DATUM: Dotted Attention Temporal Upscaling Method

Steven Walton

Abstract— Computational simulations frequently only save a subset of their time slices, e.g., running for one thousand cycles, but saving only fifty time slices. With this work we consider the problem of temporal upscaling, i.e. inferring visualizations at time slices that were not saved, as applied to ensemble simulations. We contribute a new algorithm, which we call DATUM, which incorporates machine learning techniques, specifically, dotted attention and convolutional networks. To evaluate our approach, we conduct 1327 experiments, on 32x32 pixel renderings of two-dimensional data sets. Our experiments infer imagery at unsaved time slices and compared to ground truth renderings both visually and with an established metric (peak signal-to-noise, or PSNR). We also compare to a linear interpolation method, and find that our technique has a significantly higher accuracy, in some cases producing renderings that are 19% more accurate. Overall, we demonstrate that our method can learn patterns from a single simulation within an ensemble and use this information to perform temporal upscaling on other simulations within the same ensemble that are sparsely saved. We show that with 1% of data from a new simulation, equivalent to a simulation saving imagery one out of every hundred cycles, is enough to improve accuracy for temporal upscaling.

1 Introduction

Many scientific visualization usages involve analyzing time-varying data, and yet typically only a subset of the temporal data is available to visualization routines. Saying it another way, if a field is defined as $F(T,P)$ for time T and position P, then typically there exist a discrete set of times, $t_0, t_1, t_2, ..., t_N \in T$, where the data is available to a visualization program, i.e., $F(t_i, P)$ is defined. For all other times, the field is not defined, i.e., if $T^\star$ is not one of $t_0, t_1, t_2, ...t_N$, then $F(T^\star, P)$ is not defined.

While complete spatio-temporal data is desirable for analyzing time-varying data, it is often not practical to obtain. For experimental and observational data, the temporal data obtained may be limited, for example, by the rate at which sensors can ingest data. For computational simulations, there are two reasons. The first and more minor reason is that the simulation itself only defines the field at a discrete set of times. Simulations typically advance in "cycles," with a simulation starting a cycle at some time T and ending at some time $T+\varepsilon$. Field values for times $T^\star$ where $T < T^\star < T+\varepsilon$ are not defined. The second and more major reason is that the simulation cannot store data from all of its cycles, due to prohibitive costs. These prohibitive costs involve not only the execution time to interact with I/O, but also disk space to represent many bytes (up to exabytes).

There are two strategies for obtaining additional temporal data. The first strategy is to "do it again": re-run the simulation or experiment. This is sometimes possible, but not always, as the temporal frequency may be limited by sensor technology, disk storage, etc. The second strategy — called temporal upscaling — is to use software to infer data or imagery at the missing times. The "do it again" strategy offers more certain results, but can have high costs. The temporal upscaling strategy can be done quickly, but with less accurate results.

With this work, we consider the strategy of temporal upscaling, and it is the premise of this work that temporal upscaling is useful for domain scientists due to its low costs. This premise matches other recent works, as temporal upscaling, as well as generally inferring imagery that is costly to obtain, has recently become an active research area [11, 14].

The contribution of our work is a novel, machine learning-based algorithm for temporal upscaling. Our algorithm takes a novel approach and provides a novel capability:

- Novel approach: we believe that we are the first to incorporate a dot-product attention based and fully convolutional solution to temporal upscaling.
- Novel capability: we provide a "transfer learning" function. That is, our algorithm studies one simulation and then applies ("transfers") the knowledge to another.

Using machine learning techniques, our algorithm increases temporal upscaling accuracy by as much as 19% compared to linear interpolation. In addition, the "transfer learning" capability has significant practical benefit in the context of ensemble simulations. In this case, one ensemble member can save data at high temporal resolutions (for training), and then the remainder of the simulation can output at low temporal resolutions to save costs.

Finally, our approach currently only considers a limited set of use cases, but we feel that it has significant potential for future improvement. First, our approach currently operates on "thumbnail"-size imagery due to memory and computational constraints. That said, recent innovations in machine learning have demonstrated that such techniques can be scaled up to more typical sizes fairly easily. Second, our experiments evaluate performance on a two-dimensional data set, which somewhat simplifies the problem. Regardless, we believe the nature of the machine learning approach will be applicable to three-dimensional renderings as well.

2 Related Work

Herein we introduce the related work to our own. As machine learning is a new and quickly evolving area of research within the scientific visualization community, there is not much work that is significantly similar. While this creates ample opportunity to expand on our community's scope of knowledge, it also does not provide foundational work to build upon.

2.1 Video Upscaling With Machine Learning

Video super resolution has become an active research area within the machine learning community [22, 28, 33, 42]. There are many works showing generation of video frame upscaling. For example, Google's DAIN [3] and Jiang et al's Super SloMo [17] shows how machine learning can be used to enable slow motion video by significantly upscaling the video frames. Bao et al. show MEMC-New [2] which can de-blur image motion and upscale videos with convolutional neural networks. There are many other works focusing on temporal upscaling and solving challenges of video enhancement [27, 34, 41], solving specific challenges in upscaling such as motion blue and denoising. These works have shown how machine learning can be useful in making smoother videos, even going as far as upscaling videos from the early 1900's to 60FPS and 4K.

2.2 Sequence Learning With Attention

Another active field in machine learning research is understanding the relationships between sequences of data. Many use a network mechanism called Attention [38]. Attention has been shown to be a

powerful method in machine translation and text sequencing because of its ability to learn sequences and patterns between words, even in long texts [6, 8, 24, 32, 36]. These works have shown that these various attention mechanisms are able to perform long sequence to sequence transformations.

2.3 Machine Learning in Scientific Visualization

Within the scientific visualization community, the adoption and application of machine learning techniques is an emerging subfield. Researchers have focused on taking advantage of new many-core architectures and have leveraged machine learning to improve *post hoc* analysis [4, 10] and reduce reliance on human analysis [1, 16, 19, 30, 35]. Berger et al. [4] demonstrated the use of GANs for volume rendering and analysis tasks. Weiss et al. [37] showed how super-resolution techniques can be applied to volumetric isosurfac rendering. In the domain of flow visualization, FlowNet [10] addresses the challenge of selecting a representative set of streamlines or streamsurfaces. FlowNet uses an autoencoder to learn integral curve feature descriptors, followed by dimensionality reduction and clustering to identify and visualize representative flow features.

There is also much work in super resolution techniques, where researchers take a low resolution image or dataset and use machine learning to increase the resolution. Zhou et al. [44] show that convolutional networks can be used for volume upscaling, applying super resolution techniques to volume data. Xie et al. [40] and Wiewel et al. [39] both presented methods that can generate high resolution images in fluid flows. Upscaling the quality of images has been an active research area within scientific computing.

2.4 Machine Learning to Infer Data

The work herein focuses on temporal upscaling and transfer learning. While there are several machine learning papers on the subject [9,25,29] there are two notable papers that are related to our work. The first is Han and Wang's TSR-TVD [11] where they developed an algorithm based on ConvLSTM [31] to develop a temporal upscaling algorithm. Within this work they perform temporal upscaling by using sub volumes from voxel data at cycles i and $i+k$, where k is some distance in the range of 1-7. They use multiple ConvLSTM layers to create a Generative Adversarial Network (GAN), which uses a generator and discriminator model to infer new intermediate cycles. This work demonstrates temporal upscaling with voxel data. Our work differs from this not only in our model, but in the algorithm we developed for visualization practitioners. Our work is distinct in that it is applied to ensemble simulations and focuses on temporal upscaling of unseen and highly sparse sets of images from a new simulation than what our model was trained on.

The second relevant work is that of InSituNet [14], by He et al. InSituNet accepts and trains using ensemble simulation data generated considering varying simulation input parameters. The model can then be used to predict simulation outcomes for unseen simulation input parameters.

InSituNet was developed to learn the relationship between parameters of ensemble simulations. This work also uses a GAN to produce representations of differing ensemble simulations after learning from several examples. Our work differs significantly from InSituNet in that we are focusing on the temporal upscaling problem and not trying to completely predict what a new ensemble may look like. We are also trying to learn the temporal relationship between data instead of the relationship of the input parameters.

Our work differs from these works because we are focusing on a temporal upscaling applied to ensemble simulations. Han and Wang perform upscaling applied to a single simulation whereas InSituNet tries to solve a whole ensemble. Our work is closer to Han and Wang's but we focus on the transferability of temporal upscaling and how we can use these techniques in ensemble simulations.

3 Machine Learning-Based Algorithm for Temporal Upscaling

This section describes our algorithm. It begins by formally defining the problem we address (3.1). It then describes the machine learning building blocks we incorporate in our algorithm (3.2). This section concludes with a full description of our algorithm (3.3).

3.1 Problem Definition

Intuitively, temporal upscaling takes two adjacent visualization renderings as input and produces intermediate renderings between the two. More formally, we define our temporal upscaling problem as follows. Our algorithm operates on 2D scalar data, or greyscale images, which are equivalent to fields of data. Given two fields, $F(t_i,P)$ and $F(t_{i+j},P)$, we wish to find an intermediate field, $F(t_k,P)$, where $i<k<i+j$. Thus we wish to find some mapping function $N(F(t_i,P),F(t_{i+j},P),x)\mapsto F(t_k,P)$. Here N represents our approximation function, $F(t_i,P)$ and $F(t_{i+j},P)$ are our input images, x represents a tensor of temporal parameters, and $F(t_k,P)$ is the field we are trying to approximate. Because N is an approximation function it will never exactly map to F so we say that N maps to some field that is approximately F. In this manner our approximation function solves the following equation.

$$F(t_k,P)\approx F'(t_k,P)=N(F(t_i,P),F(t_{i+j},P),x) \tag{1}$$

The idea behind transfer learning is that if we train a machine learning model from one set of data, and then keep training the same model using a slightly different technique on a second dataset but with significantly smaller size, then the model will give similar results as if it was fully trained on the full amount of the second dataset. Formally, we define our transfer problem as follows. An ensemble of simulations, $\{S_0,S_1,\cdots,S_n\}\in E$, is defined as a set of simulations that follow the same basic rules but have differing parameters. That is $S_0=S(X_0)$ is distinct from $S_1=S(X_1)$ where X_i represents the set of input parameters to the simulation, thus there is some relationship between S_i and S_j. Transfer learning is defined as using a network trained on some dataset, S_i, on another distinct dataset that it was not trained on, S_j. If we simplify Equation 1 to $N_i(S_i)=N(F(t_i,P),F(t_{i+j},P),x)$, we can say that $N_i(S_i)\approx N_j(S_j)$. Here $N_i(S_i)$ is the algorithm N fully trained on the dataset S_i, and similarly $N_j(S_j)$ being trained on S_j We can gain a better approximation if we add a small amount of data from the new ensemble, εS_j, and improve the results of our approximation. That is

$$N_j(S_j)\approx N_i(S_i)+N_i(\varepsilon S_j) \tag{2}$$

3.2 Machine Learning Components

This section describe different machine learning building blocks, and also informs their usage within our model.

3.2.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) have been shown to be effective in performing image based analysis and have become the *de facto* tool for machine learning on images [5, 12, 20]. CNNs have a significant advantage over linear layers due to their use of sliding windows over data regions. This window, called a kernel or filter, is able to learn relationships between adjacent data points. This is particularly useful for image-based work where these relationships are necessary to understand complex structures. The other advantage of CNNs is that they are extremely computationally efficient [7, 15]. As a result, CNNs have become common practice.

The kernel is the main attribute to a CNN: it holds the learned information from training. A convolutional kernel is composed of four parts: a volume size W, a kernel size K, a padding size P, and a stride S. Each of these parts maps to our problem as follows:

- W represents either the width or the height of the image.
- K is set to 3 for all our CNNs, which correlates to each window having 9 data points. In effect, this is a $3x3$ window centered around each target pixel.

- P is the number of padding values, which are usually 0's, placed around the data so convolutions can be performed on data edges. In our case ,we set the padding size to 1, which is the minimal size for our filter to process each edge pixel.
- S is the step size our kernel takes when moving. We selected a stride of 1, which corresponds to the window focusing on the next adjacent pixel.

Given these parameter choices, the convolution operation produces an output with the exact same dimensions as the input.

3.2.2 Scaled Dotted Attention

Attention is a recent machine learning technique where relationships are learned between different features. Attention's main benefit is that it is able to operate on sequences of features. This technique is commonly applied to Natural Language Processing (NLP) workloads, so a model can gain an understanding of the relationships between words. To date, attention has been demonstrated to be a useful and powerful mechanism for learning sequence tasks [8, 30, 32, 36] Attention's main advantage over other sequence-based models, such as Recurrent Neural Networks (RNNs), is that its sequence of inputs does not need to be provided in order. This allows this mechanism to be used in parallel, and thus is frequently more computationally efficient.

In our work, we use the Scaled Dot-Product Attention mechanism. Attention is defined by three learned parameters, Q, K, V, that represent a "query," "key," and "value," respectively. Its equation is defined as:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{n}}\right)\mathbf{V} \tag{3}$$

Where n is the dimension of the data. A variation of this attention mechanism introduces a learned parameter γ such that our attention mechanism is altered as follows:

$$Attention = \gamma(softmax(\mathbf{Q}(x)\mathbf{K}(x)^T)\mathbf{V}(x)) + x \tag{4}$$

We use this form and follow the same attention mechanism described in SaGAN [43]. This uses a 1x1 convolutional layer to learn the Q, K, V parameters and then γ is learned instead of using the $\sqrt{n}$ term. Additionally, we sum with the original input. This allows this model to adapt better and weight the importance of attention in each layer. For example, if attention is unneeded in a layer, then $\gamma \to 0$.

3.2.3 Residual Networks and Skip Layers

Residual Neural Networks (ResNet) are neural networks that incorporate skip layers, where information flow can skip some layers to quickly reach future layers [13].

Residual networks have several advantages. First, in very deep networks, they can help simplify the model, as a network may be too deep and become less effective. Second, later layers can receive fresh information from any earlier layer. As an example, in image processing, earlier layers tend to learn high-level features, while the later layers tend to learn low level features. By combining them one can provide output with both high-level and low-level features.

3.2.4 AdamW

AdamW is an alternative to the commonly used Adam stochastic optimization method. Adam is an adaptive gradient method that is commonly used in machine learning because it is able to optimize quickly and avoid local minima. It has been shown that adding weight decay (thus AdamW) — as opposed to L_2 regularization — provides significant improvements to finding optimal choices [23].

3.3 DATUM

Our contribution is a novel method for temporal upscaling on ensemble simulations, which is named *DATUM*: Dotted Attention Temporal Upscaling Method. DATUM uses a machine learning model with Dotted Attention, which is essential to its success, and a methodology to obtain accurate temporal upscaling using this model. This subsection describes both the algorithm and the model: the algorithm is described in 3.3.1, treating the model as a black box; the model is described at a high level in 3.3.2 and in depth in 3.3.3.

3.3.1 Algorithm Description

DATUM is more than a machine learning model, but also a method to provide accurate temporal upscaling on ensemble simulations. Our method uses the following algorithm:

- Phase I
    1. Running a single ensemble where images are saved out at a high temporal frequency.
    2. Training an attention based machine learning model until there is a satisfactory convergence.
- Phase II
    1. Run a set of new ensembles where images are saved at lower frequencies that satisfy storage constraints.
    2. Fine tune our time machine learning model with this low temporal frequency image data (transfer learning).
- Phase III
    1. During post hoc analysis when high frequency imagery is required, generate missing images on the fly using the machine learning model.

DATUM works by training an attention based machine learning model on a set of images with high temporal fidelity and then fine-tuning on a sparse representation of a new simulation from the same ensemble. In Phase I, the model is able to learn both temporal – from the attention layers – and spatial – from the CNN layers – parameters from the high-frequency training ensemble simulation. In Phase II, the model is able to adjust the learned temporal and spatial parameters based on how a different set of parameters changes a simulation's outputs. This is because ensembles utilize the same physics and there are similar patterns in how these simulations evolve. For example, a expanding wave follows a consistent pattern across different simulations, which can be learned and re-used within our method. DATUM is not expected to work well when simulations are using differing physics. In Phase III, the tuned model will be able to infer missing images at a fast enough rate that it could be used on the fly along with post hoc analysis routines.

3.3.2 High-Level Model Description

To support our algorithm, we developed a novel end-to-end convolutional neural network with skip-layers and attention layers. Our model takes three inputs: two are images and one is a set of three temporal parameters. We used 32x32 2D images as our image input as this has a lower memory footprint, enabling us to train faster on our GPUs. Training on smaller images is a standard technique in machine learning and we expect our method scale to larger sizes, but more testing needs to be done. The first two temporal parameters correspond to the cycle numbers of the two input images and the third corresponds to the cycle number of the desired image. The output of the model is simply the desired image at the cycle corresponding to the third temporal parameter.

Our model is made of a series of CNNs, Attention layers, and skip layers. A diagram of our model is shown in Figure 1. Our images begin by going through a preprocessing CNN that is followed by two attention layers. These preprocessing CNNs are composed of three layers. These CNNs all have a single channel in and out. We use an adaptive average pooling on our parameters with an output size the same as the image size. The outputs of the two preprocessing networks and the average pooling are then concatenated. This concatenation is then processed by a pretrained VGG11 model [18]. This is then followed by two attention layers. The output of this is then concatenated with the input to VGG11. Finally this is processed by three convolutional layers, with channels 515, 256, and 1, respectively. This result ends with a tensor the same size as our original image and with a single channel, representing a greyscale image. It is trivial to modify the network to work with three channel color images, but this will require more memory overhead and processing time.

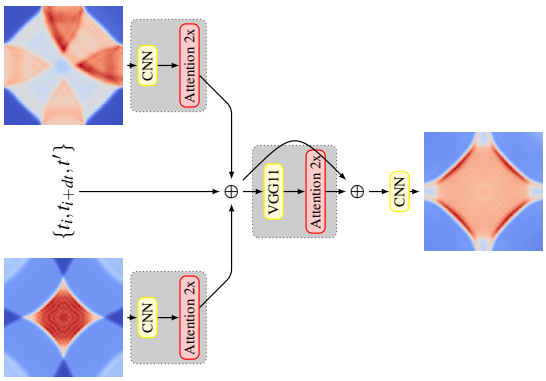


Fig. 1: Diagram representing our machine learning model. Subnetworks are highlighted with a bounded box for added clarity.

3.3.3 Detailed Model Explanation

Our network is fully convolutional, meaning that each learned parameter is done with a convolutional network. From the beginning of this network, we compose each convolutional block of a convolutional layer with kernel size 3, stride of 1, and padding of 1, followed by a ReLU rectifier. Each image is processed by three convolutions which each have a single channel. This is so that the convolutions learn high-level features from the data set. The intent of this decision is to utilize the information from the images to gain some understanding of the physics of the simulation. These convolutional layers are subsequently followed by two single-headed dotted self attention layers. These attention layers follow the same pattern as that from SaGAN as described in Section 3.2.2. This subnetwork is highlighted in a gray box in Figure 1 directly after the input images on the left.

The temporal parameters ($\{t_i, t_{i+dt}, t'\}$ in Figure 1) are an important part of the model that tells it how much to weigh each input. We use an adaptive average pooling method directly on these terms, generating output at the same size as the output of the convolutional layers. Then these parameters represented as tensors are concatenated with the tensors representing the output from the convolutions. Concatenation is represented by the $\bigoplus$ symbol in Figure 1.

In the middle of our network we use a pretrained version of VGG11 that helps determine how to weigh these inputs (the middle gray box in Figure 1). We use a pretrained network because this helps our model train faster — VGG11 has been trained on image classification and already has learned about how to process images. We chose VGG11 because of memory constraints but a deeper alternative could just as easily be used. We also slightly modify VGG11 by removing its pooling layers because pooling layers would shrink the tensor's size. At the end of the VGG output we add two additional attention layers. We can think of our network up to this point as encoding and determining the latent variables needed to produce an inferred image. Testing with more layers or attention layers within VGG showed no improvements but resulted in larger memory consumption.

In the next step, we concatenate the output from VGG with the input to the VGG block (the second $\bigoplus$ symbol from left to right in Figure 1). At this point in the network we have an approximation of the latent space. By having the input images and these parameters we provide a similar input structure to interpolation methods like linear interpolation (LERP). The major difference here is that we can think of this model as similar to LERP but with many high order parameters that learn how to bias based on inputs and the simulation. The attention layers help determine the temporal aspects — this is why we have attention at the end of the input convolutions as well as at the end of VGG.

Finally we have a small decoding step that takes this latent approximation and produces an output image. This is done by using convolutional layers to reshape the tensor into one that can be easily converted to an image — three CNN layers were sufficient for our experiments to produce the desired outcome (the last CNN block from left to right in Figure 1). We note that more layers may provide better results as image sizes increase and when images have more channels.

4 Experimental Design

This section describes our experimental design. It is organized into the algorithms we consider (4.1), our corpus of data (4.2), types of experiments (4.3), the parameters for our machine learning infrastructure (4.4), the hardware used (4.5), the software used (4.6), and an overview of our techniques for evaluation (4.7).

4.1 Algorithms Considered

Our experiments consider two algorithms:

- Our ML-based algorithm, which is described in Section 3
- Linear interpolation ("LERP") between images

4.2 Corpus of Images

For these experiments, we generated images using the Ascent [21] in situ framework and the included CloverLeaf3D miniapp [26]. CloverLeaf3D is a hydrodynamic simulation, which we used to simulate an essemble of high pressure regions within a closed container. For each member of the ensemble, the imagery we generated was of a z-slice at the center of the box.

We refer to the ensemble members as **MEM1**, **MEM2**, and **MEM3**. We note that the number of cycles per ensemble member is variable, based on the physics occurring inside the closed container. Specifically, **MEM1** ran for 600 cycles, **MEM2** ran for 300 cycles, and **MEM3** ran for 800 cycles. In terms of additional differences:

- **MEM1** starts with a high pressure region located at the center of a box. As the simulation progresses, this high pressure expands, and the pressure reflects off of the hard walls that contain the fluid. For this simulation, images that are 50 cycles apart are visually distinct and cannot be trivially inferred from the adjacent images. Figure 2 shows images from **MEM1**.
- **MEM2** is similar to **MEM1**, but visually distinct. **MEM2** evolves from a point source and thus has a more circular shape to it. The pressure is higher and the initial high pressure region is offset from our slice. This results in a faster convergence and CloverLeaf3D exits earlier, causing this data set to have fewer cycles. Figure 3 shows images from **MEM2**.
- **MEM3** is dissimilar from **MEM1** (and from **MEM2**). **MEM3** has same high pressure region as **MEM1**, but adds to it additional high pressure regions. This causes the first few cycles of the simulation to be similar to **MEM1**, up until the interaction between the multiple high pressure regions causes interfering waves and a more complex wave pattern evolves. Figure 4 shows images from **MEM3**. Overall, the images from **MEM3** are drastically different than **MEM1** (and from **MEM2**).

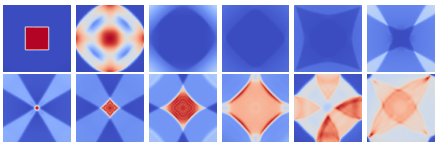


Fig. 2: Corpus of images for ensemble member **MEM1**, sampling every 50 cycles, from time-step 0 to time-step 550

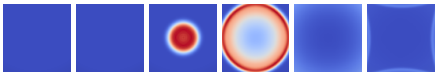


Fig. 3: Corpus of images for ensemble member **MEM2**, sampling every 50 cycles, from time-step 0 to time-step 250

Fig. 4: Corpus of images for ensemble member **MEM3**, sampling every 50 cycles, from time-step 0 to time-step 750

4.3 Experimental Campaigns

Our study consists of two campaigns:

- **Inferring Images on New Ensemble Members**: This campaign studies how our model learns on members of an ensemble simulation. We look at three ensembles in total, where we train on one and infer images from the other two. This campaign informs how our algorithm could be used by visualization practitioners.
- **Understanding Model's Data Dependence**: This campaign pursues a set of experiments to understand our model's dependence on data and help practitioners understand the limitations.

Each campaign involves many image comparisons, and we refer to each comparison as an experiment. The details of the experiments are described in the campaign descriptions (4.3.1 and 4.3.2). However, at a high level, the first campaign involves 847 experiments (comparisons) and the second campaign contains 480 experiments (comparisons), spread out over three sub-campaigns of 60, 180, and 240.

In total, we ran 1327 experiments (847 + 60 + 180 + 240)

4.3.1 Inferring Images On New Ensemble Members

A well-trained and effective machine learning model is typically noted by its ability to generalize. That is, a model's ability to perform accurately on unseen data. While we leave out testing and validation sets to determine the effective power of our model these data sets do not inform us how well this model works on novel simulations. We do not expect our model to trivially transfer to simulations that are drastically different, the same way we would not expect a model trained to identify cats to accurately identify airplanes. Instead, we compare our algorithm's effectiveness at transferring to CloverLeaf3D simulations with different input parameters. In short, we measure the effectiveness of our model on ensemble simulations.

In this campaign, we train on the entirety of **MEM1** data. We do this with a 2x repeat and 80% training size. We then evaluate how well this model can infer images for other ensemble members, specifically **MEM2** and **MEM3**. For each ensemble member, we perform "transfer learning," i.e., fine tune our model with a proportion of the images from that ensemble. These images are representative of the time slices that the simulation would save. We consider the efficacy of our algorithm as it is allowed to see more and more data. The proportion of data we consider is 0% (no data), 1%, 2%, 3%, 4%, 5%, and 10% (7 options overall). For example, we did an experiment where our model trained on **MEM1**, was allowed to fine tune with 1% of the data from **MEM2** and then inferred new images for unseen time slices at **MEM2**. In all, this means that we ran fourteen sub-campaigns, as the cross product of two ensembles (**MEM2** and **MEM3**) and seven proportions. Further, we attempted to reconstruct 50 different images from **MEM2** and 71 from **MEM3**, equating to 121×7 or 847 images overall.

For evaluation, we use LERP as our comparator. Our LERP uses the best images available, drawing from both the training set (e.g., the 1% of data used to fine tune) as well as two input images given to the model. For example, the case of 1% of data from **MEM2** corresponds to only three images, at cycle 0, 100, and 200. If the model was asked to infer the image at cycle 150 and was given inputs of cycle 75 and 175, then our LERP method would interpolate between the images at 100 and 175. This is because its closest available on the low side is from the training data, while its closest matches on the high side was one of the input images to the model. We feel this is a "best effort" approach with LERP, although it obviously does not make use of any data from **MEM1**. We believe that this method is also more representative of what would be used by scientists, since they are required to save out images, albeit sparsely, to use our algorithm. The difference is that our algorithm has learned from the previous dataset and is able to use the information gathered previously, having learned how these types of simulations evolve over time.

An important goal of the campaign is to investigate how much data is needed to fine tune our algorithm to our new ensemble. Our data proportions (0%, 1%, 2%, 3%, 4%, 5%, and 10%) range from a sparse temporal sampling (1%) to typical (3% to 4%) to a dense temporal sampling (10%). These values give us an idea of how data dependent our algorithm is and how well it adapts to new simulations.

Overall, this campaign demonstrates our algorithm's ability to generalize and perform on ensemble simulations. This demonstrates our algorithm's ability to be used in real world simulations and shows a direct use-case that simulation scientists can use today. It directly informs how sparse ensemble simulations can be and the trade-off in quality of results.

4.3.2 Understanding the Model's Data Dependence

This campaign was made up of multiple sub-campigns aimed at gathering an understanding of the model's data dependence. They experiment with using different data set sizes and different temporal spacing, to investigate how our model responds. This informs the limitations of our model, as well as directions for improving results.

Midpoint Temporal Upscaling For this sub-campaign we investigate how our model performs on a simple task. We select a fixed distance between the two input images, 100 cycles, and reproduce the midpoint. That is, if we select an image at cycle i our second input image is located at cycle $i+100$ and we produce the image at cycle $i+50$. This distance was chosen because it is a large range with distinct events occurring between these cycles, as can be seen in Figure 2. For evaluation, we supply the model with the input image at i, the input image at $i+100$ and the parametric scalars: i, $i+100$, and $i+50$. Overall, this sub-campaign evaluates whether our model is able to perform accurate reproductions of the ground truth over long temporal sequences, even when there are significant changes between the inputs.

Arbitrary Temporal Upscaling The next sub-campaign deviates from the midpoint model in two ways.

First, it considered arbitrary distances. For each test, we generated three cycles that were located at arbitrary temporal separations: c_{i1}, c_o, and c_{i2}, with $i1 < o < i2$. $i1$ and $i2$ were input images and o was the desired images. (We also required $i2 - i1 > 3$ to ensure that there was a source image between the two input images.)

The second deviate is that we allow for selection of how many times to repeat this process over the data, in this case 2. With a large enough n, as in this case, we reduce the likelihood that there is overlap between our testing and training data. If we allow for this process to be repeated $2x$ both our training and testing datasets increase proportionally and we maintain a low probability of overlap.

We supply the model with the two input images, $i1$ and $i2$, and the three parametric scalars: $i1$, o, and $i2$.

With this campaign we demonstrate the ability to reproduce both long and short temporal sequences as well as having varying reproduction distances. This shows that our model is highly flexible and able to learn a complex interpolation method through just a small selection of images.

4.4 Machine Learning Parameters

To read in this data we parse the directory where the images are sequentially denoted and assign an index associated with the ordinal data. These indices are then shuffled and split into training (80%), validation (10%), and test sets (10%). This ensures that the model is able to train on a representative sample set of the data and we leave enough data for validation and testing to ensure our results are statistically significant. For these experiments we use a learning rate of 5×10^{-5}, the AdamW optimizer with default parameters, and a batch size of 100. The batch size was chosen because this was the maximal value we could batch

within our memory constraints. We limit these experiments to a maximum of 10k epochs, and do not halt the run based on fidelity metrics. We do save randomly selected images from the validation set as the simulation progresses so that we can visually inspect the improvements of our inference as our model converges.
Because the color map represents one dimensional data, we convert the images to greyscale. This provides us with single channel data, and reduces the memory consumption of the input data by a third. Similarly, to reduce memory consumption we reduce the image sizes to 32x32, from 1024x1024. We note that we have investigated sizes up to 128x128, though the algorithm takes longer to perform as the batch size needs to be dramatically reduced because of our hardware constraints. We believe that the quality of our results are agnostic to the size of the image, and would require a similar number of epochs if GPU memory was scaled proportionally. No other preprocessing measures were applied.

4.5 Hardware

To perform our experiments we used a system with an Intel Xeon E5-1650 v3 @ 3.50GHz with a GeForce RTX 2080 Ti (11GB).

4.6 Software

For these experiments we used the PyTorch machine learning framework. Code and sample data are located at https://github.com/uocdux/DATUM.

4.7 Evaluation

For each experiment we compare our reproduction to the ground truth and the mean of the input images. We use PSNR to evaluate our results on the testing and validation sets.

$$PSNR = 20\log_{10}(MAX_I) - 10\log_{10}(MSE)$$

Where MAX_I is the maximal pixel value, in this case 255 (greyscale images), and MSE is the Mean Square Error. PSNR is a variant of the MSE metric and because of this, images that are visually distinct can have means that are close to the original. It should be noted that the higher the PSNR value the better the images are. If the images are identical then the MSE would be 0, this PSNR would be infinite.

To account for the downfalls of the metric, we visually inspect the images by plotting 10 random sample images from the test set. To visually inspect, on the same plot we also show: the ground truth, LERP, and the two input images. Through this we ensure that produced images are both visually appealing and have empirically similar.

5 Experimental Results

In this section we discuss the results of the experimental campaigns described in Section 4. We break this out into subsections discussing the results from each of our two campaigns — inferring images new ensemble members(5.1) and understanding the data dependence of our model(5.2).

5.1 Inferring Images on New Ensemble Members

This campaign demonstrates our algorithm's ability to transfer learn between ensemble simulations and investigates how much data we need to fine tune our algorithm for a new ensemble member. We look at our algorithm's ability to infer images when given access to 0%-10% of a new simulation. Specifically, we train on **MEM1** and transfer our machine learning algorithm to **MEM2** and **MEM3**. In each case we use 10% of the dataset from the new ensemble members for our testing set. Table 1 shows the percentage of training data that our algorithm and LERP had access to and the average of the associated PSNRs. Figure 5 shows sample outputs from the methods when shown 5% of the new ensemble member. Figure 6 shows the PSRN vs the number of training epochs for the same case, a mean value for the input images is depicted to show that DATUM is not just blending images.

From Table 1 we can see that our algorithm beats LERP by a fair amount on the **MEM2** case. In the **MEM3** case our algorithm wins when given access to a low, but non-zero, training amount. LERP quickly wins out against DATUM as more training data becomes available. Once given 3% of **MEM3**'s dataset DATUM's prediction has a PSNR that is relatively the same as LERP. We also can see that DATUM performs poorly when it does not have access to any information, the 0% case, from the new ensemble member. This shows that the ensemble members are not similar enough that our algorithm just memorizes images from the first member and spits them out in the new member. Our algorithm here only has access to two images form the new ensemble members but does not have any information that this is a different member. LERP has no prior information about what an ensemble should or should not look like, and thus the simpler method wins out with no priors.

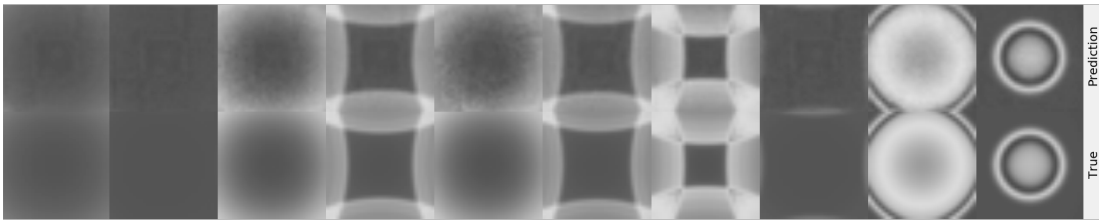
	MEM2		**MEM3**	
%	DATUM	LERP	DATUM	LERP
0	60.25	**64.89**	59.23	**77.37**
1	**82.30**	70.96	**90.48**	84.28
2	**85.11**	73.69	**90.32**	88.64
3	**84.01**	73.26	**90.77**	90.57
4	**82.31**	75.93	**87.76**	87.69
5	**86.34**	72.74	89.43	**89.45**
10	**86.52**	83.38	93.77	**100.74**

Table 1: PSNR for Transfer Learning on **MEM2** and **MEM3**. This table shows a comparison of DATUM vs LERP using the comparison described in Section 4.3.1. Bold numbers show which algorithm performed the best.

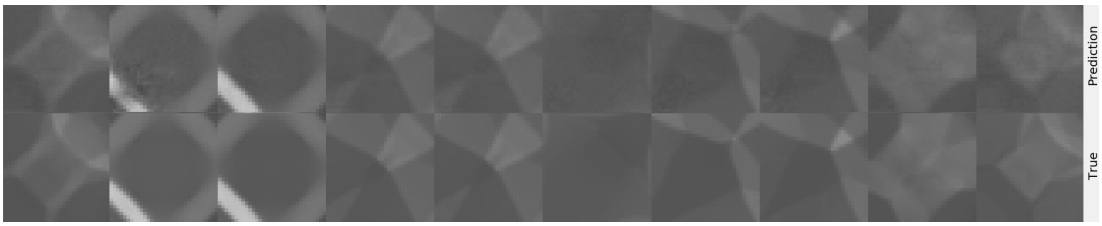
Figure 5 shows an example output from the testing set when given access to 5% of the new ensemble member. We show DATUM's prediction as well as the ground truth for visual comparison. The inference isn't perfect, but a visual inspection shows that these are reasonable approximations of the data. The main issue with our algorithm is that it has some noise in certain regions. When looking at the **MEM2** case we see regions, such as those in columns 8,9, and 10, are almost indistinguishable but there is a little noise. In columns 3 and 5 there is greater noise and our algorithm seems to have a harder time generating smooth images. Despite this, the overall shape of the prediction looks similar. When looking at the **MEM3** case we see similar success and losses. Particularly column 2 has a lot of noise. While most images do not have this noise – column 3 is almost identical and does not have the noise – DATUM's inability to infer on these images greatly affects the average PSNR of the set. This is likely due to our algorithm using a randomized distance between the input images and a high learning rate. The difference in columns 2 and columns 3 is that column 2 had used images at cycle 2 and 126, predicting cycle 75. Column 3 used cycles 52 and 82 to predict cycle 80. We note that LERP has a slight edge in this manner, because we allow it to find the closest input images while we do not account for this in our algorithm.

Figure 6 shows the PSRN at the 5% access as DATUM trains over new epochs. There are two interesting parts of these graphs. The first is that in **MEM3** it grows to a higher PSNR than the **MEM2** case and quicker. The second is that the **MEM3** case is more noisy than the similar case. A similar factor plays into the role of both these phenomena, the learning rate. We had held the learning rate constant at 5×10^{-5} for all experiments to reduce the number of changing parameters. A large learning rate causes a machine learning algorithm to converge to a solution faster, but if it is too large it may oversetp the local optima and have a noisy convergence. In this case it is clear that in **MEM3** a smaller learning rate would smooth out the results, but this would also require more training time. This is also why **MEM2** has a more gradual convergence to the optima. In **MEM3**, visual inspection shows us that while many images are correctly inferred there are a few images that are more difficult. Here the model is trying to fit these images but is oversteping the optima.

Additionally, from the PSNR results in the table and the graph we can conclude that while **MEM3** is more distant from the trained ensemble, it is a simpler simulation. We can conclude this because both DATUM and LERP are able to perform better on this dataset with less

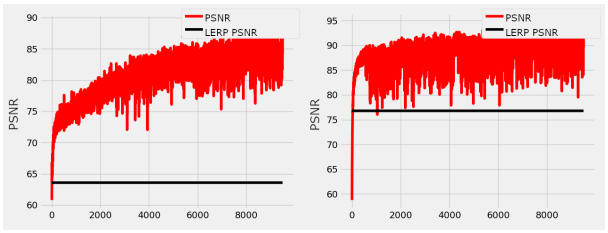


(a) **MEM2**



(b) **MEM3**

Fig. 5: Example outputs from Transfer learning on the (a) **MEM2** and (b) **MEM3** datasets when given access to 5% of the dataset. Prediction shows DATUM's inference and True shows the ground truth from the dataset.



(a) **MEM2** (b) **MEM3**

Fig. 6: DATUM's PSNR vs training epoch. (a) shows the results for **MEM2** and (b) depicts **MEM3**. A black line showing the average mean of the two input images is also shown.

information. Additionally, on the 0% case, DATURM performs worse on **MEM3**, compared to **MEM2**, while LERP performs significantly better. Lastly, we see that the average of the mean of the inputs is higher in **MEM3**, showing that the inputs are more similar. This last fact also contributes to why DATUM learns faster on **MEM3**.

5.2 Understanding Data Dependence

Our second campaign, as described in Section 4.3.2, focuses on understanding the data dependence of our model. We first look at the midpoint experiment in 5.2.1 to demonstrate our model's ability to capture long temporal sequences. Then at an arbitrary distance in 5.2.2 and focus on several experiments that help us understand the data dependence of our model. Within this campaign we only look at the data from **MEM1**.

5.2.1 Midpoint Experimental Results

As described in Section 4.3.2 we infer the midpoint image where the difference between the input images is 100 cycles.

For this campaign, we describe results using two figures:

- The first figure (Figure 7) plots PSNR vs the training epoch, which provides quantitative information about the quality of upscaling. PSNR graphs show the PSNR increasing as the model trains more. There is a solid black line representing the average PSNR achieved by LERPing methods.
- The second figure (Figure 8) plots the reconstructed images, which provides qualitative information about the quality of upscaling. These plots show a matrix of images. The columns of the matrix show the two input images, the ground truth, the LERPed image, and our model's inference, in that order. The rows of the matrix represent different time slices for the inputs and ground truth, respectively, as described in previously. One row represents a single time slice that our techniques On the x-axis we label the columns accordingly. On the y-axis we label the cycle number for the row of images – input_1, input_2, ground truth.

As can be seen in Figure 7 our model quickly surpasses the fidelity of the LERPed images.

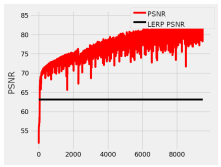


Fig. 7: PSNR as model trains on midpoint inference with step size 100 cycles

With Figure 8 we note that the ground truth and our model's inference are almost identical. We bring the reader's attention to row 9 (cycles 442,542,492) because it is the worst prediction in this subset. We note that there is difficulty in connecting the two pointed regions in the bottom right of the image. Additionally there are some slight color offsets in this region. The four corners pattern also is not as apparent in our inference. This at least demonstrates that our model is not memorizing the dataset and pasting back what it remembers.

We do note that in Figure 7 that the model is not fully converged and the fidelity is noisy between epochs. We note that the model is not fully converged and that there is a trade-off between training time and accuracy. While we do not expect our PSNR to approach infinite within a finite amount of time, we see in other experiments (below) that noise can be reduced and other factors play a role in increasing fidelity. The fidelity could be less noisy if a lower learning rate was used but this has a cost in time to convergence. We also decided to hold the learning rate, and all other ML parameters, constant to best compare results between experiments. In this way we only modify a single variable at a time.

5.2.2 Arbitrary Experimental Results

For our arbitrary distance experiment we performed more tests to demonstrate the effectiveness and limitations of our model. We first

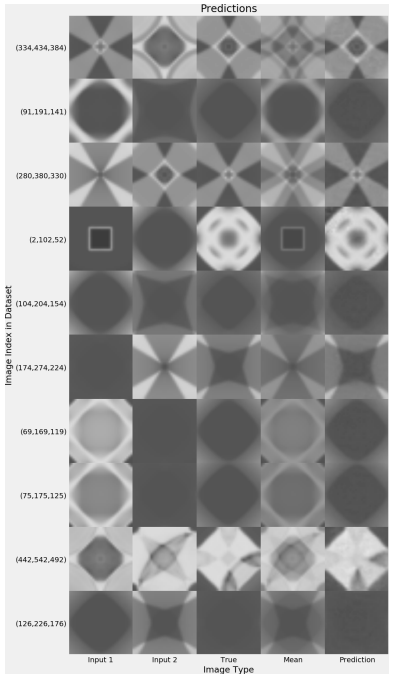
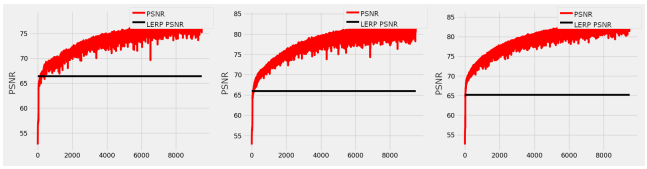


Fig. 8: 10 randomly selected samples from the test dataset



(a) 1x (b) 2x (c) 5x

Fig. 9: PSNR when we duplicate the data 1x, 2x, and 5x times

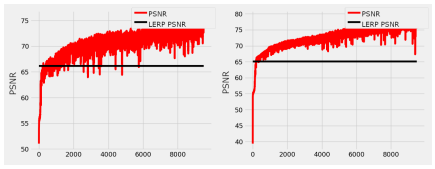
look at how many times we repeat the process described in Section 4.3.2 affects the data. Second, we investigate varying our training set size. Third, we perform a check by reconstructing the dataset from 7 inputs.

Similar to Section 5.2.1 we look at PSNR for quantitative results and sample images for qualitative results.

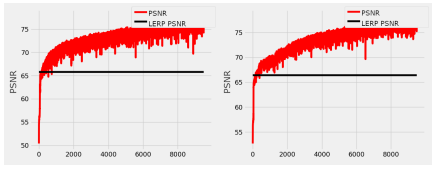
Repeat Selection vs Quality For our first set of experiments we examine how repeating our selection process multiple times affects the model's accuracy. That is, how many times we cycle over the data, pick a new m and c as described in Section 4.3.2. Essentially this is organizing the data in different ways. We may want to use multiple repetitions of data so that we decouple the model's dependence on images and increase increase its dependence on the parameters. By having one fixed parameter and image the model will learn different relationships.

From these figures we can see that 1x produces the lowest results, but performing the process twice we get substantially better results. The difference between 2x and 5x is not as meaningful. While the PSNR for the 1x results looks great, the inferred images do not look as visually pleasing. We show a sample of these images in Figure 12

These results performed better than the LERP but still have significant feature defects. It can be seen that these inferences are taking on the general shape of the ground truth image but when the ground truth image has complex features the inference becomes less refined and has a lower fidelity. We draw attention to the difference in rows 5 and 6. m is fairly small for both these rows, but row 6 has a harder time performing. We also draw attention to the last row. m is large here and the model has a harder time producing a more accurate image. We leave it to the reader to determine of our image is more accurate than



(a) 20% (b) 40%



(c) 60% (d) 80%

Fig. 10: PSNR of 20%, 40%, 60%, and 80% training sizes

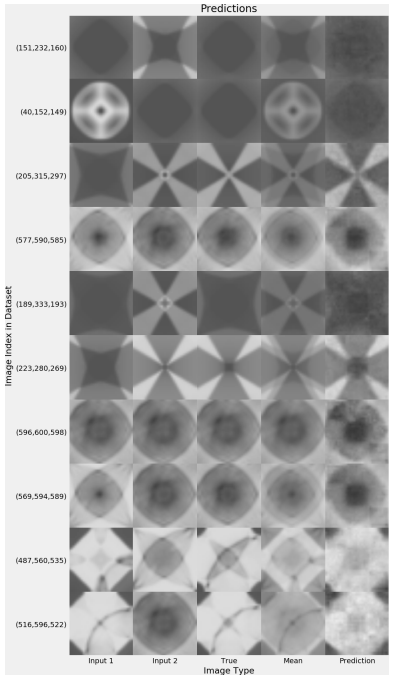


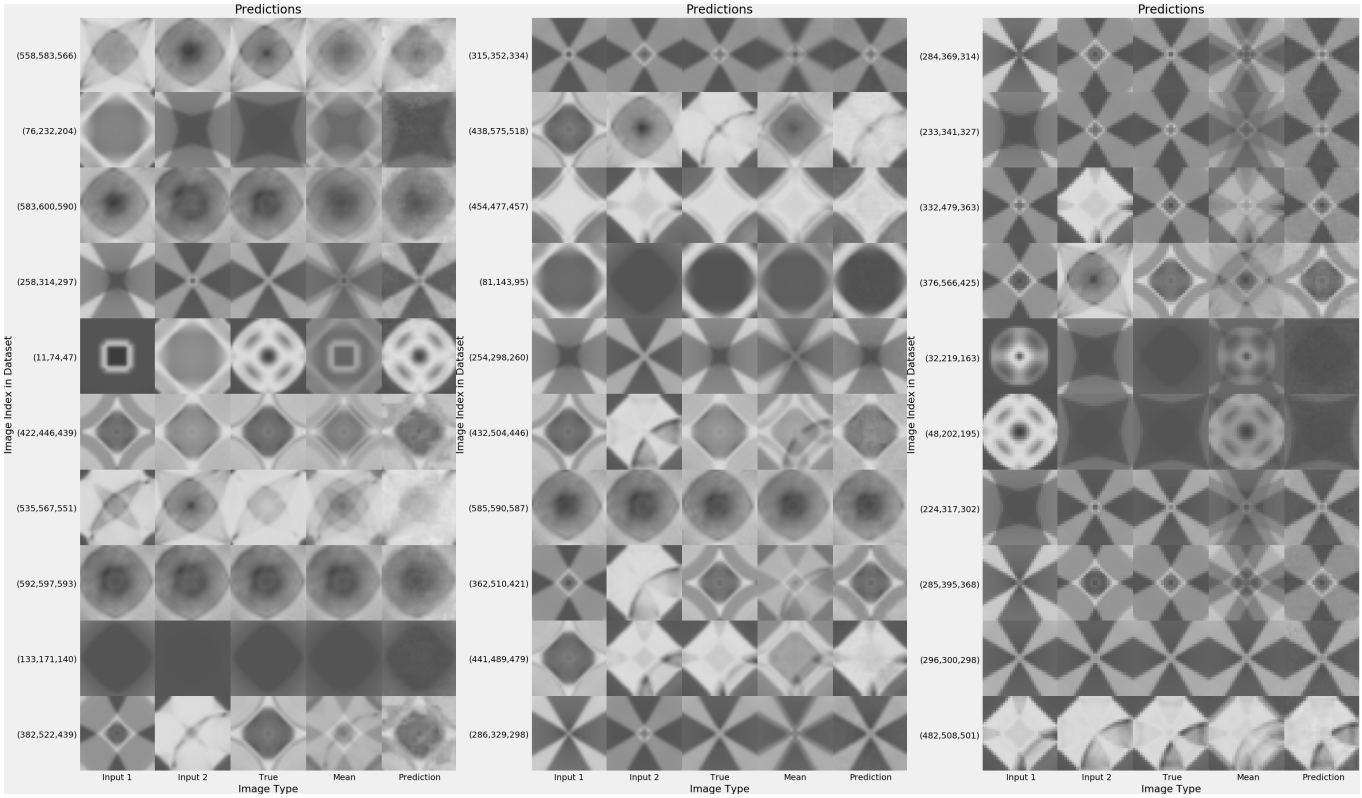
Fig. 11: Examples from test output when training on 20% of the data

the LERPed image.

In contrast to this, results from the 2x test set, Figure 12(b), looks substantially better. Here it is much more difficult to see the difference between columns 3 and 5. We note that there still are minor errors and draw the attention of the reader to rows 2 and 9. In row 2 we see that m is large (137), and the model performs quite well. The bow and arrow shape within the image is captured but more faded than the ground truth. Similarly, in row 9 the arch connecting the two spikes is less distinct and the central diamond is significantly faded. While the main part of these features is captured there is still some loss.

Lastly, we show the sample outputs from the 5x experiment, Figure 12(c). We note that the results here have slight improvements over the 2x experiment. While the PSNR of these experiments are similar it is more difficult to visually distinguish the difference between the ground truth and our inference.

Training Size vs Quality Our second experiment focuses on how dependent our model is to the amount of data it has access to. We investigate the model with access to 20%, 40%, 60%, and 80% of the data. We use the 1x method, as described above, so we can see the



(a) 1x (b) 2x (c) 5x

Fig. 12: Sample outputs from the 1x,2x,5x test dataset

largest difference between sets.

To better understand the qualitative results we also include samples from the output test sets. Figure 11 shows the result of the test set from only using 20% of the available data to train. As noted above, the training data is randomly selected from the available data. Figure 12 already shows samples from the 80% case.

Similar to the 1x experiment, the 20% experiment struggles to handle the more complex features, such as in the last row, but performs better than LERP in simple features like row 6. In cases where the input images are not distant, LERP actually performed better than this 20% experiment, as is best seen in row 7.

Arbitrary Remarks These experiments show us that there is a wide range in the inference power of our model. Section 5.2.2 shows us that the more data we have available the better our model performs. Section 5.2.2 shows us that there are tactics we can use to supplement our lack of data, by providing more training samples by rearranging the data we have. This allows users of our model to understand the limitations of our model but also gives them a method to improve accuracy when more data is unobtainable. As with any machine learning model, the more data one can feed in, the better it will train. In addition we note that the arbitrary results are able to perform better than the fixed distance results. This is because the model is getting access to a wider range of data. This shows that this training method is more beneficial to DATUM than by using a fixed temporal distance.

6 Conclusion and Future Work

In this work we presented DATUM, a method for temporal upscaling on ensemble simulations. We explored the use case of this algorithm and explored the how our machine learning model learns with different data sizes and arrangements. We demonstrate DATUM's ability to more accurately produce temporal upscaling from a sparsely saved ensemble simulation than LERPing. Overall, we feel that this work provides an important step towards a useful capability for visualization practitioners that is currently not possible. However, we recognize that there are still further avenues to improve upon this work and in this area of research. One fault is that with our limited access to hardware we were only able to train on small image sizes. While other machine learning work has shown that this can scale, access to more GPUs would allow us to demonstrate if this holds true for our work, and how far it scales. Additionally, there is an open ended question of if DATUM can be used in situ, fine-tuning on the new ensemble member in situ. This would greatly further the applicability of this work and provide a more useful tool to researchers. There are also other model structures that can be explored, such as other attention mechanisms that may improve results. Additionally we seek to expand DATUM's use cases such that instead of learning patterns in ensembles, it learns patterns in groups of physics simulations. For example, DATUM being used to learn the nature of wave patterns and being able to be applied to a wide set of wave solving simulations.

References

[1] M. Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari. Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. *CoRR*, abs/1802.06955, 2018.

[2] W. Bao, W. Lai, X. Zhang, Z. Gao, and M. Yang. Memc-net: Motion estimation and motion compensation driven neural network for video interpolation and enhancement. *CoRR*, abs/1810.08768, 2018.

[3] W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang. Depth-aware video frame interpolation. In *http://arxiv.org/abs/1904.00830*, 2019.

[4] M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *CoRR*, abs/1710.09545, 2017.

[5] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *CoRR*, abs/1405.3531, 2014.

[6] J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016.

[7] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu. Recent advances in efficient computation of deep convolutional neural networks. *CoRR*, abs/1802.00939, 2018.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[9] W. Grathwohl and A. Wilson. Disentangling space and time in video with hierarchical variational auto-encoders. *CoRR*, abs/1612.04440, 2016.

[10] J. Han, J. Tao, and C. Wang. Flownet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018. doi: 10.1109/TVCG.2018.2880207

[11] J. Han and C. Wang. Tsr-tvd: Temporal super-resolution for time-varying data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):205–215, 2020.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. G. Nashed, and T. Peterka. Insitunet: Deep image synthesis for parameter space exploration of ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, p. 1–1, 2019. doi: 10.1109/tvcg.2019.2934312

[15] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[16] M. Imre, J. Han, J. Dominski, M. Churchill, R. Kube, C.-S. Chang, T. Peterka, H. Guo, and C. Wang. Contournet: Salient local contour identification for blob detection in plasma fusion simulation data. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, D. Ushizima, S. Chai, S. Sueda, X. Lin, A. Lu, D. Thalmann, C. Wang, and P. Xu, eds., *Advances in Visual Computing*, pp. 289–301. Springer International Publishing, Cham, 2019.

[17] H. Jiang, D. Sun, V. Jampani, M. Yang, E. G. Learned-Miller, and J. Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. *CoRR*, abs/1712.00080, 2017.

[18] A. Z. Karen Simonyan. Very deep convolutional networks for large-scale image recognition. 2015.

[19] N. Kennamer, D. Kirkby, A. Ihler, and F. J. Sanchez-Lopez. ContextNet: Deep learning for star galaxy classification. In J. Dy and A. Krause, eds., *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 2582–2590. PMLR, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[21] M. Larsen, J. Ahrens, U. Ayachit, E. Brugger, H. Childs, B. Geveci, and C. Harrison. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pp. 42–46, 2017.

[22] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow. *CoRR*, abs/1702.02463, 2017.

[23] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[24] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[25] O. Makansi, E. Ilg, and T. Brox. End-to-end learning of video super-resolution with motion compensation. *CoRR*, abs/1707.00471, 2017.

[26] A. Mallinson, D. A. Beckingsale, W. Gaudin, J. Herdman, J. Levesque, and S. A. Jarvis. Cloverleaf: Preparing hydrodynamics codes for exascale. *The Cray User Group*, 2013, 2013.

[27] S. Niklaus and F. Liu. Context-aware synthesis for video frame interpolation. *CoRR*, abs/1803.10967, 2018.

[28] S. Niklaus, L. Mai, and F. Liu. Video frame interpolation via adaptive convolution. *CoRR*, abs/1703.07514, 2017.

[29] S. Niklaus, L. Mai, and F. Liu. Video frame interpolation via adaptive separable convolution. *CoRR*, abs/1708.01692, 2017.

[30] O. Oktay, J. Schlemper, L. L. Folgoc, M. C. H. Lee, M. P. Heinrich, K. Misawa, K. Mori, S. G. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert. Attention u-net: Learning where to look for the pancreas. *CoRR*, abs/1804.03999, 2018.

[31] V. Pătrăucean, A. Handa, and R. Cipolla. Spatio-temporal video autoencoder with differentiable memory. In *International Conference on Learning Representations (ICLR) Workshop*, 2016.

[32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[33] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. *CoRR*, abs/1611.00850, 2016.

[34] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In *Computer Vision and Pattern Recognition*, 2015.

[35] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[37] S. Weiss, M. Chu, N. Thuerey, and R. Westermann. Volumetric isosurface rendering with deep learning-based super-resolution. *CoRR*, abs/1906.06520, 2019.

[38] L. Weng. Attention? attention! *lilianweng.github.io/lil-log*, 2018.

[39] S. Wiewel, M. Becher, and N. Thuerey. Latent-space physics: Towards learning the temporal evolution of fluid flow. *CoRR*, abs/1802.10123, 2018.

[40] Y. Xie, E. Franz, M. Chu, and N. Thuerey. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

[41] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision (IJCV)*, 127(8):1106–1125, 2019.

[42] Z. Yin and J. Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. *CoRR*, abs/1803.02276, 2018.

[43] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks, 2018.

[44] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. Volume upscaling with convolutional neural networks. In *Proceedings of the Computer Graphics International Conference*, CGI '17. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3095140.3095178