Neural Networks 98 (2018) 305–317

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

A deep learning framework for causal shape transformation

Kin Gwn Lore, Daniel Stoecklein, Michael Davies, Baskar Ganapathysubramanian, Soumik Sarkar *

Department of Mechanical Engineering, Iowa State University, Ames IA-50014, United States

ARTICLE INFO

Article history:
Received 28 March 2017
Received in revised form 28 November 2017
Accepted 4 December 2017
Available online 18 December 2017

Keywords:
Sequence learning
Shape transformation
Convolutional neural networks
Stacked autoencoders

ABSTRACT

Recurrent neural network (RNN) and Long Short-term Memory (LSTM) networks are the common go-to architecture for exploiting sequential information where the output is dependent on a sequence of inputs. However, in most considered problems, the dependencies typically lie in the latent domain which may not be suitable for applications involving the prediction of a step-wise transformation sequence that is dependent on the previous states only in the visible domain with a known terminal state. We propose a hybrid architecture of convolution neural networks (CNN) and stacked autoencoders (SAE) to learn a sequence of causal actions that nonlinearly transform an input visual pattern or distribution into a target visual pattern or distribution with the same support and demonstrated its practicality in a real-world engineering problem involving the physics of fluids. We solved a high-dimensional one-to-many inverse mapping problem concerning microfluidic flow sculpting, where the use of deep learning methods as an inverse map is very seldom explored. This work serves as a fruitful use-case to applied scientists and engineers in how deep learning can be beneficial as a solution for high-dimensional physical problems, and potentially opening doors to impactful advance in fields such as material sciences and medical biology where multistep topological transformations is a key element.

1. Introduction

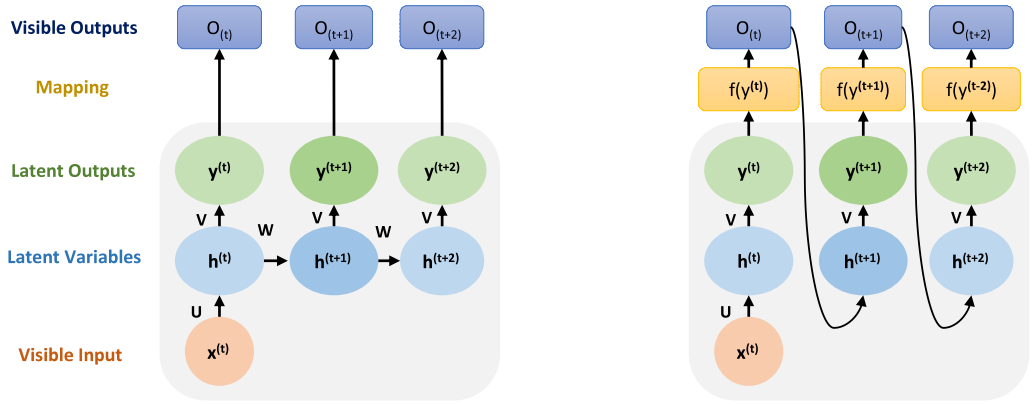
In the past years, hierarchical feature extraction has been very successful in accomplishing various computer vision tasks spanning over generic object detection, image enhancement, and medical imaging (Havaei et al., 2017; Larochelle & Bengio, 2008; Lore, Akintayo, & Sarkar, 2017; Redmon, Divvala, Girshick, & Farhadi, 2016; Ren, He, Girshick, & Sun, 2015). Aside from extracting image features typical of such applications, deep learning methods have also proved to be promising in engineering tasks such as multi-modal sensor fusion (Ngiam, Khosla et al., 2011; Srivastava & Salakhutdinov, 2012), prognostics (Akintayo, Lore, Sarkar, & Sarkar, 2016; Sarkar, Lore, & Sarkar, 2015), and robotic path planning (Hadsell et al., 2008; Levine, Pastor, Krizhevsky, Ibarz, & Quillen, 2016; Lore, Sweet, Kumar, Ahmed, & Sarkar, 2016). In this paper, we propose a methodology using the fusion of two deep learning architectures as an inverse mapping to solve a high-dimensional one-to-many problem typical to the engineering domain. Specifically, we apply the proposed framework on a flow sculpting problem, where a specific sequence of transformation steps is desired given the initial and final cross-sectional shape of the fluid flow. The same framework can be applied in other forms of design engineering problems such as manufacturing, chemical engineering, and biology where an unknown sequence of processing steps is desired to achieve the observable final product.

For sequence prediction, recurrent neural networks (RNN) (Mikolov, Karafiát, Burget, Cernockỳ, & Khudanpur, 2010) and long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) are the common go-to architectures for exploiting sequential information where the output is dependent on previous computation. However, dependencies of the computation of such architectures typically lie in the latent domain. This becomes sub-optimal or even unsuitable for certain applications involving the prediction of a step-wise transformation sequence that is dependent on the previous computation only in the visible domain (Fig. 1).

As a solution, a framework is constructed to handle such a scenario by chaining two popular deep architectures into the pipeline and avoids tedious re-implementation for a complex model. In the context of the problem under study, the resultant framework simultaneously predicts the intermediate shape between two flow shapes and learns a sequence of causal actions contributing to the shape transformation in the visible domain. This topological transformation framework can be extended to other applications, for example, in learning to transform the belief space for robotic path planning (Zhang, Kahn, Levine, & Abbeel, 2016), sequential decision making in games (Silver et al., 2016), learning the material

* Corresponding author.
E-mail address: soumiks@iastate.edu (S. Sarkar).

https://doi.org/10.1016/j.neunet.2017.12.003
0893-6080/



(a) Dependencies in the latent domain.

(b) Dependencies in the visible domain.

Fig. 1. Difference between a typical problem using an RNN (left) and the problem considered here (right). Both frameworks take in input vector **x** at index t, pass it through the hidden layer **h** and produces a latent output **y**. The latent outputs **y** are mapped to visible outputs **o** via function $\boldsymbol{f}(\cdot)$. In RNN, there are dependencies in the latent layer. In our hybrid approach for the considered problem, the visible output is used as the input to the next (i.e. dependent on the visible outputs) without dependencies in the latent layer.

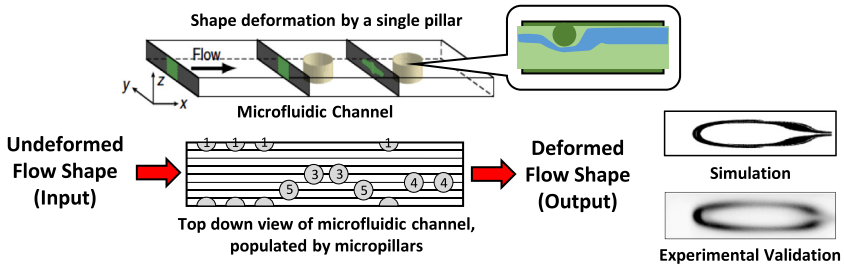


Fig. 2. Pillar programming. Each pillar contributes to the deformation of the flow. The position–diameter pair of each pillar is assigned an index which will be used as class labels for classification.

processing pathways to obtain desired microstructures starting from an initial microstructure (Wodo, Zola, Pokuri, Du, & Ganapathysubramanian, 2015), and learning a sequence of manufacturing steps in additive manufacturing (Paulsen, Di Carlo, & Chung, 2015), with *fast-design* being the main advantage without prolonged wait times in generating candidate solutions (see Section 6 for more elaborations). The contributions of the paper are outlined next:

1. A formulation of learning causal shape transformations to predict a sequence of transformation actions is presented, in the setting where only an initial shape and a desired target shape are provided.
2. An integrated hierarchical feature extraction approach using stacked autoencoders (SAE) (Vincent, Larochelle, Bengio, & Manzagol, 2008) with convolutional neural networks (CNN) (Krizhevsky, Sutskever, & Hinton, 2012) is proposed to capture transformation features to generate the associated sequence resulting in the transformation, specifically for problems where RNN and LSTM-based methods cannot be applied.
3. The proposed approach is tested and validated via numerical simulations on an engineering design problem (i.e. flow sculpting in microfluidic devices), with results showing much superior prediction accuracy over previously explored methods such as evolutionary algorithms and a multi-class, multi-label classification framework.

The rest of the paper is structured as follows: In Section 2, we discuss the problem setup and present some of the previous approach taken to solve the problem. Upon examining these previous approach, the motivation for a superior formulation is described in Section 3 . We present the proposed method in Section 4 and exhibit the results in Section 5. Finally, we outline other possible applications which can benefit from the proposed approach in Section 6 before concluding the paper in Section 7.

2. Problem setup and previous approach

In this section, we describe the problem setup for microfluidic flow sculpting and provide a brief background on previous approach in handling sequence prediction.

2.1. Microfluidic flow sculpting

Inertial fluid flow sculpting via micropillar sequences is a recently developed method of fluid flow control with a wealth of applications in the microfluidics community (Amini et al., 2013). This technique utilizes individual deformations imposed on fluid flowing past a single obstacle (pillar) in confined flow to create an overall net deformation from a rationally designed sequence of such pillars. If the pillars are spaced far enough apart (space $> 6D$, for a pillar diameter D), the individual deformations become independent, and can thus be pre-computed as building blocks for a highly efficient forward model for the prediction of sculpted flow given an input pillar sequence (Stoecklein, Wu, Kim, Di Carlo, & Ganapathysubramanian, 2016) (Fig. 2). Since its debut, flow sculpting via micropillar sequence design has been used in a diverse variety of applications like novel fiber and particle fabrication (Nunes et al., 2014; Paulsen & Chung, 2016; Paulsen et al., 2015; Wu, Owsley, & Di Carlo, 2015), biodiagnostics (Sollier et al., 2015), and tissue engineering (Hwang, Khademhosseini, Park, Sun, & Lee, 2008).

To make the application practical, we require solving the *inverse problem* instead; that is, to generate a sequence of pillars given a user-defined flow shape. This serves as the main motivation where we propose a data-driven approach to learn a sequence of actions that map between two given states. Without intelligent computer algorithms, such tasks require time-consuming trial and error design iterations. The automated determination of pillar sequences that yields a custom shape is an impactful advance. Although researchers have tried to frame this inverse problem as an unconstrained optimization problem (Stoecklein et al., 2016), they are invariably time-consuming. While many methods are used to solve the forward problem (Ashforth-Frost, Fontama, Jambunathan, & Hartle, 1995; Baymani, Kerayechian, & Effati, 2010; Duriez et al., 2014; Müller, Milano, & Koumoutsakos, 1999), only a limited amount of effort has been done in solving the inverse problem (Christiano et al., 2016; Finn, Levine, & Abbeel, 2016; Lore, Stoecklein, Davies, Ganapathysubramanian, & Sarkar, 2015; Pearlmutter, 1989; Stoecklein, Lore, Davies, Sarkar, & Ganapathysubramanian, 2017; Wada & Kawato, 1993). Thus, this serves as a technologically important motivation for using deep learning to map user-defined flow shapes and the corresponding sequence of pillars, in addition to generalizing to problems where learning topological transformation is key.

Note that in the forward problem, a flow shape as a function of predefined pillar sequences can be simulated in the order of less than a second. The simulation yields a binary flow shape image as depicted in Fig. 2, where it has been validated by carrying out the experiment in a physical platform. The quick simulation in the forward direction gives us the luxury of generating millions and millions of flow shapes from a given sequence. We then leverage the capability of deep learning to learn the inverse map of the forward function.

Specifically, the proposed method will be utilized for concrete biomedical applications that require the design of microfluidic devices. Possible applications include: (a) designing a device to move fluid surrounding cells (e.g. lymphoid leukemia cells) against a far wall of the microfluidic channel where it can be collected at high purity while the cells are maintained at the channel center. High purity allows the reuse of valuable reagents for staining cells during diagnosis, (b) wrapping a fluid around the microchannel surface to characterize binding p24 (an HIV viral capsid protein) to anti-p24 antibody immobilized on the microchannel surface. Flow sculpting can enhance reaction of low abundance proteins that can improve diagnostic limits of detection for various diseases. Hence, this study may promise new application areas for the machine learning community related to thermo-fluid sciences, design engineering, and bioengineering.

2.2. Discretization of the design space

To approach the inverse problem, class indices are assigned to pillars with different specifications. For instance, a pillar located at position 0.0 in the center of the microchannel with a diameter of 0.375 is assigned an index of 1, whereas another pillar further away at position 0.125 with the same diameter 0.375 is assigned an index of 2. All diameter and position offset values are non-dimensionalized, which is standard engineering practice. Diameter and position values are represented as ratios with respect to the channel size and locations to help enable scalability of the fluid channel. Index assignment is performed over a finite combination of pillar positions and diameters that has been obtained by discretizing the design space. In the study, there are 32 possible indices (or classes) that describe the diameter and position of a single pillar. This makes the problem space high-dimensional in nature where there is a total of 32^{n_p} possible permutations for an n_p-pillar sequence.

It is possible to approach the problem as a regression problem, where the regressed values are the positions and diameters of each individual pillars. However, discretization of the design space reduces the design space into a finite set (albeit large) and is sufficient to aid designers in obtaining the viable design.

2.3. Previous work: simultaneous multi-class classification with convolutional neural networks (CNN+SMC)

Simultaneous multi-class classification is a method used to solve a similar problem (Lore et al., 2015). Instead of solving a single classification problem, the model solves a sub-problem for each pillar using the parameters learned by the CNN. Essentially, the problem is solved as a multi-class, multi-label classification problem. This formulation trains a model by minimizing the negative log-likelihood function summed over all pillars within a predicted sequence. Thus, for a pillar sequence with length n_p, the loss function to be minimized for a data set $\mathcal{D}$ is defined as:

$$\ell_{\text{total}}(\theta, \mathcal{D}) = -\sum_{j=1}^{n_p}\sum_{i=0}^{|\mathcal{D}|}\left[\log\left(P(Y = y^{(i)}|x^{(i)}, W, b)\right)\right]_j \tag{1}$$

where $\mathcal{D}$ denotes the training set, θ is the model parameters with W as the weights and b for the biases, y is predicted pillar index whereas x is the provided flow shape image. The total loss is computed by summing the individual losses for each pillar (see Fig. 3).

2.4. Previous work: Genetic algorithm (GA)

A previous work (Stoecklein et al., 2016) has attempted to identify pillar sequences for a given fluid flow shape using the current state-of-the-art genetic algorithm (GA). The GA is an evolutionary search heuristic that solves optimization problems using techniques inspired from natural evolution. Broadly, a random initial population of pillar sequences is generated. This population is evolved through crossover and mutation toward a targeted flow shape by penalizing sequences that produce flow shapes different from the target, and rewarding those that are similar. These punishments and rewards are borne out in the breeding process between generations. After many generations of breeding, the population should produce a larger subset of flow shapes similar to the target shape than the initial population.

CNN+SMC vs. GA: Authors in Lore et al. (2015) have shown that GA requires many fluid flow shapes to be produced and evaluated during a search. Therefore, the execution time for identifying a pillar sequence for a single image can take hours (Stoecklein et al., 2016). However, a trained deep network will give results with a runtime on the order of seconds. Furthermore, the choice of cost function will greatly impact the effectiveness of a GA search, and therefore requires laborious tuning. Given the time required for a single run, this tuning process can be considerably more complex and difficult than the training of a deep network. In comparison, most of the time spent via deep learning methods are in training the model. Generating predictions, on the other hand, occurs almost instantaneously. Therefore, deep learning methods allow for near real-time flow sculpting with the process expedited by over 500x compared to GA. More importantly in this case, the user could manipulate a flow shape using a graphical interface that would elicit key expert knowledge.

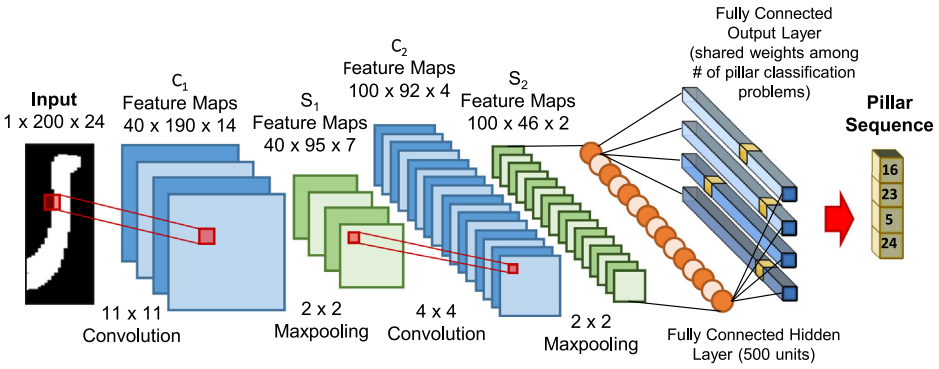


Fig. 3. CNN with the SMC problem formulation (previous approach). A classification problem is solved for each pillar in the sequence.

3. Motivation

The CNN+SMC framework in Lore et al. (2015) is not without drawbacks. Pillars are predicted in a joint manner, where each of the pillars in the sequence are inferred simultaneously. While it may produce satisfactory sequences which generate the desired shape, it is unclear how the successive pillars interact with one another. Extensive efforts have been performed in Stoecklein et al. (2017) to improve performance of CNN+SMC via intelligent sampling of the design space but the performance gain is not huge. Although CNN+SMC is capable of predicting a large number of different sequences in a total time of just seconds, the main drawback is that the sequence length is constrained because a new model needs to be trained to generate a sequence with different lengths.

In the flow sculpting application, a pillar is independently added to the flow channel after the flow stabilizes. Hence, the determination of the next pillar in the sequence is dependent only on the current flow shape and the desired target flow shape, although the current flow shape is a result from the joint contribution of all previous pillars to-date. To clarify, all previous pillars in the sequence are *causal* to the current shape but they do not influence the selection of the next pillar, since the framework only looks at the current and target shape to make a decision. Therefore, RNN-like architectures are, although scalable, deemed unsuitable for this problem due to the internal dependencies within the hidden layers.

A clearer distinction can be made between RNNs and our framework using next-word prediction as an example, where the algorithm is tasked to predict the next word given a partial sentence without access to the word that ends the sentence. In RNNs, the choice of the next word is dependent on all previous words in order to form a grammatically valid and semantically meaningful sentence unconstrained by the terminal condition. In our context, the choice of the next pillar will not depend on the choice of the previous pillar; rather, we focus on looking at the current state and into the future (i.e., the target) to decide on a transformation step that brings us closer to the desired shape.

A related concept is the spatial transformer network (Jaderberg, Simonyan, Zisserman, et al., 2015) where the localization network outputs geometrical transformation parameters; however we desire an exact class attributed to an arbitrary transformation function. In this work, we predict the pillar sequence one pillar at a time without disregarding causality, such that the produced sequences deform the flow into one that resembles more closely to the desired shape.

4. Proposed architectures

In this section, we present two core modules and the integrated framework which fuses these two architectures.

4.1. Action prediction network (APN)

To predict the sequence of pillars that results in the desired flow shape, this chapter introduces the notion of *transformation learning*. Fig. 4 shows the learning approach by supplying juxtaposed flow shapes, one before deformation, and one after, into a CNN-based APN that extracts relevant features and predicts the class of the pillar causing the deformation. In the training data generation procedure (Fig. 4a), the input is comprised of three parts:

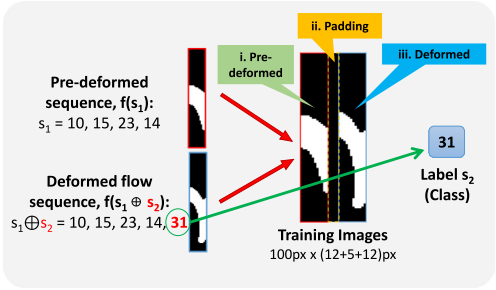
(i) The pre-deformed flow shape generated via forward function f applied on a sequence s_1 of random length, i.e. $f(s_1)$;
(ii) A zero-padding to prevent the convolutional kernels to pick up interfering features from the juxtaposed images; and
(iii) The post-deformed shape $f(s_1 \oplus s_2)$ generated from appending a random pillar index s_2 to the previous sequence s_1, where $\oplus$ is the concatenation operator.

Hence, the input to the APN can be denoted as $f(s_1) \oplus f(s_1 \oplus s_2)$. This newly-added pillar sequence s_2 (of length one) will become the label to train the model. Essentially, the action prediction network predicts the index of the pillar (hence, its position and diameter in the flow channel) given a pre- and post-deformed shape.

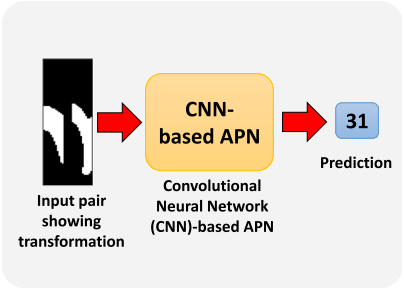
When making inference (Fig. 4b), the right portion (part c) of the input image is replaced by the final target shape. The input image is fed into the CNN to obtain a pillar index, which is appended to an initially empty pillar sequence. Because the forward model (i.e. going from sequence to flow shape) is not at all computationally expensive and takes merely milliseconds, this pillar sequence is used to regenerate the *current shape* which replaces the left portion (leftmost part i in Fig. 4a) of the input, while the right portion (rightmost part iii, the target) remains unchanged. The input enters the CNN again to obtain a second pillar index, and is subsequently appended to the previously obtained pillar sequence. At this point, we have a sequence of length 2, where the sequence will then again used to regenerate a new current shape. The process is repeated until the *current shape* matches the *target shape*, or until a user-defined stopping criterion is met.

An alternative APN is to feed the pre- and post-deformed shapes into the CNN separately as *isolated channels* before merging them together in the fully connected layer (Fig. 5). This architecture will be referred to as APN-C (where the 'C' denotes *channels* and signifies that the pre- and post-deformed shapes are treated as different channels). In other words, the flow shape images are concatenated depth-wise.

However, this formulation (including both APN and APN-C) only works well for simpler target shapes. For more complex flow shapes characterized by many sharp angles, jagged edges, islands,

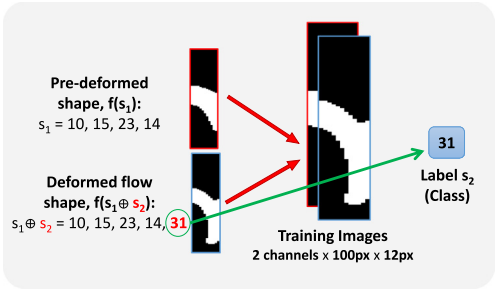


(a) Training data generation for the CNN-based APN.

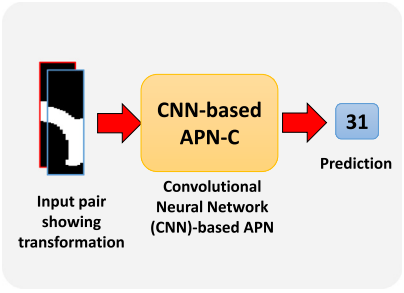


(b) Predicting a pillar given two flow shapes.

Fig. 4. The APN addresses the question: 'Given a pair of pre- and post-deformed shape, what is the identity of the pillar causing the deformation?' (a) Training set generation where the goal is to attribute a label to juxtaposed input–output pair simulated from known sequences. The aim is to learn the incremented action (i.e., the new pillar added to the sequence) that can deform the input shape into the output shape. (b) During inference, the framework takes in two shapes and outputs the associated action represented by an index (in this example, the index 31 uniquely identifies the pillar specifications).



(a) Training data generation for the CNN-based APN-C.



(b) Predicting a pillar given two flow shapes.

Fig. 5. APN-C treats the pre- and post-deformed shapes as separate channels. Training data generation, illustrated in (a), and inference in (b), are similar to the original APN except the dual-channel treatment.

swirls and curls, the transformation path may be highly nonlinear—the *current shape* may never converge to the final desired shape. Furthermore, the training data covers a vanishingly small fraction of the design space with coverage shrinking exponentially as the sequence length increases (i.e., an n_p sequence will result in 32^{n_p} different combinations), so it is necessary to learn the transformations in a meaningful way. To help alleviate this issue, we introduce the Intermediate Transformation Network (ITN) in the next subsection.

APN parameters: The input image into the APN is comprised of two 12×100 px flow shape image with a 5×100 px padding in between, resulting in a final dimension of 29×100 px. For APN-C, the two inputs are treated as separate channels without introducing padding, with an input size of $2 \times 100 \times 12$. The model contains two convolutional layers with 40 and 100 kernels respectively (sizes of 5×5 and 3×3 px), each followed by a 2×2 maxpooling layer. The fully connected layer has 500 hidden units. Training is done with 450,000 training examples and 50,000 validation examples in minibatches of 50 with a learning rate of 0.01 and rectified linear unit (ReLU) activations. We find that using ReLU activations in the hidden layers speeds up convergence as compared to the typical logistic sigmoid/tanh activations, while the output is a softmax layer over sigmoid activations. Training loss was defined by the categorical cross-entropy:

$$\ell = \sum_i t_i \log p_i \tag{2}$$

where t is the target and p is the class prediction. The training procedure employs the early stopping algorithm where training stops when validation error ceases to decrease. The architecture parameters are selected from hyperparameter optimization using random search.

4.2. Intermediate transformation network (ITN)

The ITN attempts to construct a flow shape that bridges between two flow shapes in the nonlinear transformation path. We used a deep autoencoder to extract hierarchical features from the desired final shape as a pretraining procedure, then finetuned the model to approximate the bridging shape or *waypoint*. The ITN evaluates the complexity of the target flow shape (discussed in the next paragraph) and generates a number of waypoints depending on the computed complexity. Hence, we can have a smoother transformation pathway by setting these waypoints as temporary targets for pillar predictions. The training scheme (Fig. 6b) can be formally represented by the following set of equations:

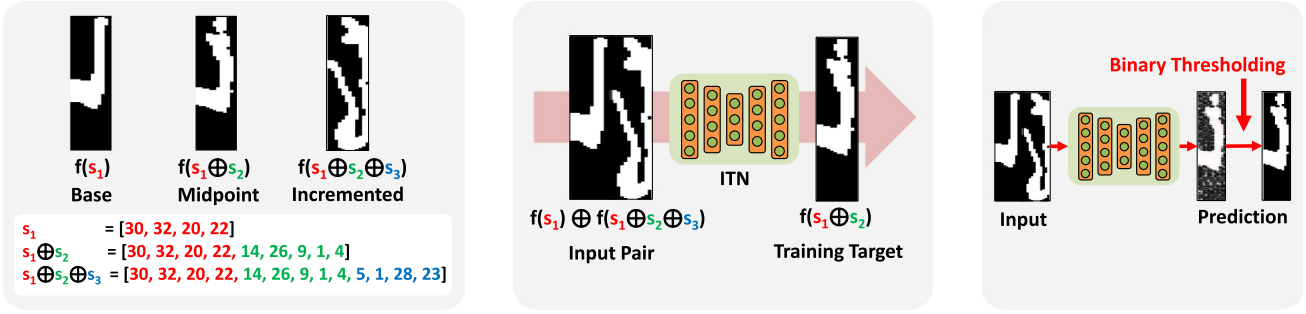
$$\mathcal{D}_{\text{input}} = f(s_1) \oplus f(s_1 \oplus s_2 \oplus s_3) \tag{3}$$

and,

$$\mathcal{D}_{\text{output}} = f(s_1 \oplus s_2) \tag{4}$$

where $\mathcal{D}$ denotes the training data, s_1, s_2, s_3 are sequences of pillar indices, f is the forward simulation function to generate the flow shapes, and $\oplus$ is the concatenation operator. No padding is needed between flow shapes $f(s_1)$ and $f(s_1 \oplus s_2 \oplus s_3)$ during concatenation as this is a fully-connected architecture. Note that the length of pillar sequences s_1 and s_2 are the same, i.e., $|s_2| = |s_3|$. Hence, every intermediate shape $f(s_1 \oplus s_2)$ in the training data lies in the middle of the transformation pathway between flow shapes $f(s_1)$ and $f(s_1 \oplus s_2 \oplus s_3)$ (see Fig. 7).

Number of waypoints: We can enumerate multiple intermediate shapes to be used as waypoints within the nonlinear transformation pathway based on a complexity measure evaluated on the target flow shape. As a metric, we used the perimetric complexity,



(a) Training data generation for the ITN. (b) Training scheme for the ITN. (c) Prediction of intermediate shape.

Fig. 6. The ITN addresses the question: 'Given two flow shapes, what is the possible shape that lies in the middle of the nonlinear deformation pathway?' Details are further elaborated in the main text.

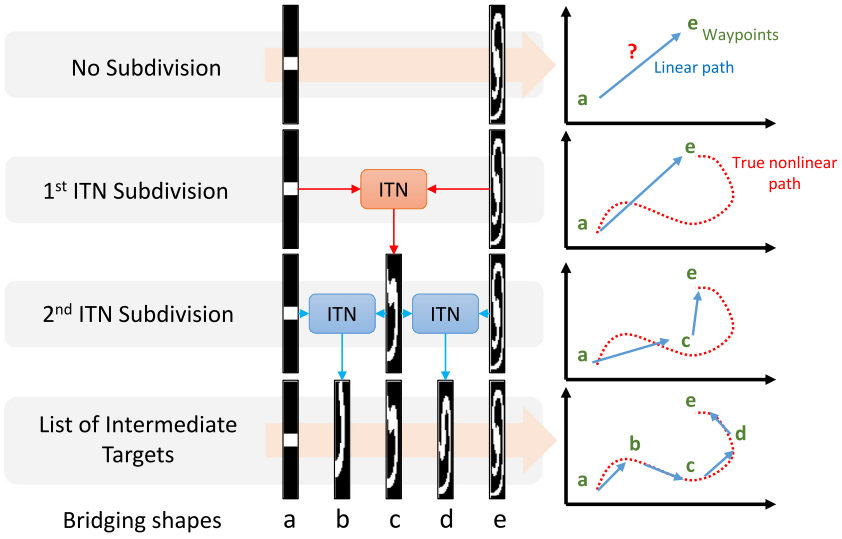


Fig. 7. A properly trained ITN can be used recursively to determine the bridging shape between two shapes. The illustration on the right shows the benefit of integrating ITN into the framework.

$\mathcal{C}$ of binary digital images (Attneave & Arnoult, 1956) described by the following expression:

$$\mathcal{C} = \frac{P^2}{4\pi A} \tag{5}$$

where P is the perimeter of the shape, and A is the area of the shape. Then, $\mathcal{C}$ is used to compute the number of subdivisions d of the transformation pathway:

$$d = \max(\lfloor \alpha \ \ln \mathcal{C} + \beta \rfloor, 0) \tag{6}$$

in order to obtain $2^d - 1$ waypoints. This expression with parameters α and β is obtained from observing the empirical relationship between $\mathcal{C}$ and d, which is more suited to be modeled as exponential for this specific problem. In reality, this decision scheme is another hyperparameter and can be changed. With tuning, we find suitable values for α and β to be 1.6891 and -0.8755 respectively To provide a general idea of the relationship between complexity and the number of subdivisions under this scheme, a complexity of 3 will not require a bridging shape (i.e. no subdivision), complexity of 3.5 will result in one subdivision, complexity of 6 will require two subdivisions, complexity of 10 will require three subdivisions, and so on. For reference, Fig. 8 shows the complexity for various flow shapes. Generalizing the scheme of mapping some measure (e.g. quantifying shape complexity with the Hausdorff measure Hausdorff, 1918) of the target to the number of waypoints will be an important future work.

Waypoint pruning: Additionally, we performed *waypoint pruning* to remove redundant waypoints. Successive waypoints that are visually similar – quantified by the *pixel match rate*, PMR (Section 5.1) – are removed. Similar waypoints above a user-defined threshold are omitted. In this way, we can avoid lengthy enumeration of pillar sequences between two successive waypoints which may sometimes degrade the algorithm performance. An example can be seen later in Fig. 9.

During inference, the outputs of the ITN may appear blurred because the bridging shape is only an approximation with sigmoid activations (Fig. 6c). To obtain binary images, we thresholded the pixel values.

Some may comment that ITN is somewhat similar to RNN. We note that this problem (along with the example applications discussed in Section 3) requires transforming the hidden states into the physical domain using a complex forward function before making the next prediction. Unlike RNNs, ITNs do not solely use the latent output directly as the input in the next time step. Due to the forward function, we need to constantly transform the output back into the physical domain and re-evaluate at every step. One may also argue that ITN is somewhat similar to RNN in terms of *memory*, but by the same argument, the difference in domain is the key. If only the latent domain is used for every step, the final sequence output may not necessarily reflect the true transformation in the physical domain.

Training the ITN: Training was done on 450,000 training examples and 50,000 validation examples in minibatch of size 50 with a

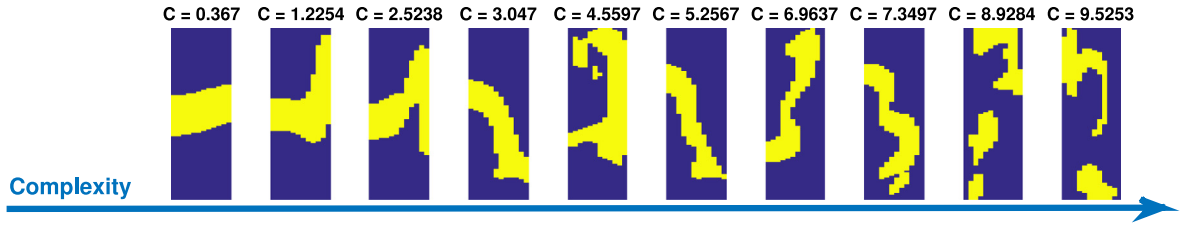


Fig. 8. Increasing perimetric complexity $\mathcal{C}$ for different flow shape examples. Target flow shapes with a chosen value of $\mathcal{C} > 3.5$ may require bridging shapes to achieve good predictions.

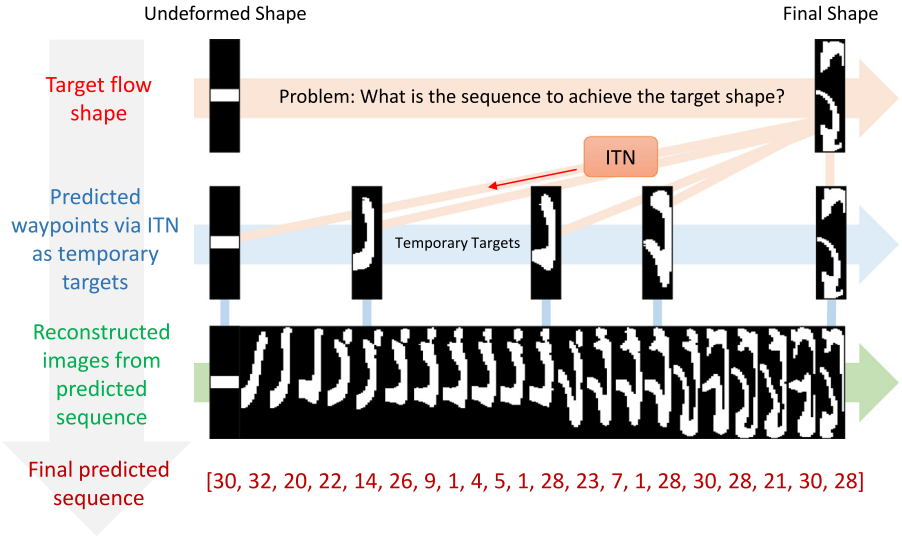


Fig. 9. A higher level overview of the framework illustrated with a complex flow shape. The sequence can be post-processed to remove redundant pillars that do not considerably deform the flow shape.

learning rate of 0.01. Training is also done using the early-stopping algorithm, and hyperparameters are optimized via random search. We used ReLU for hidden layer activations and logistic sigmoid as the output.

Note that the dimensionality of the input image vector is different than that of the output, where the input $\mathcal{D}_{\text{input}} \in \mathcal{R}^{2mn}$ and the output $\mathcal{D}_{\text{output}} \in \mathcal{R}^{mn}$. m and n denote the height and the width of one image of the flow shape respectively. The ITN has five hidden layers with 500, 400, 300, 400, and 500 hidden units connected in succession. The first three layers (500, 400, 300) form the encoder of the stacked autoencoder and the last three layers (300, 400, 500) form the decoder. In unsupervised pretraining, the autoencoder is trained to learn a compressed representation of the input at the 300-hidden unit layer (a form of dimensionality reduction) by minimizing the mean squared error (MSE) between the reconstruction of the input $\hat{\mathcal{D}}_{\text{input}}$ and the original image pair $\mathcal{D}_{\text{input}}$. In this case, the loss function is expressed as:

$$\ell_{\text{autoencoder}} = \frac{1}{2mn} \sum_{i=1}^{2mn} \|\mathcal{D}_{\text{output},i} - \hat{\mathcal{D}}_{\text{output},i}\|_2^2. \tag{7}$$

During the supervised training of the ITN, the decoding part of the autoencoder is updated with new weights. Here, the loss function to be minimized is the MSE between the reconstructed shape $\hat{\mathcal{D}}_{output}$ and the true expected flow shape $\mathcal{D}_{\text{output}}$, expressed by:

$$\ell_{\text{supervised}} = \frac{1}{mn} \sum_{i=1}^{mn} \|\mathcal{D}_{\text{output},i} - \hat{\mathcal{D}}_{\text{output},i}\|_2^2. \tag{8}$$

4.3. The integrated pipeline (APN+ITN)

With the roles of the APN and ITN clearly described, we can now integrate both networks to form an integrated pipeline. A schematic is shown in Fig. 10. At the very beginning, the ITN guesses the waypoints leading to the final shape where the number of waypoints is determined by the shape complexity, $\mathcal{C}$ of the flow. These waypoints are then considered as the current (temporary) target shape, and is concatenated with the undeformed shape placed to its left with a zero padding. The concatenated input is supplied into the APN to predict the first pillar causing the deformation, and the pillar index is appended to an initially empty sequence. Now, we have a sequence of length 1. Using this sequence, the current shape is regenerated to update the left portion of the APN input for the next APN iteration in order to obtain the second pillar index. The predicted pillar index is appended to the sequence one iteration at a time until either: (i) the current shape closely matches our temporary target based on a shape similarity metric, or (ii) an iteration limit has been reached. In our experiments, the metric z for the stopping criterion is a weighted combination of the structural similarity index, SSIM and the pixel match rate, PMR:

$$z = \lambda\ \text{SSIM} + (1 - \lambda)\ \text{PMR} \tag{9}$$

with $\lambda = 0.8$ to favor similarities in structural information over pixel-wise comparison. More details on the metrics can be found in Section 5.1.

Next, we repeat the whole process by advancing through all the waypoints generated by the ITN. Eventually, the temporary target becomes our final desired shape and we will obtain the complete

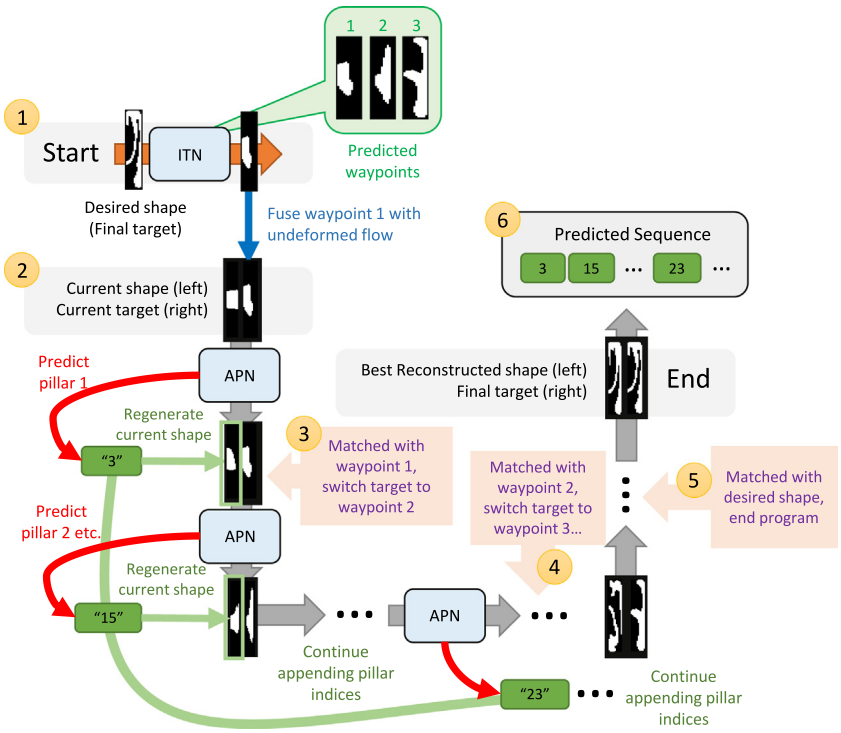


Fig. 10. A detailed view of the integrated pipeline combining both APN and ITN. (1) The ITN generates multiple waypoints that lead to the final shape. (2) Concatenating shapes as input to the APN for pillar prediction. (3) Replacing the temporary target after reaching a stopping criteria. (4) Advancing to the next waypoint. (5) Stopping the algorithm completely after matching the final shape. (6) The predicted sequence is retrieved.

pillar sequence that regenerates the desired flow shape. Note that the resulting predicted sequences vary in length. A pictorial example is provided in Fig. 9 to illustrate the integrated approach.

5. Results and discussions

In this section we first present the evaluation metrics used in the study, then show the results comparing CNN+SMC against our method using APN and the APN+ITN hybrid architecture.

5.1. Image quality metrics

To quantify the effectiveness of CNN+SMC versus our proposed approach, we evaluated the pixel match rate, PMR on 20 target flow shapes and computed appropriate statistics. The PMR, defined in Lore et al. (2015), is computed as follows:

$$\text{PMR} = 1 - \frac{\|\boldsymbol{p} - \hat{\boldsymbol{p}}\|_1}{|\boldsymbol{p}|} \tag{10}$$

where $\boldsymbol{p}$ is the target image vector, $\hat{\boldsymbol{p}}$ is the predicted image vector, and $|\boldsymbol{p}|$ denotes the cardinality of the vector (i.e., the total number of pixels in the image). Since shape transformation is an essential aspect in this problem formulation, the *structural similarity index* (SSIM) (Wang, Bovik, Sheikh, & Simoncelli, 2004) is used as a supplementary metric to compare how structurally similar are the regenerated flow shape images (from predicted sequence) to the target flow shape. SSIM explores the change in image structure and incorporates pixel inter-dependencies. SSIM is expressed as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{11}$$

where μ_x is the average of window x, μ_y is the average of window y, σ_x^2 is the variance of x, σ_y^2 is the variance of y, σ_{xy}^2 is the covariance of x and y, $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$ are two variables to stabilize the division with weak denominator with $k_1 = 0.01$ and $k_2 = 0.03$ by default, and L is the dynamic range of pixel values.

We also considered using Procrustes analysis (Dryden & Mardia, 1998) to first determine a linear transformation of the points in the reconstructed flow shape to best conform to the target flow shape, then compute the similarity measure between the two shapes. It is observed that the Procrustes similarity is positively correlated with SSIM (i.e. the Procrustes similarity is high when SSIM is high, and vice versa). On the other hand, a high PMR may not necessarily correspond to a high SSIM. Hence, results based on PMR are kept while results based on the Procrustes similarity is omitted.

5.2. Predicting sequences with APN and APN+ITN

In all of our tests, the target flow shape is generated from a random sequence. Fig. 11 shows four example target shapes with $\mathcal{C} < 8.0$ with the performance using only APN (without bridging) and APN+ITN (with bridging). In most cases, the APN-only formulation produces pillar sequences resulting in flow shapes that do not resemble the target shape as close as using APN+ITN. This can be clearly seen for cases C and D in Fig. 11. On the other hand, by using the bridging shape as a temporary target, the prediction performance saw great improvements. In addition, we see that most shapes are able to converge to both the bridging shape as well as the target in the APN+ITN formulation. In realistic applications, the sequence may be stored in memory and post-processed to remove the redundant pillars, thus producing a shorter sequence which may increase financial savings during the manufacturing process.

5.3. Comparison of CNN+SMC, APN, APN-C, and APN+ITN

Before, we have seen how the presence of one waypoint significantly aids sequence prediction for shapes with $\mathcal{C} < 8.0$.

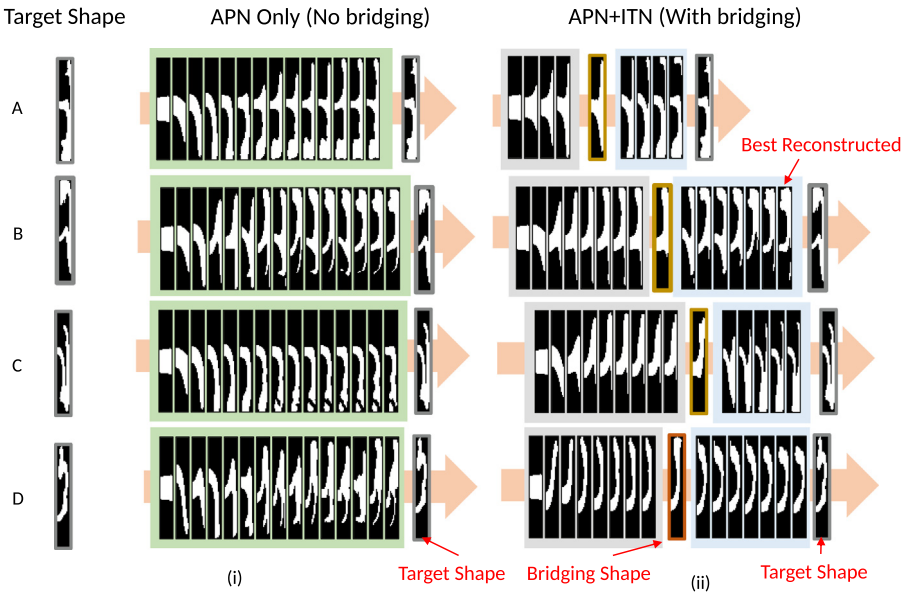


Fig. 11. Four examples of sequence prediction on test shapes with $\mathcal{C} < 8.0$ using (i) APN-only without bridging and (ii) APN+ITN with one waypoint. By predicting a bridging shape, the resulting predicted sequence is able to reconstruct flow shapes that are more similar to the target shape. Each frame shows the deformation on the flow shape with each additional predicted pillar added into the sequence.

Here, we show 20 sample-wise comparison on the performance of CNN+SMC, and our methods APN and APN+ITN are shown in Fig. 13 and tabulated in Table 1. In these tests, all target shapes are highly complex and have $\mathcal{C} > 8.0$. The reconstructed flow shapes from the predicted sequences are shown in Fig. 12. In both Fig. 13a and b, the PMR and SSIM for APN, APN-C, and APN+ITN are clearly higher than the CNN+SMC approach. CNN+SMC may return sequences which regenerate shapes that have a faint resemblance to the target shape, but with completely wrong features. However, APN+ITN is consistently more superior in terms of both PMR and SSIM than the APN-only architecture (i.e. APN and APN-C). This shows that having a bridging shape generally helps in producing sequences that generate complex flow shapes. The inferior performance of CNN+SMC is due to the model being constrained to always generate a sequence of fixed length (i.e., 10 pillars), thus the resultant flow shapes mostly fail to converge.

Statistical significance: We conducted an analysis of variance (ANOVA) by performing a one-sided t-test (under the assumption of heteroscedasticity) to conclude whether the newly proposed APN approach is indeed superior over the initial CNN+SMC network. Since SSIM has a closer relationship to the geometry of the reconstructed flow shape compared to PMR, the metric is used for the t-test. At a significance level of $\alpha = 0.01$, both APN and APN+ITN are shown to be better than CNN+SMC with a p-value of 0.00362 and $2.21(10)^{-}8$ respectively. However, there was not enough evidence that suggests APN-C is better than CNN+SMC. For a non-parametric test approach without making assumptions on the underlying data distribution, we used the two-sided Kolmogorov–Smirnov (KS)-test (Demšar, 2006; Gretton, Borgwardt, Rasch, Schölkopf, & Smola, 2012) to study the statistical difference in performance between two methods with CNN+SMC as the baseline. The KS-statistic for APN, APN-C, and APN+ITN is 0.5, 0.2, and 0.75 respectively, with a p-value of 0.008162, 0.7710, and $8.418(10)^{-6}$ respectively. Hence, we can only conclude that APN and APN+ITN approach is better than CNN+SMC, but APN-C is worse. We repeated the experiments over 5000 test shapes and visualized both PMR and SSIM in Fig. 14. All APN-based approach are reported to have higher SSIM and PMR medians compared to CNN+SMC.

Concatenated shapes vs. separate channels: Additionally, treating the input as separate channels (i.e., APN-C) results in lower performance while all other model parameters are kept equal. This suggests that having a single filter that learns features from both input images simultaneously is more effective in feature extraction compared to learning the features in isolation before merging at the fully connected layer. It is interesting to note that for a significant amount of cases, a blob is shown to be the 'best' reconstruction. This suggests that the shape transformation cannot be learned effectively through filters convolving over inputs with depth-wise concatenation. On the other hand, the APN-only formulation with sideway-concatenation produces better sequences compared to the APN-C approach. This fact provides insights where, for the case of learning a geometrical transformation, it is best to preserve the 2D-spatial features instead of merging the inputs depth-wise since convolution of the filters are performed in two dimensions.

Table 1
PMR and SSIM of regenerated flow shapes using different approaches. The numbers reported are in the format of [PMR/SSIM]. Bolded numbers correspond to the method with the highest PMR and SSIM.

Test shape	CNN+SMC	APN[a]	APN-C[a]	APN+ITN[a]
1	0.617/0.200	0.823/0.447	0.823/0.447	**0.866/0.540**
2	0.566/0.204	0.753/0.296	0.562/0.113	**0.879/0.579**
3	0.598/0.160	**0.891/0.584**	0.872/0.530	0.888/0.567
4	0.623/0.159	0.791/0.412	0.691/0.252	**0.877/0.559**
5	0.552/0.136	0.637/0.105	0.554/0.085	**0.793/0.382**
6	0.686/0.263	0.697/0.214	0.622/0.126	**0.884/0.616**
7	0.680/0.201	0.852/0.464	0.790/0.266	**0.878/0.539**
8	0.485/0.031	0.733/0.335	0.530/0.114	**0.793/0.402**
9	0.666/0.233	0.888/0.559	0.844/0.417	**0.886/0.574**
10	0.645/0.187	0.738/0.276	0.598/0.135	**0.781/0.275**
11	0.561/0.096	0.708/0.133	0.543/0.094	**0.741/0.278**
12	0.546/0.079	0.736/0.326	0.534/0.065	**0.827/0.443**
13	0.589/0.091	0.555/0.017	0.618/0.131	**0.744/0.258**
14	0.585/0.132	0.630/0.072	0.463/0.081	**0.878/0.557**
15	0.504/0.086	0.607/0.122	**0.745/0.255**	0.700/0.251
16	0.633/0.172	0.630/0.149	0.506/−0.011	**0.702/0.232**
17	0.644/0.201	0.768/0.423	0.679/0.186	**0.883/0.622**
18	0.547/0.080	0.534/0.053	0.542/0.088	**0.650/0.185**
19	0.708/0.327	0.873/0.561	0.718/0.174	**0.898/0.631**
20	0.625/0.281	0.751/0.318	0.609/0.161	**0.833/0.460**
Average	0.603/0.166	0.730/0.293	0.642/0.185	**0.819/0.447**
Median	0.607/0.166	0.737/0.307	0.613/0.133	**0.849/0.500**

[a] Denotes our architecture presented in this paper.

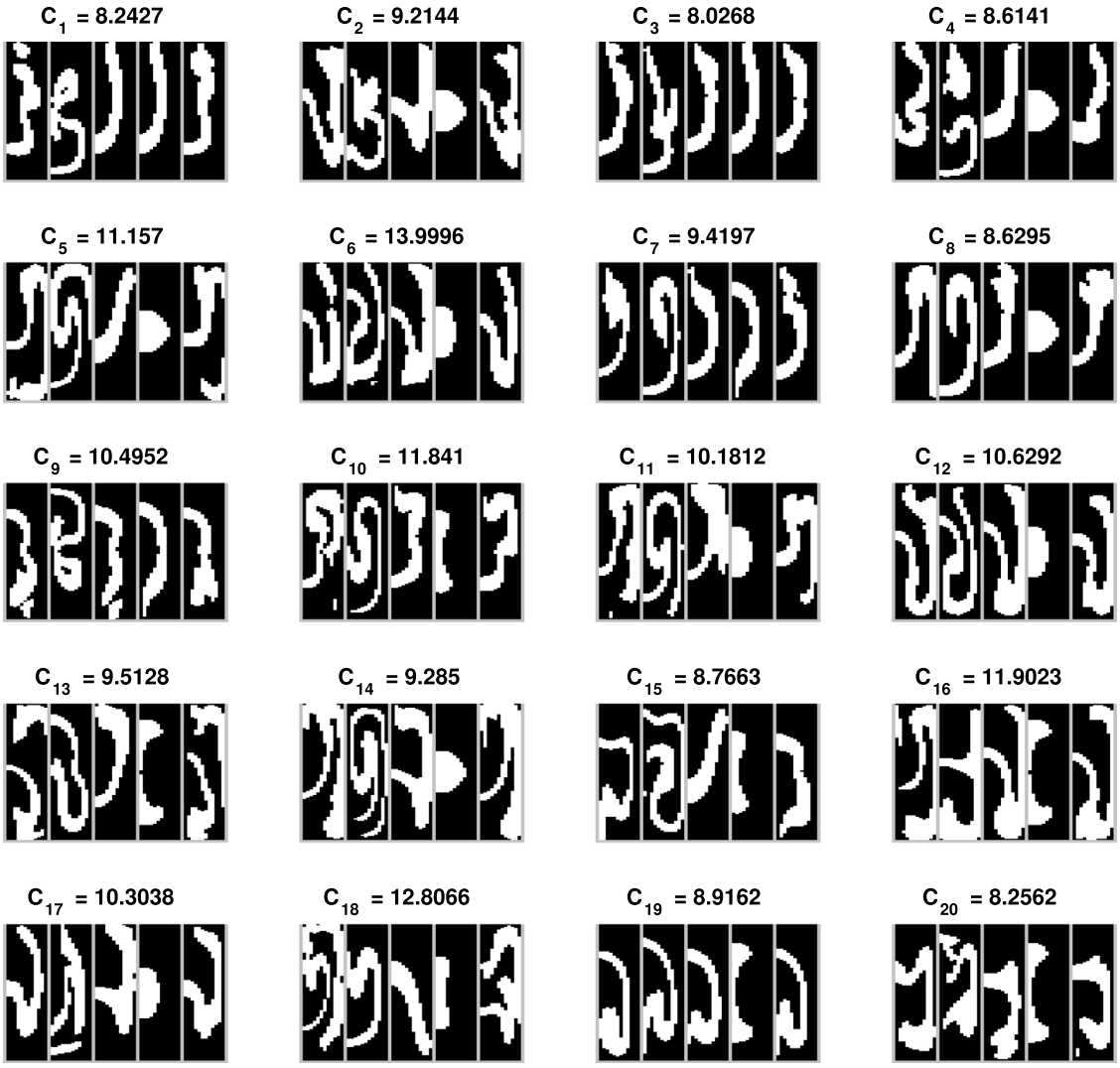


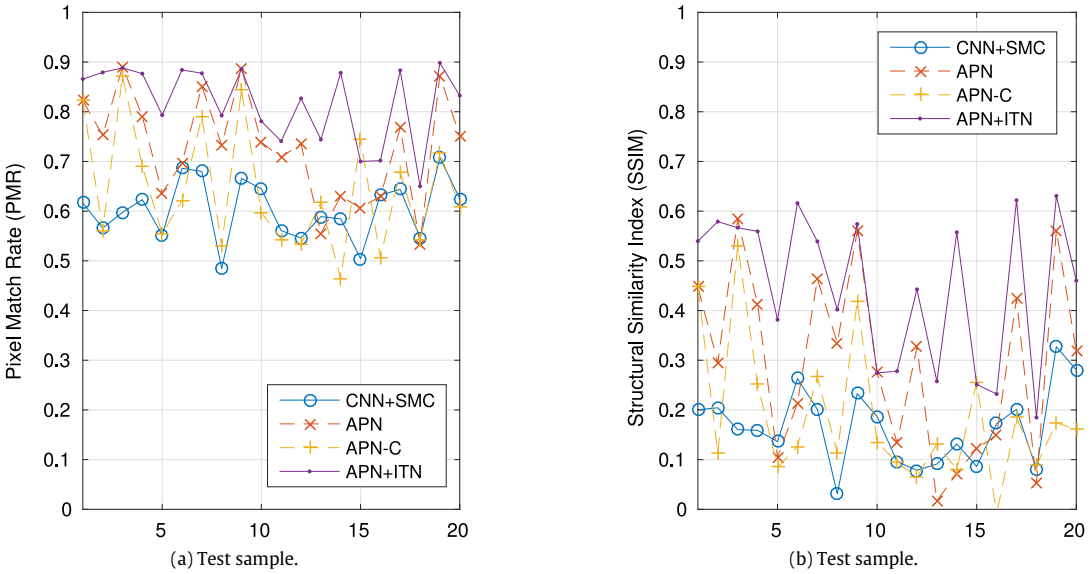
Fig. 12. 20 test shapes with reconstructed flow shapes generated from sequences predicted using different methods. From left to right, the first sub-column is the desired flow shape (ground truth). The second sub-column is the flow shape reconstructed from the predicted sequence using CNN+SMC, the third with APN, the fourth with APN-C, and the fifth with APN+ITN. c denotes the perimetric complexity of the test shapes, which have values of $c > 8.0$.

Insights from filter visualization: Fig. 15 offers insights on the learned features in both the APN and the ITN. Panel (a) shows a random selection of kernels from the first convolutional layer with patterns suggestive of an effective learning of geometric edges of the flow shapes. When combined with deeper layers, the filters jointly attribute the geometrical transformation to a single action (i.e., pillar). Panel (b) shows the weights of the first fully connected layer in the trained ITN model. Patterns learned by the first 60 neurons are visualized. Recall that the input $\mathcal{D}_{input}$ is a concatenation of flow shapes denoted by $f(s_1) \oplus f(s_1 \oplus s_2 \oplus s_3)$. Hence, the right portion of every subimage is the post-deformed shape which is more complex compared to the pre-deformed shape on its left. With a closer look at the weights, we observe that the ITN is able to learn key flow features circled in red. Furthermore, the relatively higher clarity of such features seen in the right portion (as opposed to the left portion) suggests that the post-deformed shape plays a more important role in the determination of the corresponding waypoint between the two flow shapes.

A clear advantage of using both APN and ITN together is that the model does not need to be retrained for variable sequence lengths, unlike in the CNN+SMC model where the number of pillars in the output sequence is constrained. This method is highly scalable, and has an enormous room for extension into sculpting more highly complex flow shapes.

Contrast with RNNs: There is a trade-off between model complexity and error minimization. An RNN will require a more complex model to fine-tune weights specific to a particular time step (i.e. a pillar in the present case). Additionally, a sufficiently large and complex recurrent model may succumb to the problem of vanishing gradients during back-propagation (Hochreiter, 1998; Pascanu, Mikolov, & Bengio, 2013) which warrants the use of model with even higher training complexity such as LSTMs (Jozefowicz, Zaremba, & Sutskever, 2015). Furthermore, the RNN and the current framework are fundamentally different in architecture: RNNs are parallel while the present framework can be thought to be arranged in series. RNNs are meant for sequence-to-sequence predictions, in addition to being limited by a fixed-length sequence in its most basic implementation. The sequence can be of variable length but that requires considerable modifications to the model (such as padding inputs and masking neurons). On the other hand, the APN+ITN framework can predict an arbitrary number of sequences between two end states and prune redundant elements within a sequence. Having two separate modules – each having functional specialization – can actually be more intuitive and straightforward for engineers to quickly adopt and prototype solutions.

Unfortunately, unlike RNNs, large errors may accumulate due to the serial nature of the framework. With this framework, it is



(a) Test sample. (b) Test sample.

Fig. 13. Sample-wise (a) PMR and (b) SSIM comparison using CNN+SMC, APN, APN-C, and APN+ITN. 20 test target shapes are randomly generated from sequences such that the shape complexity $\mathcal{C} > 8.0$.

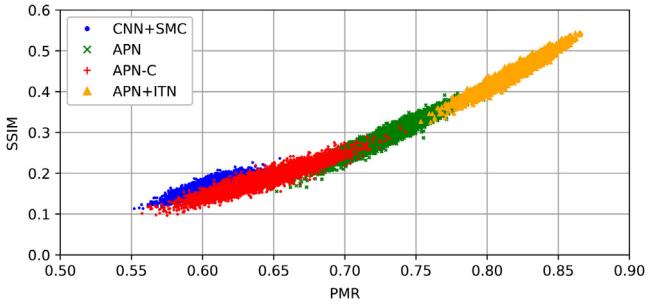
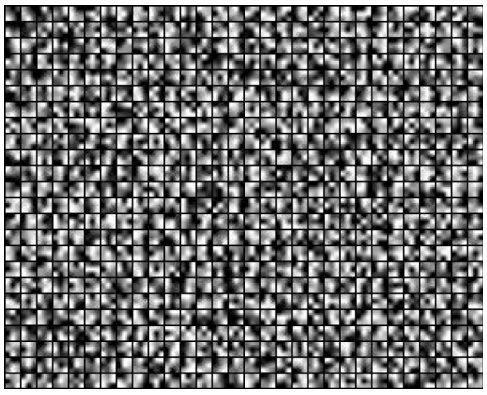


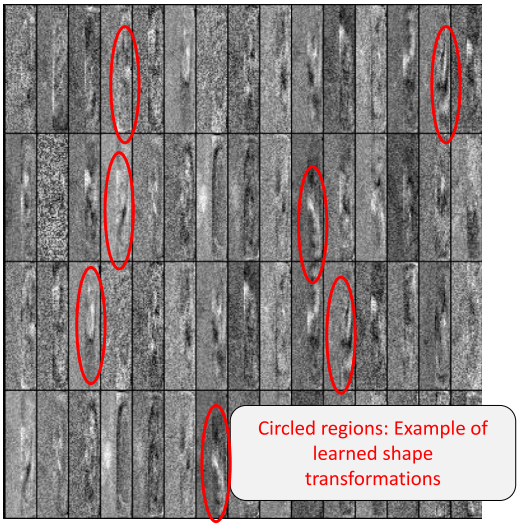
Fig. 14. Scatterplot of PMR and SSIM showing the densities over 5000 test samples with $\mathcal{C} > 8.0$. With the previous CNN+SMC approach, SSIM and PMR is generally lower. With the new formulation with APN all metrics are reporting a significant improvement in similarity between reconstructed flow shapes.

currently not possible to back-propagate this accumulated error for enhancing the learning process. The problem is mostly mitigated by the goal that the sub-sequence between two intermediate shapes should be modified iteratively to reduce the error between the reconstructed shapes versus the target intermediate shapes. As a result, errors accumulated from the present sub-sequence can be taken care of by the next sub-sequence prediction phase as it attempts to generate future sub-sequence (of variable length) that corrects this previously accumulated error. Hence, it is essential to generate substantial amount of forward simulation data (which is computationally inexpensive) for training in order to learn an accurate inverse model. Now, the only important task is to learn, and adjust, the weights of each of the module such that (i) the APN can output the corresponding action reliably given two states, and (ii) the ITN can generate intermediate shapes reliably.

Usage of GPU: Solving the inverse design problem involves mapping a large amount of data high-dimensional image data to its target labels. Although the APN+ITN framework consists of serialized modules that cannot be parallelized, learning an accurate mapping can be greatly accelerated by the use of GPUs during training time. The multiple processing cores in the GPU allows for quick computation and error back-propagation over multiple batches of training examples for the adjustment of weights in the neural network (Bergstra et al., 2010; Jia et al., 2014; Ngiam,



(a) APN Conv1 filters.



(b) ITN Dense1 filters.

Fig. 15. Visualization of (a) filter weights from the first convolutional layer in the APN, and (b) weights of the dense layer in the ITN.

Coates et al., 2011; Raina, Madhavan, & Ng, 2009). During test time, inference may be carried out on the CPU where each feed-forward operation through the neural network is in the order of milliseconds (i.e., approximately 30 inference steps per second).

6. Applicability in other domains

We consider the approach as a fruitful one based on the results shown in Section 5.2. Previously, we have briefly discussed how the proposed method can be used for other types of problems in Section 1. With the same motivation, we now describe the aforementioned possible application areas with more details. The framework can be used in:

1. Learning to transform the belief space for robotic path planning. A robot may be chasing a mobile target while maintaining a posterior (that is transformed into a visual representation analogous to the flow shape) corresponding to the location of the target. If we would like to specifically maximize posterior in a certain region, the corresponding problem would be: "What is the best course of action that should be taken by the robot in succession?".
2. Learning the material processing pathways to obtain the desired microstructures. In materials processing, processing the materials by altering the properties in succession will alter the morphology (microstructures) of the material. The equivalent inverse problem would be: "If we want the material to achieve a specific morphology, what are the processing steps that should be taken?". Additionally, the framework can be used in learning a sequence of material treatment steps that will result in different sets of desired material properties within multiple localized regions of the material.
3. Learning a sequence of manufacturing steps in additive/subtractive manufacturing (e.g., given a stock part and a desired machined part), with *fast-design* being the main advantage.

7. Conclusions and future work

In sequence prediction problems where there are no dependencies within the hidden domain, RNN-like structures may not be a suitable approach. This paper proposes a hybrid framework using convolutional neural networks (CNN) and stacked autoencoders (SAE) to learn a sequence of causal actions that nonlinearly transform an input visual pattern or distribution into a target visual pattern or distribution with the same support. We applied the framework in a complex high-dimensional design exploration problem in the design of microfluidic channels for flow sculpting. The proposed architecture is able to learn a sequence of actions that carries out the desired transformation over the input. The creative integration of the two architectures can tackle the inverse fluid problem and achieve the required design accuracy while expediting the design process. Some of the future efforts are delineated below:

1. Tool-chain optimization such as training data selection for ITN;
2. Comprehensive efforts on the generalization of the waypoint-determination process;
3. Theoretical analysis of the framework in guaranteeing optimal performance.

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GeForce GTX TITAN Black GPU used for this research.

References

Akintayo, A., Lore, K. G., Sarkar, S., & Sarkar, S. (2016). Prognostics of combustion instabilities from hi-speed flame video using a deep convolutional selective autoencoder. *International Journal of Prognostics and Health Management*, *7*.

Amini, H., Sollier, E., Masaeli, M., Xie, Y., Ganapathysubramanian, B., Stone, H. A., et al. (2013). Engineering fluid flow using sequenced microstructures. *Nature Communications*.

Ashforth-Frost, S., Fontama, V. N., Jambunathan, K., & Hartle, S. L. (1995). The role of neural networks in fluid mechanics and heat transfer. In *Instrumentation and measurement technology conference, 1995. IMTC/95. proceedings. Integrating intelligent instrumentation and control, IEEE* (p. 6). IEEE.

Attneave, F., & Arnoult, M. D. (1956). The quantitative study of shape and pattern perception. *Psychological Bulletin*, *53*(6), 452.

Baymani, M., Kerayechian, A., & Effati, S. (2010). Artificial neural networks approach for solving stokes problem. *Applied Mathematics*, *1*(04), 288.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., & Desjardins, G. et al., (2010). Theano: A CPU and GPU math compiler in Python. In *Proc. 9th python in science conf* (pp. 1–7).

Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., & Tobin, J. et al., (2016). Transfer from simulation to real world through learning deep inverse dynamics model, arXiv preprint arXiv:1610.03518.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*(Jan), 1–30.

Dryden, I. L., & Mardia, K. V. (1998). *Statistical shape analysis, Vol. 4*. J. Wiley Chichester.

Duriez, T., Parezanović, V., Cordier, L., Noack, B. R., Delville, J., & Bonnet, J.-P. et al., (2014). Closed-loop turbulence control using machine learning, arXiv preprint arXiv:1404.4589.

Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning* (pp. 49–58).

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., & Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research (JMLR)*, *13*(Mar), 723–773.

Hadsell, R., Erkan, A., Sermanet, P., Scoffier, M., Muller, U., & LeCun, Y. (2008). Deep belief net learning in a long-range vision system for autonomous off-road driving. In *IEEE/RSJ international conference on intelligent robots and systems, 2008*. IROS 2008, (pp. 628–633). IEEE.

Hausdorff, F. (1918). Dimension und äußeres Maß. *Mathematische Annalen*, *79*(1–2), 157–179.

Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., et al. (2017). Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, *35*, 18–31.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107–116.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hwang, C. M., Khademhosseini, A., Park, Y., Sun, K., & Lee, S.-H. (2008). Microfluidic chip-based fabrication of PLGA microfiber scaffolds for tissue engineering. *Langmuir*, *24*(13), 6845–6851.

Jaderberg, M., Simonyan, K., Zisserman, A., et al. (2015). Spatial transformer networks. In *Advances in neural information processing systems* (pp. 2008–2016).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on multimedia* (pp. 675–678). ACM.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd international conference on machine learning* (pp. 2342–2350).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems, Vol. 25* (pp. 1097–1105). Curran Associates, Inc.

Larochelle, H., & Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th international conference on machine learning* (pp. 536–543). ACM.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., & Quillen, D. (2016). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 0278364917710318.

Lore, K. G., Akintayo, A., & Sarkar, S. (2017). LLNet: A deep autoencoder approach to natural low-light image enhancement. *Pattern Recognition*, *61*, 650–662.

Lore, K. G., Stoecklein, D., Davies, M., Ganapathysubramanian, B., & Sarkar, S. (2015). Hierarchical feature extraction for efficient design of microfluidic flow patterns. In *Proceedings of the 1st international workshop on feature extraction: Modern questions and challenges, NIPS* (pp. 213–225).

Lore, K. G., Sweet, N., Kumar, K., Ahmed, N., & Sarkar, S. (2016). Deep value of information estimators for collaborative human-machine information gathering. In *International conference on cyber-physical systems*. Vienna, Austria.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech, Vol. 2* (p. 3).

Müller, S. D., Milano, M., & Koumoutsakos, P. (1999). Application of machine learning algorithms to flow modeling and optimization.

Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Le, Q. V., & Ng, A. Y. (2011). On optimization methods for deep learning. In *Proceedings of the 28th international conference on machine learning* (pp. 265–272).

Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., & Ng, A. Y. (2011). Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning* (pp. 689–696).

Nunes, J. K., Wu, C. Y., Amini, H., Owsley, K., Di Carlo, D., & Stone, H. A. (2014). Fabricating shaped microfibers with inertial microfluidics. *Advanced Materials*, *26*, 3712–3717.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Paulsen, K. S., & Chung, A. J. (2016). Non-spherical particle generation from 4D optofluidic fabrication. *Lab Chip*.

Paulsen, K. S., Di Carlo, D., & Chung, A. J. (2015). Optofluidic fabrication for 3D-shaped particles. *Nature Communications*, *6*, 6976. http://dx.doi.org/10.1038/ncomms7976.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, *1*(2), 263–269.

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning* (pp. 873–880). ACM.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779–788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Sarkar, S., Lore, K. G., & Sarkar, S. (2015). Early detection of combustion instability by neural-symbolic analysis on hi-speed video. In *Workshop on cognitive computation: Integrating neural and symbolic approaches (CoCo@ NIPS 2015)*, Montreal, Canada.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Sollier, E., Amini, H., Go, D. E., Sandoz, P. a., Owsley, K., & Di Carlo, D. (2015). Inertial microfluidic programming of microparticle-laden flows for solution transfer around cells and particles. *Microfluidics and Nanofluidics*, *19*, 53–65. http://dx.doi.org/10.1007/s10404-015-1547-7.

Srivastava, N., & Salakhutdinov, R. R. (2012). Multimodal learning with deep Boltzmann machines. In *Advances in neural information processing systems* (pp. 2222–2230).

Stoecklein, D., Lore, K. G., Davies, M., Sarkar, S., & Ganapathysubramanian, B. (2017). Deep learning for flow sculpting: Insights into efficient learning using scientific simulation data. *Scientific Reports*, *7*, srep46368.

Stoecklein, D., Wu, C.-Y., Kim, D., Di Carlo, D., & Ganapathysubramanian, B. (2016). Optimization of micropillar sequences for fluid flow sculpting. *Physics of Fluids*, *28*. arXiv:1506.01111.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103). ACM.

Wada, Y., & Kawato, M. (1993). A neural network model for arm trajectory formation using forward and inverse dynamics models. *Neural Networks*, *6*(7), 919–932.

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612.

Wodo, O., Zola, J., Pokuri, B. S. S., Du, P., & Ganapathysubramanian, B. (2015). Automated, high throughput exploration of process–structure–property relationships using the MapReduce paradigm. *Materials Discovery*.

Wu, C.-Y., Owsley, K., & Di Carlo, D. (2015). Rapid software-based design and optical transient liquid molding of microparticles. *Advanced Materials*, *27*, 7970–7978. http://dx.doi.org/10.1002/adma.201503308.

Zhang, T., Kahn, G., Levine, S., & Abbeel, P. (2016). Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *IEEE international conference on robotics and automation*.