Solid Freeform Fabrication 2023: Proceedings of the 34th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference
Reviewed Paper

Dreaming of Data: Examining Data Augmentation for Machine Learning in Additive Manufacturing

Glen Williams, [1] Martha Baldwin,[2] Timothy W. Simpson,[1]
Nicholas A. Meisel,[1] Christopher McComb[2]

[1]School of Engineering Design and Innovation,
The Pennsylvania State University, University Park, PA 16802
[2]Department of Mechanical Engineering,
Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

The data generated during additive manufacturing (AM) practice can be used to train machine learning (ML) tools to reduce defects, optimize mechanical properties, or increase efficiency. In addition to the size of the repository, emerging research shows that other characteristics of the data also impact suitability of the data for AM-ML application. What should be done in cases for which the data in too small, too homogeneous, or otherwise insufficient? Data augmentation techniques present a solution, offering automated methods for increasing the quality of data. However, many of these techniques were developed for machine vision tasks, and hence their suitability for AM data has not been verified. In this study, several data augmentation techniques are applied to synthetic design repositories to characterize if and to what degree they enhance their performance as ML training sets. We discuss the comparative advantage of these data augmentation techniques across several canonical AM-ML tasks.

1. Introduction

Studying design repositories through metamodels can provide useful insight into whether or not an engineering design dataset will be successful in training a deep learning construct. However, what should be done if a dataset is estimated to be insufficient? Instead of simply giving up, researchers presented with incomplete, small, or otherwise low-quality datasets often turn to data augmentation to enhance their dataset prior to training a machine learning model. In this study, several data augmentation techniques are applied to artificial design repositories to characterize if and to what degree they enhance their performance as machine learning training sets. The intersection of additive manufacturing (AM) and machine learning (ML) presents numerous, complementary opportunities for design engineers. Manufacturing products using AM allows designers to take advantage of opportunistic AM attributes, such as part consolidation to reduce the number of parts in an assembly by increasing the complexity of individual parts [1] and hierarchical complexity to make parts that are lighter while remaining strong [2]. These beneficial attributes of AM can help designers focus more on problem solving than on limitations of traditional, tooling-driven manufacturing.

In the same way that AM enables designers to manufacture increasingly complex and general shapes that might better solve a problem, ML could allow them to better predict how well those diverse solutions will perform in a multitude of ways. Some examples of researchers using ML to analyze AM parts include predicting AM build metrics [3], predicting thermal properties of AM designs during a build [4], classify parts as manufacturable or not [5], and prediction of product

geometric values to result in desired stress-strain performance [6]. However, both AM and ML are not perfect solutions, and many open questions regarding how to best utilize these technologies remain. ML approaches are generally most successful when developers have access to large training and validation datasets. Unfortunately, engineering data can be costly to create, obtain, and manage, resulting in datasets that are much smaller or more poorly labeled than those in some other fields. This problem is only compounded by the fact that AM technologies are relatively new and heuristics that might be useful for labeling datasets are the topic of contemporary research [7,8] and therefore in flux.

Data augmentation can be a powerful tool to enhance the performance of ML constructs while reducing their tendency to overfit [9,10]. It should be considered as an additional standard practice as part of the ML development process. Many ML studies focus on making incremental improvements to ML architectures while making fundamental assumptions that often hold a training and test dataset constant. In this type of study, an architecture-focused ML development process is often followed (see Figure 1). In this architecture-focused ML approach, the researcher has selected a) a dataset to use and hold constant and b) an architecture to test and present as an incremental improvement if successful. In the ML feature optimization loop shown in Figure 1, automated training algorithms, such as backpropagation, are used to iteratively alter ML construct weights and biases to improve the performance of the construct.

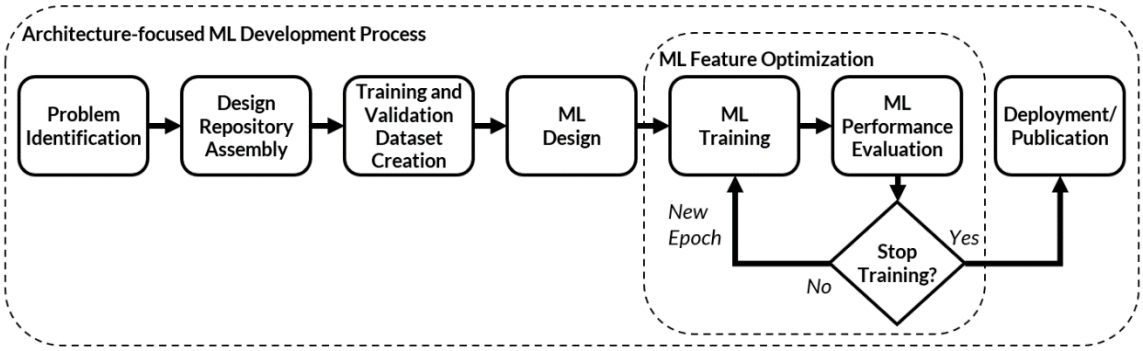

Figure 1: An architecture-focused ML development process iterates only on the ML construct architecture.

While this architecture-focused approach is time-tested for fields with highly standardized, agreed-upon benchmark datasets, it may not as applicable for other fields. For instance, the space at the intersection of Design for AM (DfAM) and ML, which we refer to together as DfAM-ML, suffers from a lack clearly and readily available datasets. For instance, surrogate modeling studies may require the generation of a new dataset for specific problems. The collective lack of understanding surrounding the impact of design repository attributes and data augmentation techniques on ML outcomes is a gap that could be further explored. To investigate this gap, this study presents a broader ML development process, the augmented dataset nested iterative ML development process (see Figure 2).

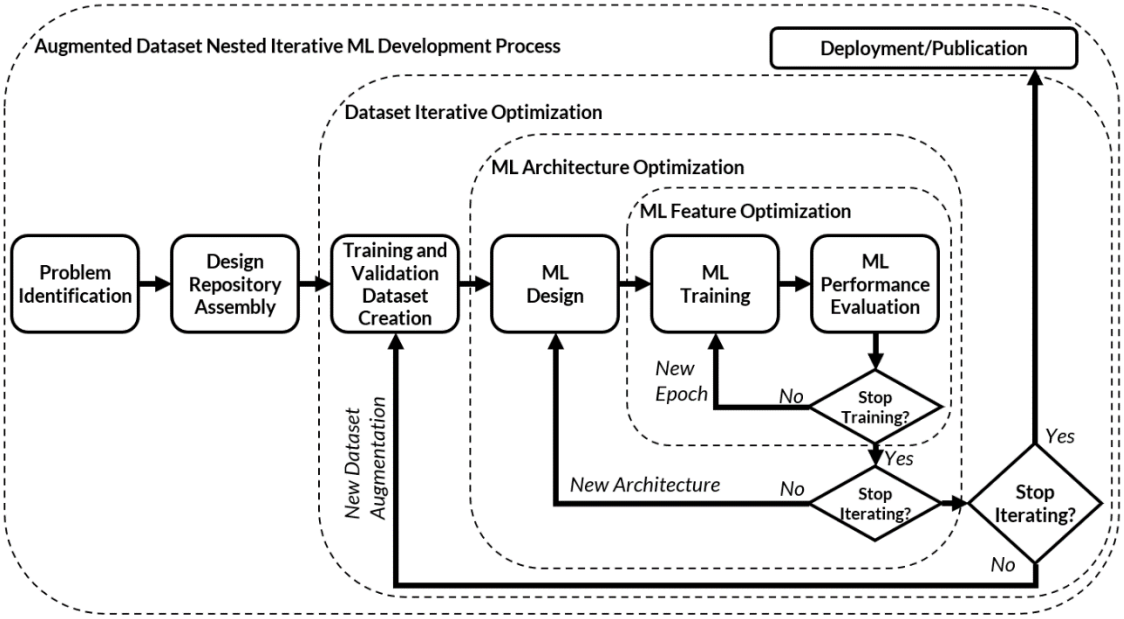

Figure 2: In the augmented dataset nested iterative ML development process, the model features, ML architecture, and training and evaluation datasets may all be varied to achieve greatest performance.

In the augmented dataset nested iterative ML development process, the same ML feature optimization is conducted, but additional loops to augment the dataset are conducted as well. Although these additional iterative loops may increase the research and development labor cost in the short term, they ideally will result in a greater likelihood of success of the entire ML development process. This study investigates specific practical techniques that could be applied to the new dataset augmentation loop in Figure 2.

In this work we investigate the details of how data augmentation can be useful in improving sub-optimal datasets in the context of DfAM-ML development. By studying a variety of data augmentation techniques across many, hierarchically complex design repositories with differing levels of constituent and aggregate complexity, and in the context of a variety of ML classification and regression problem types we aim to advance the heuristics around the DfAM-ML development process to increase the speed and efficiency with which useful ML constructs may be designed, trained, and tested. Specifically, we study the following research questions:

1. Which data augmentation techniques are most effective in improving the performance of ML training activities?
2. Do the data augmentation techniques affect trained ML construct performance similarly across different problem types?

The remainder of the work is organized as follows: First, we document the methodology of our candidate data augmentation techniques (section 2). We then present our results (section 3), discussion of the results (section 4), and conclusions regarding potential future work related to this study (section 5).

2. Methodology

This section documents the data augmentation techniques investigated (section 2.1), process for creating the artificial design repositories (ADRs) (section 2.2), overview of the CNN architecture (section 2.3), and training procedures (section 2.4).

2.1. Data Augmentation Technique Treatments

In this study, five data augmentation approaches were individually applied as treatments to Artificial Design Repositories (ADRs). An ADR is a synthetic dataset which contains constituent designs that were not necessarily designed for the primary purpose of manufacturing [3]. These treatments aim to increase the performance of a design repository as a machine learning training dataset. The treatments include a) no data augmentation (control), b) duplication, c) experience, d) single randomized orientation, e) multiple orientations, and f) gaussian noise (see Figure 3).

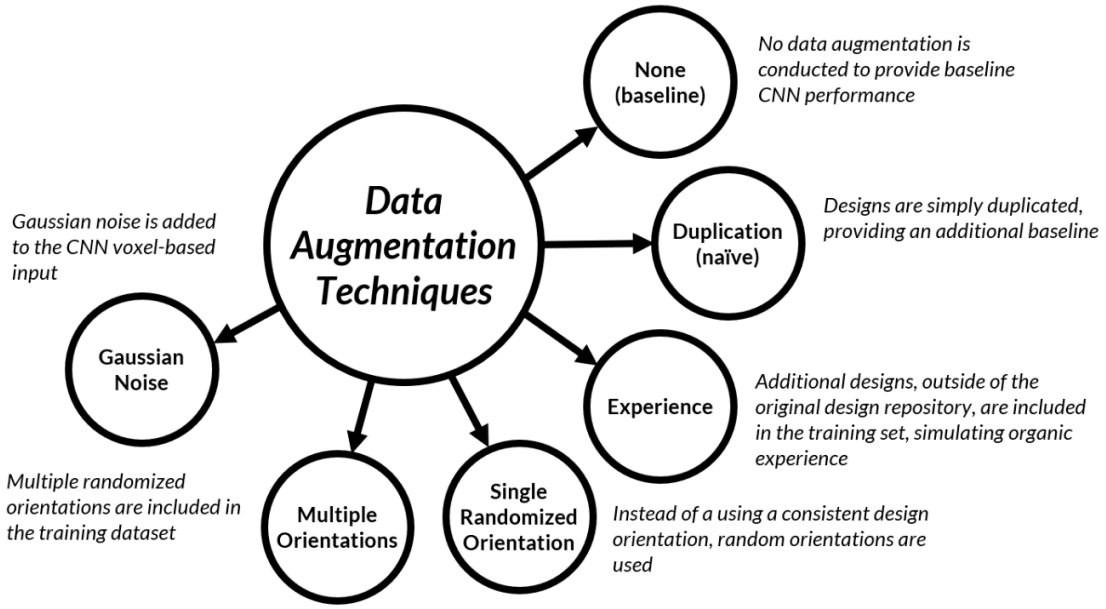

Figure 3: Overview of the data augmentation techniques investigated in this study.

Each treatment from the figure is explained in further detail as follows:

- In the *no data augmentation (control) treatment*, the ADRs were included as-is with no modifications to the training dataset. See section 2.2 to better understand the generation of the ADRs. Including a control treatment is important to establish a baseline for comparison of the magnitude of positive or negative effects each data augmentation treatment had. In these control repositories the designs were modeled in the same orientation, resulting in a consistent placement and orientation in the voxel-based input to the CNNs.
- In the *duplication data augmentation treatment*, the existing constituent designs in the training dataset were simply duplicated once. This is the simplest way to double the size of the dataset. Heuristically, this method may be susceptible to overfitting, but is evaluated to test that hypothesis empirically.
- In the experience data augmentation treatment, the dataset is doubled in size by including random designs from other artificial design repositories. This treatment simulates the action of augmenting a dataset the hard way: by waiting for more data to come in, gaining

more experience. In this analogy, random constituent designs added to the ADR are akin the new experience of designing more parts and storing their data in the repository.

- In the *single randomized orientation data augmentation treatment*, the dataset size was not doubled, but instead the models were rotated to a random orientation. This treatment aims to provide the CNN a greater variety of voxel-based features during the training process than the consistent orientation designs of the control treatment.
- In the *multiple orientations treatment*, the dataset size was doubled with the addition of another random orientation of the same model, instead of its replacement (as is the case in the single randomized orientation treatment). This resulted in a larger dataset that also included a greater variety of voxel-based features.
- In the *gaussian noise treatment*, normally distributed pseudo-random noise was applied to the constituent design voxel-based inputs. This gaussian noise has the effect of altering the volumetric occupancy binary voxels, which are usually only values of 1.0 or 0.0, to be decimal values between 1.0 and 0.0. The magnitude of the noise was centered around 0.05 with a standard deviation of 0.025. Each calculated random noise sample for a given voxel was added to that voxel's value if it previously had a value 0.0 and subtracted from its value if it previously had a value of 1.0.

2.2. Artificial Design Repositories

The ADRs used in this study were created using procedural modeling of geometries based on parametric design templates. Parametric design templates are advantageous for ADRs because they allow for rapid automated creation of constituent designs while achieving a desired level of geometric and topological variability in those designs. This variability is what makes the ADR suitably similar to real-world design repositories for the purposes of the current theoretically-oriented study. A multistep process was used to model the constituent designs and assemble them into ADRs suitable for the current work (see Figure 4). Specifically, the ADRs used in this study included 310 constituent parametrically designed parts. The parts were analyzed for classification and scalar regression labels relevant to DfAM-ML problems. A detailed example of the hierarchical modeling modelling process is shown in Figure 5.

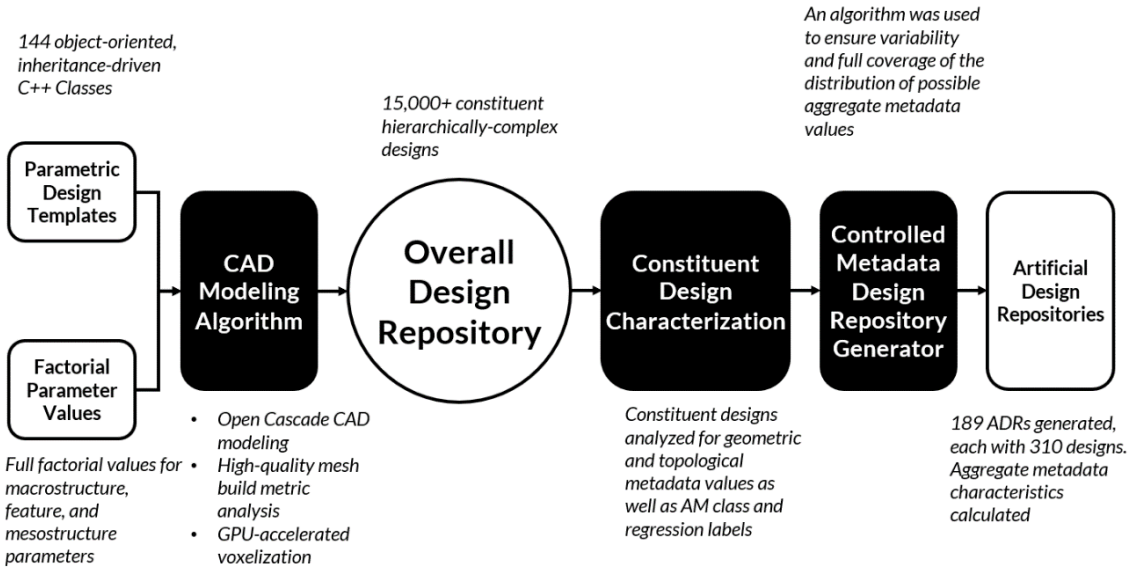

Figure 4: Flowchart of the overall Artificial Design Repository generation methodology

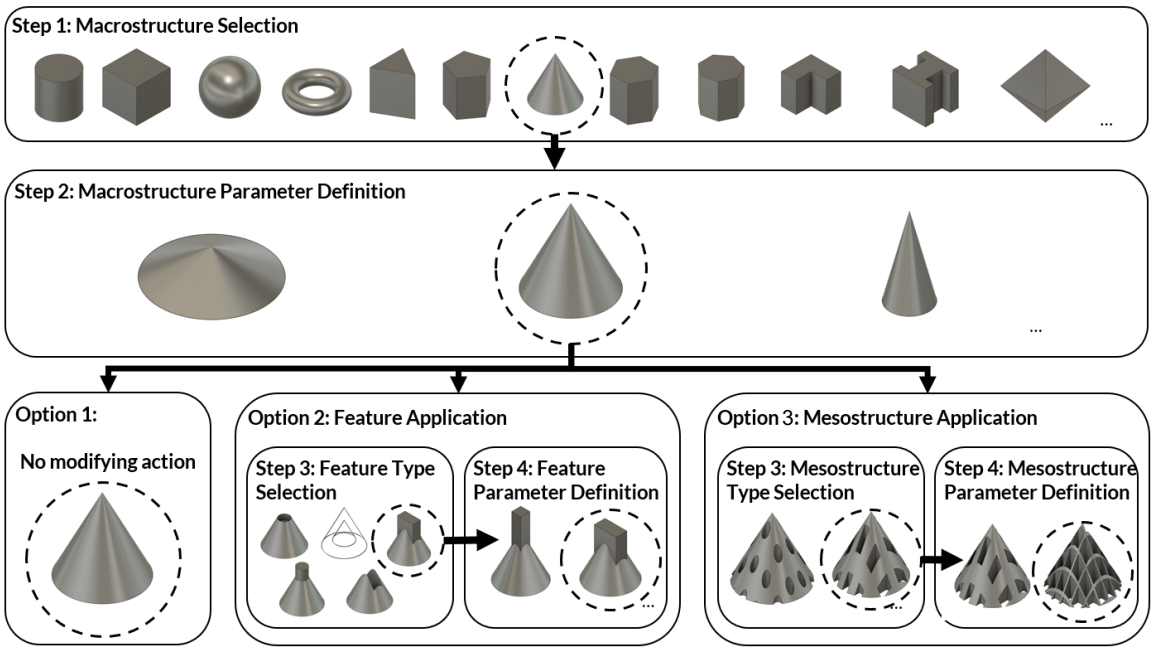

Figure 5: Detailed illustration of the hierarchically complex modeling process demonstrating high degrees of variability of geometric and topological complexity were achieved in the artificial design repository modeling process.

Creating a constituent design involves first selecting a macrostructure design template from 18 options and specifying arguments for its dimensional parameter values (see step 1 in Figure 5). These 18 unique macrostructure design templates are constructed to have between 1 (the sphere) and 7 (the I-beam) numeric dimensional parameters. They were selected to relate to common material stock forms, such as prisms, cylinders, and beams. After a design macrostructure was selected and defined, a design could optionally have a single additive or subtractive feature applied to it through constructive solid geometry Boolean CAD operations [11] (see option 2: step 3 in Figure 5). The 5 optional features included uniform shell hollowing, circular hole, circular post, rectangular hole, and rectangular post (see option 2: step 3 in Figure 5). Parameters of features included values such as “wall thickness” and “hole depth” (see option 2: step 4 in Figure 5). Features add to the number of parameters already available when procedurally generating models, ranging from 1 added parameter (the shell) to 6 added parameters (the rectangular post). Since a “design template” in the context of this study is a form produced from a set of quantitative parameter values, a combination of a macrostructure design template and a feature is an entirely new design template when compared to the macrostructure alone. For instance, applying constructive solid geometry Boolean cutouts or joins to a conical surface results in ellipsoidal and hyperbolic edges, which are not modeled in the original cone base design template, increasing the variety of graph topologies in the boundary representation models. See option 2: step 3 in Figure 5 for a diagram of the possible features.

Instead of a feature, a design could optionally have a mesostructure applied to it (see option 3 in Figure 5). Although mesostructures in DfAM applications are often 3D, 2D mesostructures were used in this study because of the limits imposed by the computational demand of the sheer number of designs modeled. These mesostructures included 2D patterns of circular cutouts and 2D pattern

rectangular prism cutouts. Applying mesostructures has a similar effect of increasing the geometric and topological complexities of designs as features, except to a higher degree. The mesostructure also add parameters to the design template, in this case related to size and 2D spacing distances of the mesostructure unit cells.

2.3 Convolutional Neural Networks

This study employed a similar convolutional neural network (CNN) to the prior metamodel study done by Williams et al., but with fewer channels in the convolutional layers and more neurons in the fully-connected layers (see Figure 6) [12]. The input was a 64 x 64 x 64 volumetric grid, a "voxel-based model". The voxels were binary numbers represented as 32-bit floating point values, with a 1.0 representing material presence and 0.0 representing material absence. These CNNs were trained with binary cross entropy loss function [13]. The hidden layers of the CNN were organized into an initial convolutional section and secondary fully-connected section. The layers included three alternating 3D convolutional-max-pool layer pairs, a flattening layer, and 3 fully-connected layers. The rectified linear activation function was used for all hidden layers. Each CNN model had between 33,614 trainable parameters.

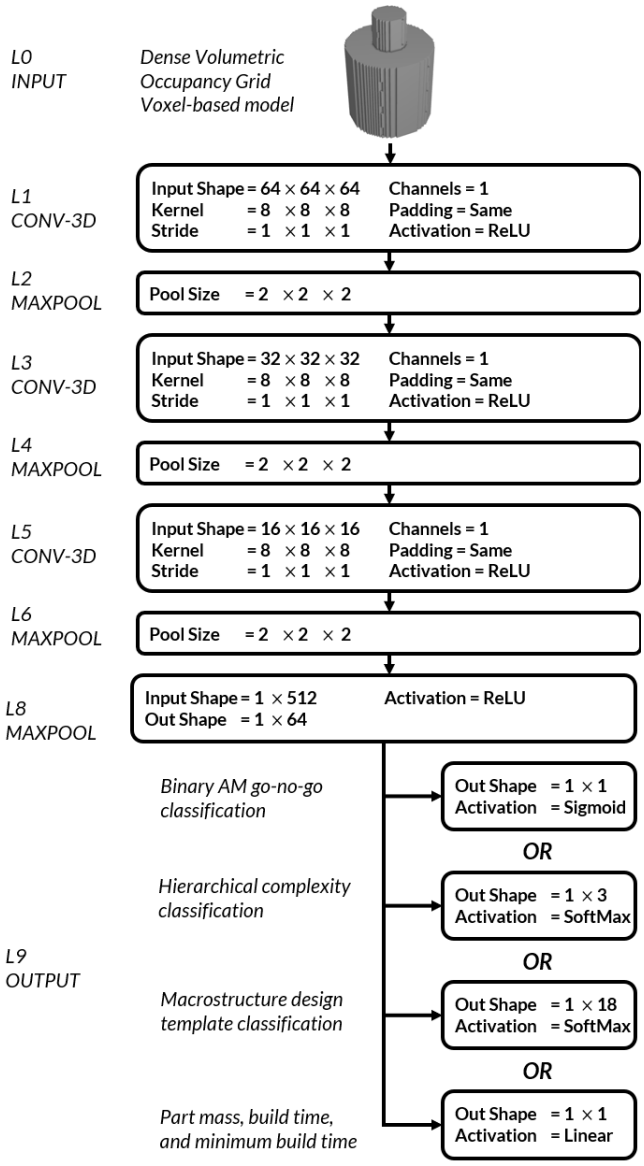

Figure 6: Diagram of the convolutional neural network architecture studied.

The networks differed in output dimensionality and loss function depending on the problem type studied. Binary classification networks (Go-no-go LPBF AM manufacturability classification) used a single binary output neuron (32-bit, 1.0 or 0.0 floating point) with a sigmoid activation. These CNNs were trained with binary cross entropy loss function [13]. Ternary classification CNNs (Hierarchical complexity classification) used 3 binary output neurons (32-bit, 1.0 or 0.0 floating point), which labels that were one-hot encoded [14]. Multi-class classification CNNs (design template classification) used 18 binary output neurons (32-bit, 1.0 or 0.0 floating point), which labels that were also one-hot encoded. Both ternary and multi-class classification CNNs were trained with categorical cross entropy loss function [15] and using a SoftMax activation function [15]. All scalar prediction CNNs (part mass, build time, minimum build time) had a single 32-bit floating point output neuron

with a linear activation function and were trained with a mean squared error loss function. In line with the learnings of Williams et al. relating to the standardization of orientation in a design repository, all designs were rotated to a random orientation before their input voxel-based model was prepared to maximize CNN performance [16].

2.4 Training

The training procedure was as follows: First, an ADR was loaded. Next, the ADR was split into randomized 75% training and 25% testing dataset. The CNN model was then trained using the training data. The history of each trained CNNs' loss, validation loss (either cross entropy or mean squared error), accuracy (either classification or mean squared error), and validation accuracy for each epoch were recorded. The trained CNN was then used to make predictions on the evaluation dataset, which was a constant dataset that included five times the number of constituent designs as an ADR, equal to 1,550 designs. This evaluation dataset consistent of a uniform random sampling of all the models and was intended to simulate analysis of future designs that would not be guaranteed to be similar to the training dataset. The CNNs were developed and trained using TensorFlow version 2 [17], Python version 3.10.4, and the "Adam" Optimizer [18]. All training was completed on an NVIDIA P6000 Graphics Processing Unit.

In this study, the CNN accuracy metric of interest was either classification accuracy or mean squared error when predicting values on an evaluation dataset containing a large number of random designs external to the training ADR. Each design repository in this study was used to train 36 CNNs (1 for each data augmentation treatment for each CNN problem type). This replicates the activity of evaluating the data augmentation techniques 189. In total, 6,804 CNNs were trained in this study. The performance of data augmentation treatments on an individual ADR was analyzed by calculating the difference between the control CNN performance on the evaluation dataset and the augmented CNN performance. These difference values were min-max normalized prior to analysis. The classification CNNs used the classification accuracy as the performance metric and the scalar prediction CNNs used the mean squared error as the performance metric. Mean squared error was subtracted from 1.0 after normalization so that all deltas of the performance metrics across different CNN problem types could be compared on a scale in which 1.0 is the best possible value.

3. Results

Applying the data augmentation treatments to design repositories resulted in variable increases, and sometimes decreases in trained CNN accuracy. The central tendency, spread, and distribution of data augmentation performance deltas (the difference between the augmented dataset performance and the control) were different for each data augmentation treatment in the context of each CNN problem type. Table 1 includes a summary of the mean and standard deviation of the normalized trained CNN accuracy delta for the different data augmentation treatments when applied to classification CNNs. Table 2 includes a summary of the mean and standard deviation of the normalized trained CNN accuracy delta for the scalar prediction (regression) CNN problem types. Figure 7 provides a set of plots which communicate this information visually. For all the plots, the vertical values have been min-max normalized based on the range of accuracy values observed for all CNNs in that problem type.

Table 1: The mean and standard deviation trained CNN performance delta for all data augmentation treatments applied to each classification CNN problem type. In this context, CNN accuracy refers to classification accuracy on the unseen evaluation dataset, min-max normalized to the range of accuracy values observed for that problem type.

	CNN Problem Type						
Data Augmentation Treatment	**Binary go-no-go AM mean CNN accuracy**	**Binary go-no-go AM std. CNN accuracy**	**Ternary Complexity Type mean CNN accuracy**	**Ternary Complexity Type std CNN accuracy**	**Template Classification mean CNN accuracy**	**Template Classification std CNN accuracy**	**Mean across all CNN problems**
Duplication	5.80E-02	1.12E-01	1.10E-02	1.20E-01	1.59E-01	1.71E-01	7.60E-02
Experience	1.62E-01	2.03E-01	-1.10E-02	3.87E-01	2.40E-01	1.72E-01	1.30E-01
Single Randomized Orientation	4.20E-02	1.37E-01	-1.60E-02	1.48E-01	1.15E-01	1.53E-01	4.70E-02
Multiple Orientations	7.20E-02	1.16E-01	1.00E-03	1.35E-01	1.94E-01	1.81E-01	8.90E-02
Gaussian Noise	3.80E-02	1.37E-01	5.00E-03	1.89E-01	1.62E-01	1.81E-01	6.80E-02
Mean across all treatments	7.40E-02	1.52E-01	-2.00E-03	2.19E-01	1.74E-01	1.77E-01	8.20E-02

Table 2: The mean and standard deviation trained CNN performance delta for all data augmentation treatments applied to each scalar prediction (regression) CNN problem type. In this context, CNN accuracy refers to 1.0 minus the mean squared error accuracy on the unseen evaluation dataset, min-max normalized to the range of accuracy values for that problem type.

	CNN Problem Type						
Data Augmentation Treatment	**Part Mass mean CNN accuracy**	**Part Mass CNN accuracy**	**Build Time mean CNN accuracy**	**Build Time std CNN accuracy**	**Minimum Build Time mean CNN accuracy**	**Minimum Build time std CNN accuracy**	**Mean across all CNN problems**
Duplication	2.65E-01	5.50E-01	4.74E-01	6.38E-01	4.84E-01	6.43E-01	4.08E-01
Experience	3.65E-01	3.53E-01	5.98E-01	4.37E-01	5.94E-01	4.28E-01	5.19E-01
Single Randomized Orientation	-1.56E-01	4.41E-01	2.77E-01	6.11E-01	2.89E-01	6.03E-01	1.37E-01
Multiple Orientations	2.60E-01	5.54E-01	4.68E-01	6.39E-01	4.79E-01	6.03E-01	8.90E-02
Gaussian Noise	3.28E-01	3.80E-01	5.82E-01	4.33E-01	5.58E-01	4.52E-01	4.89E-01
Mean across all treatments	2.12E-01	5.00E-01	4.80E-01	5.71E-01	4.81E-01	5.72E-01	3.28E-01

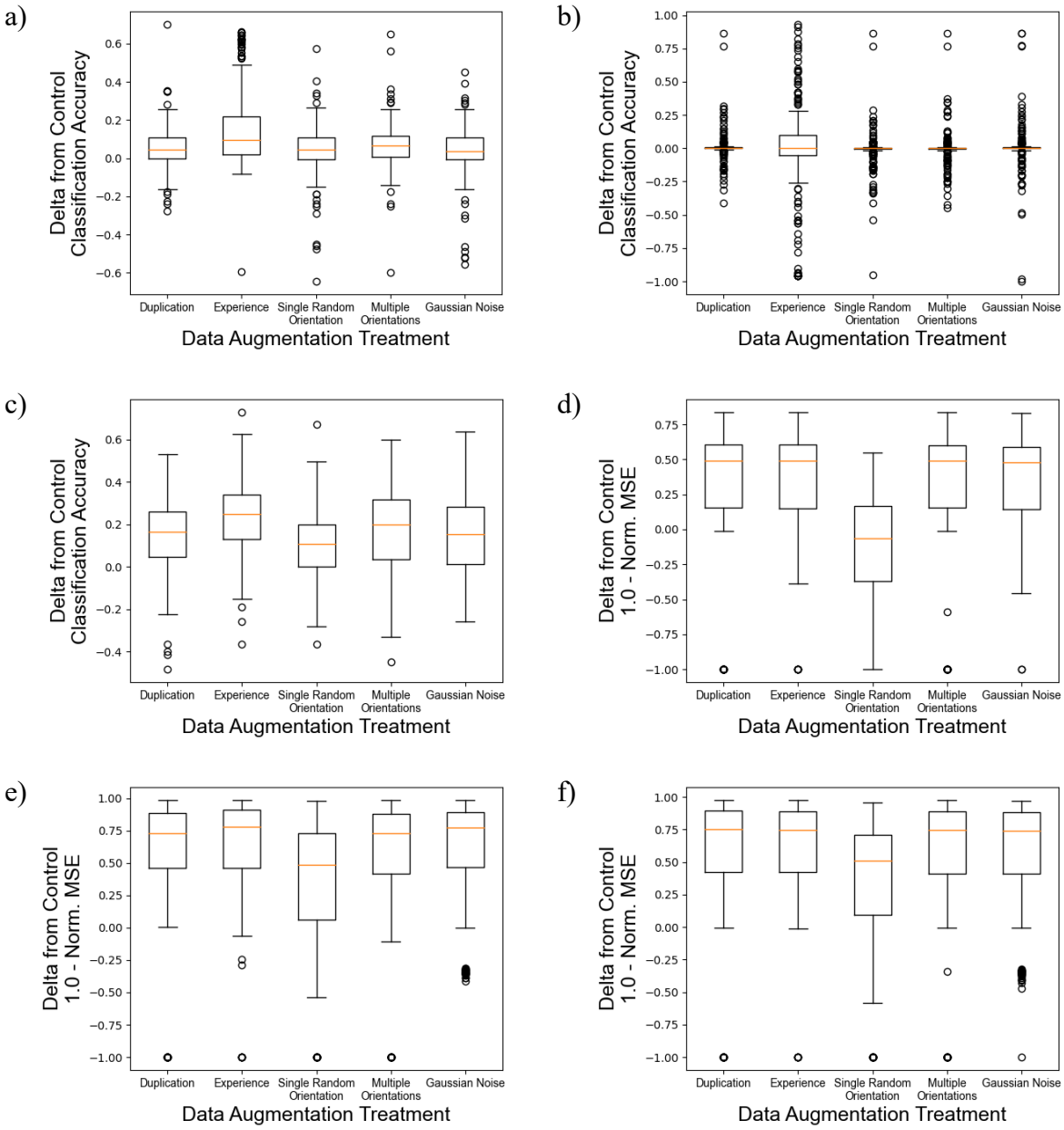

Figure 7: Box plots of the normalized change in trained CNN prediction accuracy on the evaluation dataset (unseen data) for all design repositories after application of the data augmentation treatments in the context of the (a) Binary Go-no-go AM CNN classification problem (b) complexity type CNN classification problem (c) macrostructure template type CNN classification problem (d) the part mass CNN scalar prediction (regression) problem (e) build time at a particular orientation CNN scalar prediction (regression) problem and (f) minimum build time CNN scalar prediction (regression) problem. In plots (a), (b), and (c), accuracy (the vertical axis) is the classification accuracy for the entire evaluation dataset. In plots (d), (e), and (f), accuracy (the vertical axis) is 1.0 minus the mean squared error for the entire evaluation dataset.

4. Discussion

Overall, the experience data augmentation treatment, which simulated the addition of new designs from outside the design repository over time, was clearly a high performing data augmentation technique. Experience exhibited the largest trained CNN accuracy deltas compared to the control on average for classification problems and the second largest for scalar prediction problems (see Table 1). This result confirms the intuition that new data is generally better than techniques to modify and augment data. However, the result that experience was not the best for both problem types is interesting. For classification, including more examples increased CNN performance the most. This makes intuitive sense, because in a classification dataset the frequency and balance of the output categories in the dataset is generally important [19]. However, in the scalar prediction problems, categorical labels are not relevant to the optimization of the CNN, and variability in the high-dimensional, voxel-based input through the addition of noise appears to have a particularly beneficial effect.

Data augmentation techniques that increased the size of the dataset generally performed better than those which did not. For instance, the single random orientation treatment, in which the constituent designs were rotated to a random angle, but the original design orientation was not included in the training dataset, was among the lowest performing for both classification and regression problems. The single random orientation treatment decreased the performance of the CNNs on average for some classification and regression problems. This result further supports the results from Williams et al., in which random rotations were found to be beneficial to CNN performance [16]. Although random rotations do frequently increase the performance of a neural network in this study, they do not appear to do so consistently across all problem types and when averaged over many different, hierarchically complex design repositories. Adding more design repositories, greater complexity and variability to those design repositories, and more CNN problem types further contextualizes the impact of random rotations on trained CNN performance, showing it is not as effective as some of the other data augmentation techniques considered.

The duplication data augmentation treatment outperformed the multiple orientation data augmentation treatment in both classification and regression problems, despite both treatments doubling the size of the input training dataset. This result is interesting because it suggests that the size of the input dataset, although important, is not the entire story in terms of the impact of data augmentation. This observation further supports the conclusions of the metamodel study by Williams et al., in which size was found to be the most important metadata attribute of a design repository by far [12]. The results of the current work show that the precise details of how the size of a design repository is increased can have complex effects on the overall performance of the trained CNN. In other words, a greater quantity of augmented designs appears to be often, but not always, better for CNN training outcome. Specifically, duplication might be more effective than including multiple orientations in this case because it simulates the effect of increasing the number of training epochs, essentially giving the CNN more time to train on the dataset. Including hyperparameter variation in the experimental methodology could provide evidence to causally link the differences observed between duplication and multiple orientations.

In terms of CNN problem type, classification CNNs that were trained to predict the macrostructure design template had their performances increased the most on average by the data augmentation

techniques. CNNs trained to predict minimum build time were the most improved by data augmentation among the scalar prediction CNN problem types. Conversely, classification CNNs that were trained to predict the ternary hierarchical complexity level of a design were the least improved overall when data augmentation techniques were applied to their training datasets. In the ternary hierarchical classification problem, the greatest frequency of average decreases in CNN performance due to a data augmentation treatment was observed. The experience and single randomized orientation treatments both decreased performance on average and the average classification delta was below zero. Part mass predicting CNNs were the scalar prediction CNNs that were least improved by the data augmentation techniques, experience an average decrease in performance from the single random orientation. These lower performing augmentation results shows that although the data augmentation techniques used in this study most often improved CNN performance, that is a possibility that they could have a negative effect. Keeping this result in mind when preparing a large dataset for a DfAM-ML development project could be crucial to best utilizing limited computational resources, as more data augmentation may actually cause a decrease in performances in some situations. The risk of hindering DfAM-ML performance with detrimental data augmentation, as evidenced by this result, motivates further study into the general topic of data augmentation across a variety of DfAM-ML problem types.

5. Conclusion

In this study, a characterization and analysis of five data augmentation techniques to improve DfAM-ML training performance of artificial design repositories (ADRs) was conducted. The magnitude of the change of performance of all design repositories after data augmentation in the context of the 6 different CNN problem types studied were analyzed. In terms of our first research question, which data augmentation techniques are most effective in improving the performance of ML training activities the experience data augmentation treatment, in which entirely new designs were obtained over time, was the best performing treatment for classification problems, while addition of gaussian noise to the voxel-based input was the best performing for scalar prediction problems. Experience was consistently an effective data augmentation treatment for both classification and regression problems. In terms of our second research question, do the data augmentation techniques affect trained ML construct performance similarly across different problem types, we found that the central tendency of performance improvement was similar in scale for different problem types, but that the distribution was highly dependent on the problem type.

Overall, we conclude that, based on our analysis, the best way to augment data is to prioritize obtaining entirely new data if possible. However, given that data augmentation is typically used in situations when obtaining more data is difficult or impossible, including many orientations of the same design in the training dataset outperforms other techniques tested. Given the large degree of variability in the performance deltas observed across different data augmentation techniques, future work should further investigate these questions. In particular, this study shows that some data augmentation techniques can cause an average decrease in trained ML performance, and thus additional study is needed to uncover additional heuristics that enable DfAM-ML developers to apply data augmentation strategically to avoid decreasing performance.

The results of this study have implications in the field of DfAM-ML development. All the data augmentation techniques demonstrated were effective at increasing machine learning training performance overall. This result means that industrial organizations can extract greater value from their design repository by processing it prior to using it as training data. However, the study also showed that data augmentation did not always increase ML performance, and sometimes reduced the training dataset quality. This result means that AM-ML developers should not blindly attempt data augmentation but should rather incorporate it as a part of a broader strategy. Using such a strategy means training multiple ML constructs using multiple data augmentation approaches and statistically comparing the accuracies. The degree to which data augmentation techniques are iterated should be budgeted for and balanced with the computational resources available, since both data augmentation and further ML training iteration both incur greater computational cost.

Future work should expand both the breadth and depth of DfAM-ML details. The breadth could be expanded by probing whether changing the ML architecture and input dimensionality affects the data augmentation performance enhancement. The depth could be expanded by investigating finer details of the ML training process, such as when overfitting occurs during long training activities with many epochs for different data augmentation techniques. Additionally, the interactions between multiple data augmentation techniques could be further studied to understand how they may be best used in combination.

Acknowledgements

This material is based upon work supported by the National Science Foundation (Grant No. CMMI-1825535). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Oh, Y., Zhou, C., and Behdad, S., 2018, "Part Decomposition and Assembly-Based (Re) Design for Additive Manufacturing: A Review," Addit Manuf, **22**(April), pp. 230–242.

[2] Hanks, B., Berthel, J., Frecker, M., and Simpson, T. W., 2020, "Mechanical Properties of Additively Manufactured Metal Lattice Structures: Data Review and Design Interface," Addit Manuf, **35**, p. 101301.

[3] Williams, G., Meisel, N. A., Simpson, T. W., and McComb, C., 2019, "Design Repository Effectiveness for 3D Convolutional Neural Networks: Application to Additive Manufacturing," Journal of Mechanical Design, Transactions of the ASME, **141**(11), pp. 1–12.

[4] Pierce, J., Williams, G., Simpson, T. W., Meisel, N. A., and McComb, C., 2021, "Stochastically-Trained Physics-Informed Neural Networks: Application to Thermal Analysis in Metal Laser Powder Bed Fusion," International Design Engineering Technical Conferences.

[5] Balu, A., Ghadai, S., Sarkar, S., and Krishnamurthy, A., 2020, "Orthogonal Distance Fields Representation for Machine-Learning Based Manufacturability Analysis," Volume 9: 40th Computers and Information in Engineering Conference (CIE), American Society of Mechanical Engineers.

[6] Jiang, J., Xiong, Y., Zhang, Z., and Rosen, D. W., 2022, "Machine Learning Integrated Design for Additive Manufacturing," J Intell Manuf, **33**(4), pp. 1073–1086.
[7] Booth, J. W., Alperovich, J., Reid, T. N., and Ramani, K., 2016, "The Design for Additive Manufacturing Worksheet," Volume 7: 28th International Conference on Design Theory and Methodology, American Society of Mechanical Engineers, p. V007T06A041.
[8] Bracken, J., Bentley, Z., Meye, J., Miller, E., Jablokow, K., Timothy, S., and Nicholas, M., 2021, "Investigating the Gap between Research and Practice in Additive Manufacturing."
[9] Wong, S. C., Gatt, A., Stamatescu, V., and McDonnell, M. D., 2016, "Understanding Data Augmentation for Classification: When to Warp?," International Conference on Digital Image Computing: Techniques and Applications.
[10] Shijie, J., Ping, W., Peiyi, J., and Siping, H., 2017, "Research on Data Augmentation for Image Classification Based on Convolution Neural Networks," Chinese Automation Congress, pp. 4165–4170.
[11] Bespalov, D., Ip, C. Y., Regli, W. C., and Shaffer, J., 2005, "Benchmarking CAD Search Techniques," ACM Symposium on Solid Modeling and Applications, SM, pp. 275–286.
[12] Williams, G., Meisel, N. A., Simpson, T. W., and McComb, C., 2020, "Deriving Metamodels to Relate Machine Learning Quality to Design Repository Characteristics in the Context of Additive Manufacturing," Proceedings of the ASME Design Engineering Technical Conference, **11A-2020**.
[13] Dr.A, .Usha Ruby, 2020, "Binary Cross Entropy with Deep Learning Technique for Image Classification," International Journal of Advanced Trends in Computer Science and Engineering, **9**(4), pp. 5393–5397.
[14] Armano, P. G., and Tamponi, D. E., 2012, "Assessing Encoding Techniques through Correlation-Based Metrics," Proceedings - 2012 11th International Conference on Machine Learning and Applications, ICMLA 2012, **1**, pp. 634–639.
[15] Ho, Y., and Wookey, S., 2020, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling," IEEE Access, **8**, pp. 4806–4813.
[16] Williams, G., Meisel, N. A., Simpson, T. W., and McComb, C., 2019, "Design Repository Effectiveness for 3D Convolutional Neural Networks: Application to Additive Manufacturing," Journal of Mechanical Design, Transactions of the ASME, **141**(11).
[17] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X., 2016, "TensorFlow: A System for Large-Scale Machine Learning," Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, pp. 265–283.
[18] Kingma, D. P., and Lei Ba, J., 2015, "Adam: A Method for Stochastic Optimization," ICLR.
[19] Kotsiantis, S. B., 2007, "Supervised Machine Learning : A Review of Classification Techniques," Informatica, **31**, pp. 249–268.