Mar 2011.

[14] H. Foroosh, J. B. Zerubia, and M. Berthod, "Extension of phase correlation to subpixel registration," *IEEE Transactions on Image Processing*, vol. 11, pp. 188–200, Mar 2002.

[15] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, pp. 147–151, 1988.

[16] J. Shi and C. Tomasi, "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[17] A. Barjatya, "Block matching algorithms for motion estimation," *IEEE Transactions Evolution Computation*, vol. 8, pp. 225–239, Jan 2004.

[18] C. Je and H.-M. Park, "Optimized hierarchical block matching for fast and accurate image registration," *Signal Processing: Image Communication*, vol. 28, no. 7, pp. 779–791, 2013.

[19] S. T. Khawase, S. Kamble, N. Thakur, and A. S. Patharkar, "An overview of block matching algorithms for motion vector estimation," *The Second International Conference on Research in Intelligent and Computing in Engineering*, pp. 217–222, Jun 2017.

[20] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1647–1655, 2016.

[21] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, vol. 2, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[22] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis* (J. Bigun and T. Gustavsson, eds.), (Berlin, Heidelberg), pp. 363–370, Springer Berlin Heidelberg, 2003.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, May 2017.

[24] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, Jun 2015.

[25] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, (New York, NY, USA), pp. 160–167, ACM, 2008.

[26] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. G. Learned-Miller, and J. Kautz, "Super slomo: High quality estimation of multiple intermediate frames for video interpolation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9000–9008, 2017.

[27] Z. Gong and Z. Yang., "Video frame interpolation and extrapolation." 2019.

[28] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[29] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala, "Video frame synthesis using deep voxel flow," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 4473–4481, 2017.

[30] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, pp. 1–31, Mar 2011.

[31] F. C. Glazer, *Hierarchical motion detection.* PhD thesis, University of Massachusetts, Amherst, MA COINS TR 8702, 1987.

[32] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2720–2729, 2016.

[33] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," 2000.

[34] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Epicflow: Edge-preserving interpolation of correspondences for optical flow," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1164–1172, 2015.

[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[36] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), pp. 234–241, Springer International Publishing, 2015.

[37] L. Hui, L. Liang, D. Peijun, and S. Huasheng, "Image registration based on fourier-mellin transform," *GEOMATICS AND INFORMATION SCIENCE OF WUHAN UNIVERSITY*, vol. 37, no. 6, p. 649, 2012.