Learning to Design From Humans: Imitating Human Designers Through Deep Learning

Humans as designers have quite versatile problem-solving strategies. Computer agents on the other hand can access large-scale computational resources to solve certain design problems. Hence, if agents can learn from human behavior, a synergetic human-agent problem-solving team can be created. This paper presents an approach to extract human design strategies and implicit rules purely from historical human data, and use that for design generation. A two-step framework that learns to imitate human design strategies from observation is proposed and implemented. This framework makes use of deep learning constructs to learn to generate designs without any explicit information about objective and performance metrics. The framework is designed to interact with the problem through a visual interface as humans did when solving the problem. It is trained to imitate a set of human designers by observing their design state sequences without inducing problemspecific modeling bias or extra information about the problem. Furthermore, an end-to-end agent is developed that uses this deep learning framework as its core in conjunction with image processing to map pixel-to-design moves as a mechanism to generate designs. Finally, the designs generated by a computational team of these agents are then compared with actual human data for teams solving a truss design problem. Results demonstrate that these agents are able to create feasible and efficient truss designs without guidance, showing that this methodology allows agents to learn effective design strategies.  

[DOI: 10.1115/1.4044256]

Keywords: computer-aided design, design automation, design teams

1 Introduction

Advancements in machine learning and computational modeling have brought us closer to the goal of developing intelligent agents that can solve a variety of problems. Machine intelligence has surpassed human levels in several problem-solving milestones [1–5]. One common feature among all these problems is that they are well-defined with a set of rules and actions that are universally followed. Humans are still seen to perform better than machines for tasks which require skills like strategic reasoning, abstract decision-making, creativity, and explainability. All of these skills are characteristic of the field of design. In design, the sheer complexity of most problems and the loosely bound definitions and requirements necessitate a combination of the abovementioned skills to generate solutions. This motivates the need to combine the diverse qualities of both humans and machines to facilitate solving challenges in design. This work focuses on developing methods to learn essential design strategies from human data and use that to help improve the performance of computational design agents.

Design is an iterative process; in order to create something, humans interact with an environment by making sequential decisions. Expert designers apply efficient search strategies to navigate massive design spaces [6]. The ability to navigate maze-like design problem spaces [7,8] by making relevant decisions is of great importance and is a crucial part of learning to emulate human design behavior. Based on this understanding, Fig. 1 schematically represents a human solving a design problem, in this case a truss design problem as used in the example study in this paper.

The problem is decomposed into two steps: perception and problem solving. The designer perceives the current design state and takes an action to modify it based on previous knowledge and understanding of the problem. This new design is then fed back into the current design state and the process is repeated. Designer behavior guides the decisions regarding what actions to take and is shown as a black box, the input, and output of which are observable. However, what happens inside is hidden and not entirely known.

The motivation for this research is primarily based on understanding what goes on within that black box and uncovering the implicit rules and strategies. We present a data-driven approach that learns human design and problem-solving strategies and also learns to generate new designs only from observing the visual states of an evolving design, without explicitly being provided with any information about the performance metrics or meaning behind design modifications. Since the presented deep learning framework does not encode any specific information about allowable actions and instead represents the whole process using.

---

Contributed by the Design Theory and Methodology Committee of ASME for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received February 23, 2019; final manuscript received July 5, 2019; published online July 17, 2019. Assoc. Editor: Conrad Tucker.
images, this research has the promise to provide a domain-agnostic method of representing a generative design process. Every design process has a different set of actions or operations, preventing a common algorithm or framework to be used across problems. Our domain-agnostic method uses an image-based common representation to define and track the evolution of design problems.

This work illustrates how sequential images from historical designer data can be used to emulate designer. Specifically, an independent agent algorithm is developed that utilizes methods from unsupervised representation learning, imitation learning, and image processing to model the entire design process as shown in Fig. 1. The agent uses a convolutional autoencoder to map the design states to low-dimensional embeddings, then it uses a neural network-based transition function to predict a design embedding based on the current state and then maps the embedding back to the original image dimensions. This process produces an image of how the next design should look like. Finally, a control algorithm driven by image processing constructs outputs the parameters to execute the operation. This agent is trained offline, does not require carefully labeled state-action data and is agnostic to objectives and other parameters. Even though the current framework utilizes images (a two-dimensional array), the framework can be extended to other problems where the raw state is representable using an N-dimensional array. The contributions made by this research can be listed as follows:

(a) A generalized methodology is presented that can be used across various design problems to learn implicit design strategies organically from data, using a pixel-based representation.

(b) An agent framework is developed that generates and follows design suggestions and automates the design process in an end-to-end manner.

(c) The agent maintains a process similar to a human designer and hence provides a platform to use an agent in computational teams or along with humans in a hybrid agent-human team.

For the entirety of the paper, the term framework is used to refer to the combined deep learning model and the term agent is used to refer to the end-to-end algorithm that is the deep learning framework and the automated inference algorithm both combined. The paper is organized as follows: Sec. 1 contains a review of the relevant studies and builds up a foundation for the different methods used in the framework. Section 2 begins by explaining the setup of the design study and the nature of the datasets used, then it explains the network architectures, training strategies, and some intermediate results with the (image-based, or imaginal) deep learning part. Section 3 explains the experimental setup that was required to evaluate the performance of the agents and their computational teams. The results and discussions from the experiments are found in Sec. 4. The final conclusions and future work are mentioned at the end in Sec. 5.

2 Background

This section is divided into four subsections, each representing a different area of research this paper relates to. A review is conducted to discuss ideas and show how this work is related to the existing research in each of these fields and in what ways it is new and different from them.

2.1 Design as a Sequential Decision-Making Process. The idea of problem solving (or design) as a sequential decision-making process is bolstered by similar representations used in robotics [9], game AI research [10], behavioral economics [11], decision-making [12], and also design research [13–15]. Here, the basic idea is that an agent is placed in a design environment and the goal of the agent is to interact and create a design maximizing the given performance metrics. Formalizing design as a problem-solving process opens up possibilities of applying logic, inference, and different mathematical models to often abstractly defined design processes. Modern-day design processes work in a computer-in-the-loop setting, where designers leverage computational tools along with their learned knowledge to make informed decisions [16]. Data collected from these processes can be utilized to extract key features and insights about them, while some of the previous studies have found that certain probabilistic models can be used to represent design strategies and describe skilled behavior [14,15], generate designs with similar trends to humans [17], and finally generalize them to transfer across certain problems [18]. Modeling design processes in an environment is challenging since limiting the designer to a small set of operations can limit designer skills and these operations further have multiple parameters associated with them. Also, these variables and actions are specific to the design problem that is being modeled. This makes it difficult to develop an agent or a framework that can be used across the different domain of problems since a new learning architecture is required every time. This problem is discussed and solved in Secs. 2.2–2.4.

2.2 Pixel-Based Representation of Design Processes. Humans are shown to think and solve problems visually [19], creating mental imagery [20], and incrementally making changes to it until reaching a solution [21]. Humans are able to abstract and store information as images and then recall them when solving a problem; this process is collectively called imaginal thinking. Some previous works have also shown its utility in helping humans solve design problems using an interface [22,23]. This motivates the use of an image (pixel-based) common representation for defining a design problem. It is especially beneficial for design due to the great diversity of design variables in different problems leading to algorithms to be problem specific. An empty image encompasses the complete design space since any design can be represented within that image (if the problem is 2D), hence a design process can be represented as a designer changing the pixels of an image to achieve the best configuration of pixels. This form of representation can be extended to any problem that shows the current problem state as a set of pixels (or even voxels for 3D data). This method leads to a very high-dimensional problem that can be very difficult to manipulate directly (e.g., for design). The research presented in the subsequent sections utilize image-based representations and formulates a learning problem based on it.

2.3 Using Deep Learning for Design Feature Extraction—Dimensionality Reduction. Dimensionality reduction of design spaces is a critical problem and numerous techniques have been discussed in the literature. Such reduction is done in order to use computational techniques efficiently; however, there exists a tradeoff between geometric variability (possibility of creating novel designs) and dimensionality (complexity) that must be addressed [24]. Techniques like Karhunen–Loève expansion [10,11] and principal component analysis (PCA) [12] are linear methods that determine a set of dimensions that can maximally explain the variance in the data. Higher accuracy nonlinear methods have also been utilized by researchers like kernel PCA and multidimensional scaling [25,26]; however, these methods are non-invertible, i.e., one can go from high-dimensional space to a low-dimensional embedding, however, the embedding cannot be converted to the original design. This is an essential requirement of our framework. Recent advances in deep learning have led to the development of methods for highly nonlinear dimension reduction that are invertible and accurate. Autoencoders are one such family of methods which can determine lower dimensional embeddings [25,27–30] and is an efficient data-driven way of learning representations [31]. Autoencoders are a type of neural network that is trained to reconstruct an output that is identical to the input. Often, a density constraint limits the dimensions of the embedding and forces the autoencoder to generate a lower-dimensional representation. Autoencoders have proven to perform much better than other
available methods, especially for design tasks [25,32]. A variant of these autoencoders, a deep convolutional autoencoder, combines a convolutional architecture [33] with the encoder–decoder structure of an autoencoder [30]. It has the ability to extract high-level semantic information from image data and encode it to a lower dimension embedding. It is used in this paper as a part of a larger framework to encode design information to a lower dimensional embedding and back to an image which is human readable. Adding further constraints on the distribution of the embedding elements leads to a variational autoencoder architecture [34], which has certain advantages in design generation tasks. However, that requires a higher amount of data hence it may be explored in future works. Other recent works in design have also used convolution deep learning architectures to solve different tasks with both 2D and 3D data with interesting results [35–37].

2.4 Mimicry—Learning to Imitate From Raw Pixels. Understanding what happens inside the human behavior “black box” has been approached differently in different fields of research. Historically, researchers from psychology consider learning human behavior as learning a function that generates the same output given a specific input. These methods were aimed at learning rule-based relationships [38,39], or learning similarities [40,41] in problem-solving behavior. Studies in cognitive science have put forth several core ingredients of human intelligence, including intuitive physics [42–45], problem decomposition skills [46,47], ability in learning-to-learn [48], and others [49]. More recently, learning from imitation or imitation learning has been actively utilized to solve various problems since it does not require significant state-space exploration, making it a faster approach in practice. Researchers have taught robots to imitate and learn how to drive [50], reach [51], manipulate [52], play Atari games [53], and also fly and control a helicopter [54]. However, all these require properly labeled state-action pairs, and such datasets are very few in number and very problem specific. Recent work by Liu et al. [55] proposes a framework that learns only by observing state-space trajectories. This allows a network to learn in third person (or use historical data) which is closer to how humans learn by just observing without explicit action information. The research presented builds off some of the ideas from that work and adapts it to work with design data. Some other works develop on the two-step problem representation idea by having an encoded representation of the environment and a control network that can navigate it [56]; however, they do not utilize any human knowledge and use active learning methods like deep reinforcement learning to solve the problem, which is different from our approach.

Significant research has been done in deriving qualitative explanations for human design strategies [8,57–60] and using them to further design optimization algorithms [61,62]. Also, methods have been developed to extract and verify design knowledge from design data, like inter part dependencies from design databases [63], learning a machine learning-based design method recommender [64], and also developing more efficient optimization algorithms [65] from crowdsourced design solutions. These previous efforts, however, do not address the problem of making a generalized parameter-independent algorithm and solving a sequential decision-making design problem which is an essential factor for our framework in order to make it similar to how humans solve it. In conclusion, prior research relates to the methods that we have used; however, our work integrates concepts from different fields and proposes a unique two-step deep learning framework that decomposes the solving process of a design similar to a human and, hence, makes it compatible for future hybrid learning possibilities.


In this section, the methods involved in this research are explained. The first subsection discusses the original design study and the source of the dataset. The second subsection is a formulation of the problem. The third subsection explains the model architecture and the training strategies used. The final subsection illustrates how the deep learning framework is integrated with an inference algorithm to develop an agent.

3.1 Design Study and Database. The nature of the developed framework is such that it can utilize historical data for training and is generalizable to any problem with a pixel-based state representation. For the purpose of the work addressed in this paper, the database used for learning comes from a truss design study (conducted by McComb et al. [66]). The study involved senior undergraduate students in mechanical engineering who were tasked to design a truss structure given a certain set of objectives to fulfill. The students were randomly divided into 16 teams (three members each). An interface was designed for the study which allowed students to apply one of the nine operations shown in Fig. 2 and also interact with the rest of the team members to complete the truss structure. Two metrics were provided to participants for feedback on the quality of these trusses: a factor of safety (FOS) and total mass. The former is a metric to represent how strong and resistant to failure the design is, where a higher value represents higher strength. A value above 1 illustrates sufficient strength and deems the design feasible. Mass shows the amount of material that has been used in the structure. These two metrics together are used to evaluate the strength to weight ratio (SWR), i.e., the ratio of the FOS and mass. This represents how optimized the structure is and is also used to evaluate the quality of a structure; a high FOS with low mass will lead to a high SWR showing that high strength was achieved with minimal mass. During the study, the design interface was also used to capture data whenever a subject made any change in the design. The study was conducted in a controlled environment where different constraints were provided to the subjects in gradual time steps.

The resulting database contains the design state information for all the subjects under different design scenarios. Previous work
used this sequential design data to develop a method of representing design strategies and contrast high and low performing designers [14,15,17]. For the purpose of our research, we are using only the first portion of the study which is unconstrained. The first image in Fig. 2 shows the initial state of the design problem, with the pink arrows representing the load-bearing nodes while the other nodes are the support nodes. The database contains only the design state information with nodes and members, excluding any loading arrows or water. The subjects were tasked to create feasible designs (>1 FOS) with a minimal mass of the truss structure. This consists of a total of 12,830 design states that appear in design sequences generated by humans. These states include parametric specifications of nodes and members that exist in the design structure. As the designs progress, the number of nodes and members vary greatly in the dataset, changing the number of variables to completely define a design. In the current work, the design state is represented by an image with a fixed number of pixels instead of the variable number of parametric values. This representation leads to a more visually interpretable representation which is what the subjects in the study actually observed. Every design in the solution space can be fully represented with an image. Most real-life design problems can be visualized similarly using software tools which provide state-space information to the human users along with a list of actions to modify/create the design. Decomposing design as a sequential decision-making process simplifies it to the schematic shown in Fig. 1. Having an image as the primary representation of a state can allow developing a framework that can be used across multiple problems since there is no problem-specific modeling in terms of actions or design variables. Even though the truss design problem is visually simpler in comparison with real-life mechanical design problems, it still captures the basic challenges of perception followed by selecting actions and relevant parameters. The focus of the current work is on developing this method for a truss design problem, however extending to other problems is a tangible next step and the effectiveness shall be tested and discussed in future work.

### 3.2 Problem Formulation

The problem that this work deals with is formulated as a prediction problem, where the framework is expected to predict the next design state given previous state information. The dataset used contains a set of sequence demonstrations \( \{D_1, D_2, \ldots, D_n\} \) of human-generated designs from the truss study. Each demonstration is a sequence of state observation \( D_t = \{s_{t1}, s_{t2}, \ldots, s_{tn}\} \) where \( s_t \) is the observation at time (analogous to iteration number, since one iteration is one time unit) \( t \) in the \( t \)th demonstration sequence and is an image that shows the current design (truss structure). Each designer creates multiple demonstrations and all demonstrations begin from the same initial state. Every demonstration is a trajectory through different intermediate design states that eventually leads to a final design, representing a search through the design space. This dataset does not contain any parameters for the metrics for node and member or information about the loading parameters. However, there are implicit relationships between consecutive states corresponding to design operations which lead to changes in pixel intensities. The framework (in Sec. 3.3) aims to learn those implicit relationships among these states from the dataset since they correspond to designer strategies for the given problem and boundary conditions.

This task to replicate human strategies can be formulated as learning a function that converts \( s_t \) to \( s_{t+1} \) (\( 1 \leq t < T_n \), where \( T_n \) is the total time for the \( t \)th demonstration). In order to develop new designs in the study, humans were provided with a set of nine operations \( \{A\} \) that change the existing designs; the set \( A = \{a_1, a_2, \ldots, a_9\} \), and each of the operations have a parameter set associated with them \( \{w\} \). Hence, \( o_{t+w} = a_j(\theta, w) \) represents generating a new design by applying operation \( j \) with parameters \( w \). The whole design process can be seen as a designer analyzing the current state \( \theta \), selecting and applying one of these operation functions with relevant parameters, and then repeating the process all over again. The designer looks at the current state and makes the decision about the operations and the parameters using implicit design knowledge and is able to eventually create a well-performing truss structure. The database captures that information as it preserves the sequence of the progression of design. The desired framework must satisfy the following requirements:

- (a) ability to semantically interpret the design features of state \( \theta \);
- (b) create an embedding that can represent large variability in the designs with less features;
- (c) map consecutive design states to learn how design operations function;
- (d) learn how designs progress iteratively to generate design trajectories.

The first two requirements relate to the perception part of the problem and the next two relate to problem solving. In order to fulfill them, two different neural network architectures are used. They are explained in the next section.

### 3.3 Model Framework

To learn the low-dimensional embeddings of the design data, a convolutional autoencoder is designed. It has two parts, encoder \( (e(\cdot)) \) and decoder \( (d(\cdot)) \). The encoder is used to encode the raw pixel data into a semantically meaningful embedding that can capture all the important features of the structure at a reduced size. The decoder on the other hand is used to convert this low-dimensional embedding back into the original size image. This step helps visually realize the image. Another neural network is used as a transition network \( (t(\cdot)) \), that takes the embedding from the encoder as the input and converts it into a new design embedding that helps in progressing the design, and which is regained by the decoder. Figure 3 shows the complete deep learning framework.

#### 3.3.1 Perception—Dimension Reduction

This part of the framework deals with the perception of the design. The raw designs are high dimensional and using them to predict other high-dimensional designs is computationally intensive. Thus, identifying semantically meaningful embeddings that are low dimensional will make later operations less computationally intensive. This was achieved with a nine-layer autoencoder (as shown in Fig. 4).

Dimension reduction was achieved in the encoder by using a combination of three layers of stride 2 convolutions [67] and a max-pool layer. The kernel sizes for the convolution layers were in decreasing order \((12 \times 12), (9 \times 9), \) and \((5, 5)\) with the increasing number of channels—32, 64, and 128, respectively, in order to capture large features in the first layer (like nodes and member thicknesses). For the decoder, one unpooling layer and three deconvolution (or convolution transpose [68]) layers with stride 2 were used for upsampling the dimensions. The kernel sizes were in increasing order \((7, 7), (9, 9), \) and \((12, 12)\). The number of channels were the same as the encoder but in reverse order \((128, 64, 32)\). Nonlinear activations were used for all the layers, all convolution layers except for the final layer used ReLU activations [69] because of their efficiency in deep networks, while the linear layer at the center and the final convolutional layers uses sigmoid.
activation to avoid gray value and forcing the network to make a decision for every pixel (since sigmoid is biased toward 0 or 1 values). The dataset was split into test and train images with a 80-20 split (80% training, 20% test data) and the autoencoder was trained in an unsupervised manner since there are no labels and the network automatically identifies the optimal latent representations. The main objective for training an autoencoder is to reconstruct the same input so that the output is the same as the input. Also, an information bottleneck is designed in the hidden layers to reduce the dimensions. The autoencoder reduces the initial 76 x 76 image (5776 dimensions) down to 512 dimensions in the embedding. A mean square error (MSE) was used as the loss function to train the autoencoder. Adam optimizer [70] was used to train the weights of the network. A sufficient low value of 0.0012 MSE was achieved with a binary accuracy of over 91% over the test set. An R-squared value of 0.9684 is achieved in the test case, indicating that the network is capable of explaining nearly all of the variance in the data.

3.3.2 Prediction—The Transition Network. This part of the framework is a nonlinear function that maps the embedded representations of designs in consecutive time steps. This function basically manipulates the current designs to generate a new design that shall eventually lead to a complete final design. This manipulation of the current designs to generate a new design is similar to how a human designer approaches a design problem, in a step-by-step manner taking feedback and applying relevant operations. A neural network is used to approximate this function. The network takes five previous design embeddings as input (5 x 512 units), concatenates it, and then down-samples it in the two linear layers (1024, 512 units) back to 512 units. A nonlinear activation function Leaky ReLU [71] was used for all the linear layers. Multiple design states are concatenated in order to make sequential information accessible to the network for better predictions [3]. The weights of the neural network implicitly represent the design operations that are applied to manipulate the design states. These operations modify the design embeddings by adding, deleting, or manipulating their values (matrix operations) in order to generate new designs. Since this step manipulates an abstract representation of a visual image, it is similar to how humans build designs [21].

This network was trained after the autoencoder. Training data for the transition network consist of embedding sequences from the design study. \( S = \{ e(\phi_i), e(\phi_0), ... e(\phi_i) \} \) \( \forall i \in H \) (all sequences) is the total set of embedding sequences, for the purpose of this network with 5 inputs, the dataset was converted to \( S = \{ [e(\phi_0), e(\phi_1), ... e(\phi_i) \}, [e(\phi_0), e(\phi_1), ... e(\phi_i) \}, ... e(\phi_0), ... e(\phi_i) \} \) \( \forall i \in H \). The semicolon separates the input data from the labels, i.e., the next design state in the sequence. For training, test and training sequences were separated with a 80-20 split (80% training data, 20% test) and the network was evaluated on how well it performed on the test set. This converts the imitation problem into a supervised learning problem which was based on the previous five design embeddings the next embedding needs to be predicted. The training data used is only the state sequences from the design study, so the network is only trying to learn the implicit relationships between the states; it has no explicit understanding of design goals, constraints, or intent. The transition network was combined with the encoder and decoder networks as shown in Fig. 5 to complete the deep learning framework. For the training of the transition network, weight values for the encoder–decoder networks were kept fixed and MSE loss was calculated. Adam optimizer [70] was then used to backpropagate the loss values to train the weights.

The training was accomplished in a 2-step process where the first step pretrains the network and the second step fine-tunes to predict meaningful designs. In the first step, the network must learn to recreate the current design itself. This pretraining step is important since the weights of the network are initialized randomly and training may not lead to meaningful predictions [72]; pretraining helps in finding better weights by providing good initialization to the network weights. In this fine-tuning step, the learning rate is lowered, and the network is trained to recreate the next design in the sequence. Through experimentation, the mentioned architecture was finalized for the transition network, a final MSE of 0.0072 (or binary accuracy of 90.05%) is achieved for the test case and an R-squared value of 0.8105 was achieved by the final network. This indicates that the network was able to explain a large majority of the variance in the data. The final network was arrived at through an iterative hyper-parameter search, where numerous architectures were compared, preference was given to smaller networks as larger networks tend to overfit the data. A major boost in performance was seen when past design states were concatenated in the input, showing the importance of the relation between design trajectories and next state prediction.

3.3.3 Framework Results. After training both the autoencoder and the transition network, updated weights were put together in the overall framework as shown in Fig. 5. The input side of the framework consists of five instances of the encoder networks. The embeddings generated by those networks are then concatenated together and a 3-layer neural network down-samples the embeddings before the decoder network and converts the embedding back to the full image size. The final generated image from the framework within an iteration is the suggestion of how the next design should look, hence creating a mental image of the design solution. However, these are still pixel-based information and specific parameters need to be identified to create an actual design.

To visualize the suggested move better, a difference of the generated and input image is taken and then overlaid as a heatmap.

![Fig. 4 Network architecture for the convolutional autoencoder](image)

![Fig. 5 Network architecture for the complete framework](image)
on the original image. Figure 6 shows three images: the current design, the suggestion heatmap, and the ground truth (i.e., the actual subsequent design from data). The suggestion heatmap shows the prediction of design change as a color gradient on the current design state. The color gradient spans across pink to green with 0 (no change) values being represented as black. Pink regions represent the regions where the network adds materials in the prediction, while at the green regions, it deletes or reduces the material. Black implies no change is predicted in those regions. The ground truth is the next design state from the original dataset, showing the actual design created by the designer in the study. A heatmap that matches the ground truth shows the perfect prediction. Figure 7 shows some more sequences that have been taken from the test set which contain designs that were not seen in the training phase. From both Figs. 6 and 7, it can be observed that the network successfully learns to predict semantically meaningful pixel regions (that appear like discrete operations relating to nodes and members) and also correspond to the operations undertaken by the human designers in the test dataset. This shows that the framework was able to extract design features from the problem space along with learning to develop truss designs to match human operations using supervised imitation learning. However, it can be seen that there is some level of noise in the heatmaps which can be improved in future work by using more advanced architectures and other methods of training. Currently, the framework is able to provide exceedingly well results using very basic architectures, utilizing more complex architectures and methods may be needed for more complicated visual setting (more than nodes and lines) and also may lead to better results in terms of noise and efficiency; however, for the scope of current work, the framework appears to be sufficient. The network is also seen to produce several suggestions at once, showing that it is unsure about which move to take. That may be because of several reasons: at a given design state, there may not be a single best operation to take that mimics a human strategy or different sequences of operations can be used to reach the same final state hence which operation to do first could be arbitrary; or at a particular state, the design can go in various possible trajectories and if the network is unsure it produces more noise and less discrete features; also the network could possibly be learning an “average” of multiple designers and their strategies, since it is not trained to only follow a particular style of design generation (although that would be feasible with the current network). In future work, further analysis needs to be done that can identify the primary factor for multiple operation suggestions. Since the heatmaps do seem to follow a singular “design idea” and the suggestions are coherent, the current performance is sufficient and promising for it to act as a design suggestion engine. Now, in order to evaluate how well the suggestions perform in an actual design process, an inference algorithm (similar to having a control system) needs to be designed that can map these suggestion heatmaps (ideas) to the operator parameters (actions) and then select and implement a single operation from the list required to sequentially generate a parametric truss design.

3.4 Developing a Deep Learning Agent. The framework generates the design as a picture and the suggestions/changes are shown as the highlighted pixels in the heatmaps. These images are basically how the deep learning framework imagines/suggests that the design should progress in the next few steps and is a pictorial visualization with no detailed information about which parts are members, their size, or where the nodes are. However, in order to evaluate a design, it needs to have parametric information for the nodes and members involved in the change/suggestion. Interpreting this heatmap information to make changes to the current design is a trivial activity for humans; however, it is very complex to map the process algorithmically. A rule-based algorithm was developed to act like the control system of the framework which can automatically inference from the heatmaps and map pixel-based suggestions to operator parameters. When combined with the framework, this completes the process of generation of a new design state given the current design states. Details of the rule-based inference algorithm are explained in the Appendix.

In brief, the algorithm uses image-processing techniques to preprocess and isolate different regions that may correspond to distinct operations from the heatmap. Further, the regions are analyzed
based on the different parameters to check and classify it as a particular operation, and finally, a candidate list is prepared based on the different regions; these candidate lists can appear similar to Fig. 8. The final step in the algorithm is to select one operation from the list to finally apply on the design. This is achieved by comparing the similarity between the suggestion and the candidates and the one with the highest similarity is probabilistically selected. Hence, the algorithm goes through these different steps and outputs the particular operation parameters to change the design of the truss structure. The framework is shown in Fig. 9, it can be considered an independent agent as it senses an environment, makes decisions, and then acts on the environment generating new designs [73]. The design state is the sequence of the previous five designs which is used as an input for the framework. This engine generates a suggestion heatmap, which is fed into the inference algorithm. This algorithm processes the suggestions and classifies the pixel representation into a design operation. This move is then executed, and a new design is generated.

4 Experimental Setup

This section details how the experiments were carried out to evaluate the quality of the framework by creating a team of identical deep learning-based agents. In order to evaluate the performance of the agent algorithm, we compare the generated designs with human data from the original design study. The design study was conducted in a team (of three designers) and they interacted at pre-defined intervals. When interacting, members of a team shared their designs with the option for each designer to select any one of the three designs in the team to continue working on (typically the best design of the triplet). In order to make a fair comparison, three different agent instances of the algorithm were combined into teams and they were made to interact periodically at the same average rate as the humans, at which point the best design from the agent team is accepted by all agents. This greedy method of interaction was chosen as a simple approximation to the process of human interaction in a team, which is complicated and modeling that is beyond the scope of the current work. This is the only step where the team of agents is implicitly aware of the metrics. However, this step is separate from the design generation steps of the agents (heatmap generation and operation selection) and occurs in <2% operations; hence, the agents individually still continue to work in a metric agnostic manner for generating trusses. The complete loop is shown in Fig. 9 and is repeated for 250 iterations per agent (equivalent to the human average). This whole setup was repeated to produce 16 team solutions in order to compare with the 16 design teams in the study.

5 Results and Discussion

Data from the experiments, generated by deep learning agents (DLAgent), and human designers are analyzed in this section. Figures 10–12 show the plots for performance metrics: FOS and SWR with respect to the number of iterations. The lines in the background connect the data points (designs) from the same designer showing trajectories of how the design progressed. The comparison is made between solid and dashed trajectories where solid represents the DLAgent and dashed is for Human data. The plot presents the data for all 48 humans and 48 DLAgents. The graph limits to only 250 iterations for the purpose of better visualization since the majority of the data can be shown within this range; for calculating the performance values, all iterations are considered. Due to
the nature of truss design, whenever a new node is added, it is virtually hanging in the air and hence the overall structure is infeasible. This leads to zero values of FOS and SWR, which is not a good representation of the quality of the truss structure since, minus that extra node, the structure could be feasible. This leads to a significant amount of noise in the data where the quality metrics unnecessarily dive to zero. In order to avoid that, a denoising operation is carried out once a design becomes acceptable in the process, the most recent nonzero value is adopted as the current value. This way the data is denoised while also maintaining the trends. The bold solid and dashed lines show the mean values at the given iteration for the respective class. Mean values are shown in order to provide a sense of the average trend in the data. Figure 10 shows a plot of the FOS over time. The dotted line shows the threshold for a feasible design (FOS = 1); all designs above that line are feasible. It can be observed that although a few human designers reach very high values like some overperforming humans, there still are a considerable number of designs that are feasible and have good SWR, and the agents are much more consistent with the quality of their designs. Observing the mean lines, DLAgents seem to perform quite close to the humans throughout and slightly exceed them in the later part of the process. The gradual increase in SWR throughout the process shows that the deep learning framework was able to learn certain design strategies from human data and is able to implement them in the generation process.

Figure 11 shows the plot for comparing the SWR over time. Humans appear to perform better than the DLAgents especially in the early part of the process since the dashed mean line is higher than the solid line. Select individual human designers also seem to have reached much higher values throughout the process. However, it should be noted that a high SWR can be achieved with infeasible FOS (<1.0) but very low mass values. This motivated us to calculate a refined strength to weight ratio (RSWR) value as shown in Fig. 12.

For Fig. 12, only designs that are feasible (with FOS ≥ 1.0) are allocated an SWR value unlike in Fig. 11 where an infeasible design could also contribute in representing the quality of the design. Here, a majority of the human data points are removed, showing that most of the designs in the early part of the process with high SWR were actually incomplete or infeasible designs. We observe that even though designs by DLAgents do not reach very high values like some overperforming humans, there still are a considerable number of designs that are feasible and have good SWR, and the agents are much more consistent with the quality of their designs. Observing the mean lines, DLAgents seem to perform quite close to the humans throughout and slightly exceed them in the later part of the process. The gradual increase in SWR throughout the process shows that the deep learning framework was able to learn certain design strategies from human data and is able to implement them in the generation process.

Figure 13 shows the mean value across the best designs created by individual designers for the two performance metrics. It shows that the DLAgent performs better in terms of FOS and comes considerably close in terms of SWR. This implies that the DLAgents have learnt to mimic certain aspects of the human strategies and learnt to generate design sequences even after they started from mere random initial points. It must be noted that the DLAgents are generating designs only on the basis of transition features learnt from data. On the other hand, the humans had access to various metrics like FOS, mass, and also a colored grading showing which members were under stress at every iteration. All of this information guided them to produce better designs in the process. Agents receive no information about mass and hence cannot get direct feedback for how heavy the design is which makes limiting it difficult, possibly leading to achieving higher mass in their structures. This helps the designs with strength (FOS) but adversely affects the SWR. However, as seen later in the SWR result plots the agents maintain good SWR values meaning that the agents learn to efficiently utilize the extra mass. It can be concluded that the agents have learned to create feasible good performing designs; however, humans were still more efficient in the early parts of the process indicating a room for improvement.

6 Conclusion

Human designers possess great problem-solving skills. Essential insights can be extracted from them to develop computer agents that can work with humans in unison as a team. A novel methodology is presented in this paper that is used to develop agents that learn purely from human data and generate high performing designs. The methodology involves training a deep learning framework on historical human design data just by observation. The framework learns to generate designs without any specific design operation information. The generated design is basically a visualization (imagination) of how the new design should look like. Then, that design is fed into a rule-based algorithm that acts as the control system of the agent. Its task is to apply design operations to realize the visualization. This work integrates methods from deep learning, concepts from psychology, and behavior modeling to achieve design automation in a comprehensive end-to-end
A two-step deep learning framework is developed that can perceive the design state information from pixels and then generate new designs after learning to imitate human designers. Finally, a rule-based inference algorithm converts the image into a parametric truss structure which allows the algorithm to act like an agent as it can now iteratively work on its own to generate complete designs in a manner similar to humans.

The comparative results show that the agent learns to create designs that are comparable with human designs from the study. While making this comparison, it must also be considered that the agent did not have access to real-time feedback on the quality of the design or the objectives of the process (only the team interaction was implicitly guided by it). Humans, on the other hand, could get feedback at every step helping them in the process. Moreover, the agents start the design process from randomly generated initial states, while humans begin on their own. However, irrespective of the differences the agents were seen to perform exceedingly well and achieve feasible designs that are even comparable with human level. This implies that the agents were able to extract implicit design strategies from data and use them to generate designs. The framework has the ability to learn the relationships between consecutive design states and hence can be applied to consecutive decision-making design processes whose sequential states have incremental changes and can be represented as an \( N \)-dimensional array.

Learning from historical human data helps the agent identify the important regions of a design space and implicitly learn to navigate it, helping reach good performing designs. This imitation learning, however, has its limitations since the agent can at best do as well as the teacher, i.e., the human in this case, and also it may only learn to work with designs similar to shown in the training data in case of overfitting. In order to overcome this, the networks are trained separately to first create design embeddings that can learn general features of a truss design and hence can represent even new unseen truss designs and then learn generic design operations on them. This can allow the agents to explore new designs by possible interpolations between design embeddings. The agent might still learn bad strategies in case the quality of the subject is bad. The aim for the current research was to illustrate an ability to extract design strategies from data however in future work knowledge about metrics could be infused to ignore low performing strategies along with a careful selection of training data. Further experimentation also needs to be carried out to test the novelty in designs and how different they are from the training data to evaluate the generality of the learnt operations. This motivates us to utilize and test other methods of learning in the agents as well as identify the effect of training data on the final performance. For the current study, the agents were only trained to imitate humans using metric agnostic design data. In the future, embedding these agents in a goal-driven environment where real-time rewards are provided for optimal designs can lead to interesting results. Also, creating hybrid approaches of imitation learning with other methods of active learning can also be explored since it can significantly enhance agent performance.

Acknowledgment

This paper has been submitted for publication in the 2019 Design Automation Conference (IDETC). This material is based upon work supported by the Defense Advanced Research Projects Agency through cooperative agreement (Grant No. N66001-17-1-4064; Funder ID: 10.13039/100000185). Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

Appendix: Details of the Rule-Based Image-Processing Algorithm

The algorithm is developed to infer operation parameters from suggestion heatmaps generated by the deep learning framework. This algorithm works as a control system of the agents which helps them realize the visualizations of the next design, generated by the deep learning framework. The rules are specific to the design problem at hand and would need to be modified for application across problems. However, the deep learning framework can remain consistent. The algorithm provides different methods for identifying moves related to nodes and members. They are detailed as follows (also shown in Fig. 14), through three basic steps:
a preprocessing step, a classification step, and, finally, the selection and application of operation.

A.1 Preprocessing and Isolating the Suggestions

Node. The preprocessing step involves thresholding the heatmap so that node regions can be clearly identified and be isolated from noisy pixels. For isolating the suggestions, a counting islands algorithm [74] is used to identify pixel regions with high intensity to isolate multiple possible nodes and each region is further analyzed in the next part of the algorithm to determine its candidacy.

Member. For detecting members, it is essential to preserve the structure of the shape-highlighted pixels. Hence, a combination of dilation and erosion algorithm [75] is used to enhance the suggestion features/shape and then a threshold is applied to get rid of the extra noise. Applying these processing techniques have shown to be useful while extracting meaningful features from an image [75] and showed improvements in inference results for the current work.

A.2 Classify the Operation(s)

Node. Based on the pixel intensities, the center of mass is computed for each of the isolated regions from the previous step. The location of this point defines the parameters associated with this node operation. Finally, feasibility conditions are checked to ensure that it is possible to apply the operation. Operations that are associated with nodes are “Add a Node” and “Delete a Node.”

Member. For classifying member-based operations, every pair of nodes is evaluated since, in order to create a member, there must exist two nodes in the design. Based on the intensity and the number of the highlighted pixels in the region between a pair of nodes, that candidate pair is further evaluated for feasibility and a relevant operation is assigned. Operations associated with members are “Add Member”, “Delete Member”, “Increase thickness”, and “Decrease thickness”.

A.3 Select and Apply the Operation. Once the heatmap is evaluated for all possible suggested operations, a candidate list is generated with the operation type and relevant parameters. Finally, the candidate image that is most similar to the framework generated design is probabilistically selected. This similarity is evaluated by comparing the image structure (using SSIM index [76]). A higher SSIM value means that the selected candidate image is closer to what the deep learning framework suggested. This metric is, however, not a perfect method to identify the best candidate from the list since the similarity can be biased toward high pixel operations like adding members in comparison with adding a node which involves smaller pixel regions. In order to overcome that, a probabilistic component is introduced in a selection of the final operation from the candidates.

References
