Title:
Accuracy Assessment and Reinforcement Training of Deep Neural Networks in the Context of Accelerating a Simulation-Based Toolpath Generation Method

Authors:
Tathagata Chakraborty, tathagata.chakr@hcl-software.com, HCLTech
Christine Zuzart, christine.zuzart@hcl-software.com, HCLTech
Chinmaya Panda, chinmay.panda@hcl-software.com, HCLTech
Nitin Umap, nitin.umap@hcl-software.com, HCLTech

Keywords:
Deep Learning, Simulation, Toolpath Generation, CNN, Accuracy Assessment

DOI: 10.14733/cadconfP.2024.6-12

Introduction:
Simulation-driven techniques in science and engineering are now becoming viable with the increasing availability and affordability of computation power. In Computer-Aided Manufacturing, simulation-based methods can be thought of as low-code alternatives to traditional computational geometry-based approaches for generating high-speed toolpaths. Over the long term, the cost of additional computing power is more than offset by the cost savings of not having to maintain complex software [1]. However, simulation-based techniques are still too compute-intensive for the average computer and for interactive use. In this paper, we show how deep neural networks (DNNs) can significantly improve the performance of a simulation-based method for computing high-speed 2.5D toolpaths.

Despite the success of DNNs in many fields, it is still challenging to use DNNs in domains with little or no margin of error. This is because DNNs generally give over-confident results (in classification problems) and do not directly report any measure of confidence in their output (in regression) [3]. The lack of an accuracy measure often restricts their use to error-tolerant domains. In toolpath generation, the margin of error must be very low, and using DNNs in CNC machining requires robust uncertainty estimation. In this paper, we also present a novel method for assessing the accuracy of the results predicted by a DNN that adds a suitable inductive bias when training the model. Uncertainty estimation also enables us to identify model failures and retrain the model on the failure cases for improved accuracy in a reinforcement learning loop.

Simulation-based Toolpath Generation:
Deep learning methods are increasingly being used to accelerate simulations in many fields. DNNs today accelerate fluid flow simulations [12], seismic wave simulations [11], material and micro-structure simulation [14, 10], and molecular system dynamics [16]. The methods vary widely in how and where they are applied in the process. However, learning-based methods are increasingly gaining acceptance

[1] In the case of desktop software, the user bears the additional computation cost, and the cost savings from not having to maintain complex code accrue to the software developer. Eventually, however, the cost savings get passed down to the users directly and indirectly via a more robust application.

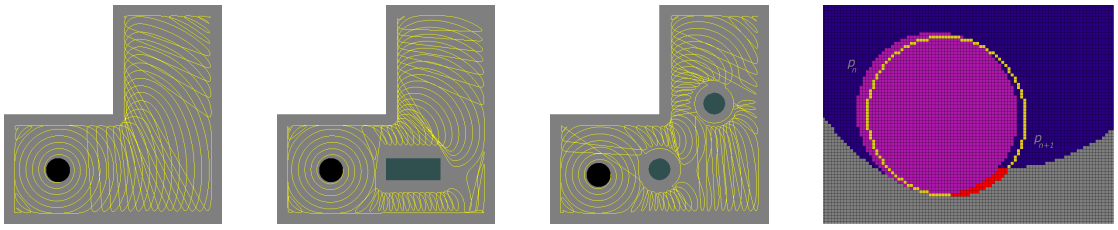

Fig. 1: Post-processed toolpaths for an L-shaped pocket with islands using the method described in [2]. Extreme Right: Pixel-based tool engagement computation.

into scientific computing, where, until recently, such techniques were eschewed for their uncertain nature and lack of accuracy.

In an earlier paper [2], we described a simulation-driven boundary-following algorithm for generating high-speed toolpaths for 2.5D pockets (see Fig. 1 for some results). The technique was a low-code and easy-to-implement approach for computing high-speed curvilinear toolpaths. However, the method was computationally too intensive for interactive applications. Some simulation-based methods, like ours, are difficult to scale out since one only has a single simulation environment where multiple actions cannot be simultaneously explored.

In this paper, we extend the work done in [2] by accelerating the computationally intensive part of the process using a convolutional neural network (CNN). In [2], we compute the toolpath incrementally as a series of small, fixed-length segments. The orientation of a particular segment is computed based on the direction of the previous segment and the tool engagement value computed at the current segment. To determine the next movement direction, the tool engagement values at several orientations are calculated, and the one closest to the desired engagement is chosen. Computing the tool engagement value involves capturing two copies of a small region of the framebuffer in the neighborhood of the tool and calculating the difference in the count of pocket pixels in these two copies (see Fig. 1 (extreme right)). Synchronization requirements between the CPU and the GPU bottleneck this part of the process.

One way to speed up the simulation process is to reduce the number of orientations evaluated at each step. This can be done by analyzing the tool's neighborhood and estimating the most likely direction for tool movement. However, implementing a conventional algorithm to solve this problem can be tedious and error-prone since one must account for numerous possible neighborhood configurations. Today, such problems, that are difficult to solve analytically but where there is sufficient amount of labeled data available, are best solved using a DNN.

Over-confidence and Accuracy Assessment of DNNs:

Modern DNNs have a large number of parameters, which increases prediction accuracy but may simultaneously pressurize the network to memorize the data. Various regularization techniques are therefore incorporated into the network so that the network generalizes to unseen input data. Regularization penalizes complex models and reduces overconfident predictions by discouraging overly complex decision boundaries. However, getting the amount of regularization just right can be challenging. It is often impractical to train a DNN on all possible input cases. Training the network to recognize all types of adversarial input is also tricky. A large amount of data is also challenging to obtain and adversely affects the training time. Due to all this, DNNs are rarely properly calibrated [5]. Additionally, because of how the cross-entropy loss is computed in classification models, the network always makes very confident predictions [1].

Accuracy assessment of the predictions from a DNN (or any machine learning model for that matter)

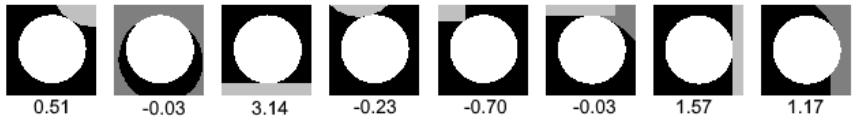

Fig. 2: A random sample of input training images (66x66 pixels) with the target tool movement angles.

is a long-standing open problem, with much ongoing research [5, 6]. For classification problems, post-processing techniques where the penultimate layers of the network are analyzed can be used [5, 9]. The network can also be trained to recognize outliers and report them during inference [4]. A more generic technique is to use an ensemble of models, where predictions from multiple networks are averaged and analyzed. Ensemble methods can be used to determine the accuracy of both classification and regression models [7, 8].

In this paper, we propose a novel method for determining the accuracy of regression models in simulation-based techniques. We note that this technique can be used only for a limited class of problems, where additional side-effects of the simulation can be used to add inductive bias to the model. Our technique is tangentially related to some ideas from physics-guided machine learning [13] and physics-informed neural networks [15].

Generating the Training Data:

The data for training the DNN is generated using the brute-force simulation method described in [2]. As much data as required for training a DNN can be generated by running the brute-force simulation method on different pocket shapes and tools. For the results reported here, we trained the CNN model (see Fig. 3) on approximately $25k$ input images. An addition $5k$ images were used as a test or validation set. In the brute-force method, we capture a small region in the neighborhood of the tool and determine the next tool movement angle by computing the tool engagement along several different directions. We generate the input data by saving the captured neighborhood region as an image. For the target data, we compute the angle that the determined tool direction makes with the x-axis. A set of captured input images is shown in Fig. 2 along with the target angles. In these images, the tool is rendered in white, the area outside the pocket is in light gray, the remaining material is in a darker shade of gray, and the already cut material is colored black.

Models and Performance

A simulation-based toolpath generation strategy involves an agent and an environment and required balancing exploration with exploitation. One may, therefore, advocate the use of a reinforcement learning algorithm for this case. However, for this study, we already have a simulation-driven strategy developed in [2], and the aim is to accelerate the computation rather than discover a toolpath strategy from scratch. Since our training data consists of images, it immediately suggests using CNNs. However, our input data (see Fig. 2) occupies a much lower-dimensional space within the domain of images, so even a fully connected network is likely to perform equally well. Alternately, vision transformers can also be used, but they are both slow and difficult to train, so we ignored them after some initial testing.

Fig. 3 (right) shows the performance of a fully connected (FC) model and a CNN model (the architecture of which is shown in Fig. 3 (left)). At $\pm 10^\circ$ tolerance (we execute the brute-force method within this range to determine the final move direction), both the FC and CNN models have nearly the same accuracy. However, at 2.36 million, the FC model requires more than an order of magnitude of parameters to reach the same level of performance as the CNN model with only 144K parameters. The

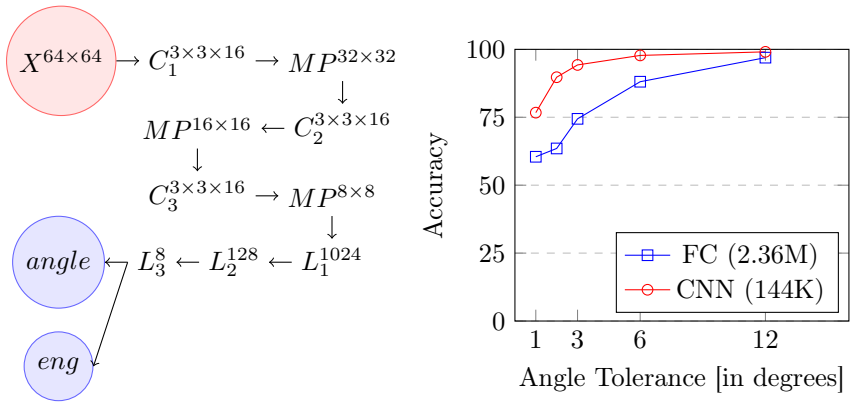

Fig. 3: Left: CNN model architecture. Right: Accuracy of the models at different angular tolerances.

results reported in this paper were all generated using the CNN model shown in Fig. 3 (left).

Accuracy Assessment and Results:

Existing techniques for computing a confidence score for neural networks are all generic methods applicable irrespective of the problem domain. We show that for a certain class of problems, there are potentially more robust methods for estimating the confidence of a neural network.

Our method consists of training the network to predict one or more additional target parameters (see Fig. 3 (left), where the last layer of the CNN outputs two target values). During training, the sum of the loss across all the targets is minimized. During inference, the system takes action based on the primary target value predicted by the model. It then compares the computed additional values with the predicted ones to determine the accuracy.

The novelty of our method lies in counter-intuitively training the neural network to predict these additional values. For toolpath generation, the primary target is the tool movement angle, and the additional target is the engagement value. Apart from helping us determine the accuracy of the prediction, an additional value, like the potential tool engagement, adds an inductive bias to the model. A physics-based inductive bias can guide the optimization of the network in the right direction, ensuring that the network learns the right abstractions. Other additional parameters like the previous tool move direction can also be added.

Fig. 4 shows the resulting toolpaths for some scenarios – when using the CNN model without any fallback and tolerance (left), with fallback and $\pm10°$ tolerance (middle), and with accuracy measurement and fallback (right). As can be seen in Fig. 4 (left), if we only use the angle predicted by the CNN model, the model fails in many cases (the $\pm1°$ tolerance accuracy of the model is only around 77%). Sometimes, the model's errors in one place get magnified later when the tool moves over the previously cut section and encounters regions far from what it has seen in training. With fallback and tolerance (see Fig. 4 (middle)), the algorithm evaluates a $\pm10°$ range centered on the direction predicted by the CNN model and chooses the best direction from among these. In most cases, this approach results in an almost usable toolpath. However, sometimes, such a toolpath can have some sharp corners and kinks where the correct direction may fall just outside the tolerance range or is clearly wrong (see inset Fig. 4 (middle)).

No easy methods exist to identify these errors without some form of accuracy assessment. With accuracy assessment, we compare the engagement value predicted by the network with that computed when we move in the general direction predicted by the network. Suppose there is a significant difference

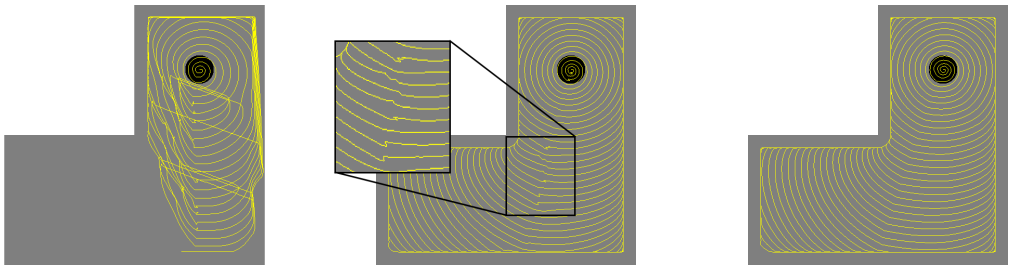

Fig. 4: Toolpaths generated using predictions from a DNN - without any tolerance (left), with tolerance (middle), with accuracy assessment and fallback (right).

in the predicted and computed engagement values (if the predicted values are assumed to distributed normally around the actual value, then, for example, a value that is more than two standard deviation away can be considered significant). In that case, we reject the prediction and fallback on the brute-force method where a much larger range of angles $\pm 90°$ is evaluated. Algorithm 1 shows an outline of this method, where the function $PredictDNN$ evaluates the CNN model and the $FindBestMove$ function is the brute-force function that evaluates all the angles in a given range centered around a given direction, both functions returning an angle and tool engagement pair of values. The identified failure cases can be stored and used to retrain the model in a reinforcement learning loop.

Algorithm 1: Assessing the accuracy of the move direction predicted by the CNN

Input: σ `// standard deviation of the engagement value difference`
ctx `// the context neighborhood image input`
$P_{tp} = [p_1, p_2, \cdots, p_n]$ `// points comprising the toolpath`

Function `FindNextMove(`ctx, P_{tp}`):`
  $(\theta_{pred}, \varepsilon_{pred}) \leftarrow PredictDNN(ctx)$
  $\vec{d} \leftarrow (\sin\theta_{pred}, \cos\theta_{pred})$ `// direction predicted by the CNN model`
  $(\theta_{best}, \varepsilon_{best}) \leftarrow FindBestMove(P_{tp}, \vec{d}, \pm 10°)$
  if $|\varepsilon_{pred} - \varepsilon_{best}| > 2\sigma$ **then**
    $\vec{d} \leftarrow p_n - p_{n-1}$ `// direction of the last toolpath segment`
    $(\theta_{best}, \varepsilon_{best}) \leftarrow FindBestMove(P_{tp}, \vec{d}, \pm 90°)$
  end
  return θ_{best}

At first glance, our system may seem similar to multi-target DNNs (MT-DNNs) (see [17]). However, while MT-DNNs try to club multiple tasks into the same model, our approach trains the model for a single task but adds multiple redundant targets. Also, in MT-DNNs, the additional targets are not actionable, and there is no attempt to measure the accuracy of such systems using these additional target values.

Performance Improvements:

In the brute-force version of the simulation-based method as described in [2], each step in the process required, on average, the evaluation of around 180 different angles (from $-90°$ to $90°$, in $1°$ increments). It is possible to reduce this number by adding various heuristics to break out early from the evaluation as soon as a likely direction is found. It is also possible to evaluate the full angle range progressively.

However, this increases the number of edge cases that must be specially handled and complicates an otherwise simple algorithm.

With a DNN, we can reduce the number of evaluated angles by 9 fold to the range $-10°$ to $10°$ centered around the direction predicted by the DNN. The fully connected and the CNN models both gave over 95% accuracy on the test data when the accuracy was evaluated in the $\pm 10°$ range (see Fig. 3). To the net time taken, we must add the model inference time. However, in practice, the inference time is negligible compared to the time taken to copy the framebuffer regions from the GPU to the CPU. The inference overhead is small because, although the models contain many parameters, the computation is carried out in a highly vectorized form in both CPU and GPU.

Conclusions:

In this paper, we have suggested a novel method for adding inductive bias to a neural network model by having it predict additional target parameters. We have shown that for simulation-based methods where backtracking at least one step is possible, one can measure the accuracy of the network prediction by comparing the simulation-generated value of the additional parameter with that predicted by the network. This approach makes it possible to use DNNs for scientific and engineering simulations where high accuracy is expected at each and every step of the process.

We have also shown how a simulation-driven toolpath generation algorithm can be accelerated by over an order of magnitude by narrowing the range of the angles explored using a DNN. Even with these measures, there is still a possibility where the DNN predicts a wrong direction of movement but an engagement value valid for the incorrectly predicted direction. The overall algorithm must be designed so that it is possible to detect such worst-case problems. We, therefore, additionally perform gouge checks and check for toolpath smoothness in regions where tool engagement is high and near constant.

Acknowledgement:

This research was conducted as part of the CAMWorks project. CAMWorks is a popular CAM software used by small and mid-sized machining workshops worldwide. We want to thank everyone in the CAMWorks team, past and present, who have made this research possible.

Tathagata Chakraborty, https://orcid.org/0000-0002-2752-2533
Christine Zuzart, https://orcid.org/0009-0005-3128-4418
Chinmaya Panda, https://orcid.org/0009-0004-6096-9582
Nitin Umap, https://orcid.org/0000-0002-9063-1230

References:

[1] Bai, Y.; Mei, S.; Wang, H.; Xiong, C.: Don't just blame over-parametrization for over-confidence: Theoretical analysis of calibration in binary classification, International Conference on Machine Learning, PMLR, 2021 Jul 1, 566-576.

[2] Chakraborty, T.; Panda, C.; Umap, N.: Simulation-Driven Computation of High-Speed Pocket Machining Toolpaths, Computer-Aided Design, 21(1), 2024, 88-103. https://doi.org/10.14733/cadaps.2024.88-103

[3] Gawlikowski, J.; Tassi, C.R.; Ali, M.; Lee, J.; Humt, M.; Feng, J.; Kruspe, A.; Triebel, R.; Jung, P.; Roscher, R.; Shahzad, M.: A survey of uncertainty in deep neural networks, Artificial Intelligence Review, 56(Suppl 1), 2023 Oct, 1513-89. https://doi.org/10.1007/s10462-023-10562-9

[4] Grabinski, J.; Gavrikov, P.; Keuper, J.; Keuper, M.: Robust Models are less Over-Confident, Advances in Neural Information Processing Systems, 2022 Dec 6, 35:39059-75.

[5] Guo, C.; Pleiss, G.; Sun, Y.; Weinberger, K.Q.: On calibration of modern neural networks, International conference on machine learning, PMLR, 2017 Jul 17, 1321-1330.

[6] Hendrickx, K.; Perini, L.; Van der Plas, D.; Meert, W.; Davis, J.: Machine learning with a reject option: A survey, arXiv preprint, arXiv:2107.11277, 2021 Jul 23.

[7] Ju, C.; Bibaut, A.; van der Laan, M.: The relative performance of ensemble methods with deep convolutional neural networks for image classification, Journal of Applied Statistics, 45(15), 2018 Nov 18, 2800-18.

[8] Lakshminarayanan, B.; Pritzel, A.; Blundell, C.: Simple and scalable predictive uncertainty estimation using deep ensembles, Advances in neural information processing systems, 30, 2017.

[9] Mandelbaum, A.; Weinshall, D.: Distance-based confidence score for neural network classifiers, arXiv preprint, arXiv:1709.09844, 2017 Sep 28.

[10] Mendizabal, A.; Márquez-Neila, P.; Cotin, S.: Simulation of hyperelastic materials in real-time using deep learning, Medical image analysis, 59:101569, 2020 Jan 1. https://doi.org/10.1016/j.media.2019.101569

[11] Moseley, B.; Nissen-Meyer, T.; Markham, A.: Deep learning for fast simulation of seismic waves in complex media, Solid Earth, 11(4), 2020 Aug 24, 1527-49. https://doi.org/10.5194/se-11-1527-2020

[12] Obiols-Sales, O.; Vishnu, A.; Malaya, N.; Chandramowliswharan, A.: CFDNet: A deep learning-based accelerator for fluid simulations, Proceedings of the 34th ACM international conference on supercomputing, 2020 Jun 29, 1-12. https://doi.org/10.1145/3392717.3392772

[13] Pawar, S.; San, O.; Aksoylu, B.; Rasheed, A.; Kvamsdal, T.: Physics guided machine learning using simplified theories, Physics of Fluids, 33(1), 2021 Jan 1. https://doi.org/10.1063/5.0038929

[14] Peivaste, I.; Siboni, N.H.; Alahyarizadeh, G.; Ghaderi, R.; Svendsen, B.; Raabe, D.; Mianroodi, J.R.: Accelerating phase-field-based simulation via machine learning, arXiv preprint, arXiv:2205.02121, 2022 May 4.

[15] Raissi, M.; Perdikaris, P.; Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics, 378, 2019 Feb 1, 686-707. https://doi.org/10.1016/j.jcp.2018.10.045

[16] Vlachas, P.R.; Zavadlav, J.; Praprotnik, M.; Koumoutsakos, P.: Accelerated simulations of molecular systems through learning of effective dynamics, Journal of Chemical Theory and Computation, 18(1), 2021 Dec 10, 538-49. https://doi.org/10.1021/acs.jctc.1c00809

[17] Zeng, Z.; Liang, N.; Yang, X.; Hoi, S.: Multi-target deep neural networks: Theoretical analysis and implementation, Neurocomputing, 273, 2018 Jan 17, 634-42.