# Predicting Future Performance of Convolutional Neural Networks in Early Training Stages

Yunfeng Xin, Chengzhe Xu, and Hangyi Zhao

**Abstract**—The training process of convolutional neural networks (CNN) can be lengthy, and being able to predict the performance of a certain set of hyperparameters on a specific CNN architecture can save both time and computation cost. This project focuses on predicting the final accuracy of a CNN using only the validation accuracy of the initial 100 epochs. With our best model, we are able to achieve an average error of 0.38%.

**Index Terms**—Neural Networks, Perdiction Methods, Machine Learning, Unsupervised Learning

## 1 Introduction

Convolutional neural networks (CNNs) are widely applied in modern machine learning tasks. However, as CNNs tend to stack numerous convolutional layers and fully connected layers on top of each other, training a CNN is compute-demanding. Under most circumstances, it requires an iterative process of fine-tuning the hyperparameters of each layer and evaluating the performance after the training process converges, which can be highly time-consuming [1].

The validation accuracy in each epoch of the training process forms a curve. If we are able to predict the pattern of the curve, the final validation accuracy can be predicted using only the beginning fraction of the curve. As a result, we won't need to complete the entire training process to evaluate the performance of the model, and can adjust the hyperparameters earlier.

This project aims to reduce the time spent on the iterative fine-tuning process by applying machine learning to predict the final validation accuracy to enable automatic parameter tuning without having to complete the entire lengthy training process. The input to our algorithm is the validation accuracy during the initial 100 epochs of CNN training. We then apply various machine learning techniques such as curve fitting, clustering, nearest and K-nearest neighbor, neural networks, and Markov Chain Monte Carlo (MCMC) to predict the final validation accuracy of the CNN at epoch 150, and compare their performance in predicting the final validation accuracy.

## 2 Related Work

The field of automatic parameter tuning and early performance prediction has been covered in various previous works. Bergstra and Bengio [2] suggested using random search to find a set of neural network architecture with good performance. Eggensperger [3] used bayesian network to accelerate hyperparameter search. However, both processes can be lengthy as they involve completing the entire training process.

Kolachina et al. [4] presented the potential shapes of the learning curve and the parametrized forms in their work in 2012, which provided a foundation for speculative evaluation of the validation curves. Based on his work, Domhan et al. [5] presented ways of predicting the shape of the curve using bayesian networks with a strong prior stating that the curve has a non-decreasing property. However, their work only stated the performance on a single learning curve, and provided the estimated error based on the last point of the data curve, which can be noisy. In addition, bayesian learning can require a good amount of data to learn a good set of parameters and can still be cumbersome for deep neural networks. Klein et al. [6] improved the learning process of the bayesian networks by using sampling and is the state-of-the-art work in this field, but their method requires extensive learning and inference computation.

This project examines alternative algorithms under limited dataset size and try to reduce the amount of training and computation needed. In addition, instead of the last point as a performance evaluation metric and providing the evaluation result only for one curve data as we have seen in Domhan et al.'s [5] work, we use the last three epochs of validation accuracy to provide a better error estimation, and evaluate the performance of the model across a variety of validation accuracy curves..

## 3 Dataset and Features

Our team did not find an existing dataset containing the validation accuracy data during CNN training process that matches our purpose. Thus, we decided to create our own dataset and generated 34 validation accuracy curves from 3 different neural network architectures (two convolution layer CNN, three convolution CNN and Vgg-19). For each neural network architecture, we used different hyperparameters which includes different number of channels, optimizers (SGD/ADAM), and activation layers (ReLU/MAXOUT) to make sure our data covers a good number of possible scenarios. For each setting, we generated three to four validation curves. The dataset containing 34 curve data is randomly split into 22 training curves, 6 validation curves, and 6 test curves.

The acquired validation curves are shown in Fig. 1 and covers a good range of possible curve shapes. However,

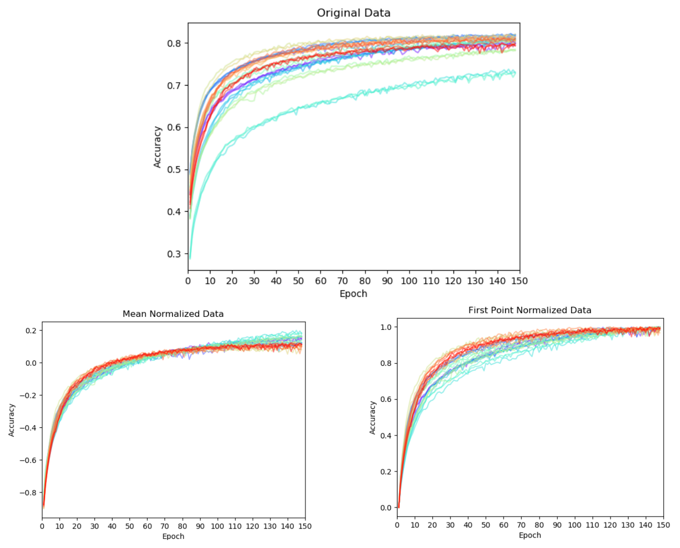

Fig. 1: Plots showing the results of the origin curve data (top), mean-centered normalization (bottom left), and first-point-centered normalization (bottom right).

the variance can pose a challenge for machine learning algorithms. Thus, we also applied normalization techniques to the dataset. The first method subtracts the mean of the validation accuracy in the first 100 epochs from all the values and then scales them by the difference between the maximum and minimum values, which can be represented in the following equation:

$$x'_j = \frac{x_j - \text{mean}_{i=1}^{100} x_i}{\max_i x_i - \min_i x_i} \tag{1}$$

The second method subtracts the first-epoch validation accuracy from all the values and uses the same scaling as the previous method.

$$x'_j = \frac{x_j - x_1}{\max_i x_i - \min_i x_i} \tag{2}$$

The resulting normalized data can be found in the bottom section of Fig. 1. We can clearly see that under our visualization schemes, these data are similarly distributed while retaining their own characteristics. To retrieve the original curve shape, we can simply apply the inverse of Eq. 1 and 2.

## 4 Methods

Our team used five different supervised and unsupervised algorithms to solve the problem, which includes direct curve fitting, (K-)nearest neighbor, clustering, Markov Chain Monte Carlo, and sliding window prediction with a neural network. This section covers the general approach, and detailed execution of these methods will be discussed in the experiments section.

### 4.1 Direct Curve Fitting

We start by using the most intuitive method which does not require any training examples. In direct curve fitting, we try to find the best shape from the pools (Table 1) that fits the first 100 epochs of validation accuracy with the loss function defined as the absolute deviation:

| Curve Name | Parametric Form |
|---|---|
| Vapor Pressure | $\exp(a + \frac{b}{x} + c\log(x))$ |
| Pow3 | $c - ax^{-a}$ |
| Log Log Linear | $\log(a\log(x) + b)$ |
| Log Power | $\frac{a}{1+(\frac{x}{e^b})^c}$ |
| Pow4 | $c - (ax + b)^{-a}$ |
| MMF | $a - \frac{a-b}{1+(kx)^c}$ |
| Exp4 | $c - e^{-ax^\alpha + b}$ |
| Janoschek | $a - (a-b)e^{-kx^c}$ |
| Weibull | $a - (a-b)e^{-(kx)^c}$ |
| Ilog2 | $c - \frac{a}{\log(x)}$ |

TABLE 1: Pool of Potential Curve Shapes.

$$l(f) = \sum_{i=1}^{n}\sum_{j=1}^{100} \left|x_j^i - f_{x^i}(j)\right| \tag{3}$$

The prediction is then obtained by calculating $f(x)$ at the corresponding epochs:

$$\hat{y} = f(x) \textbf{ where } x \in [1, 150] \tag{4}$$

### 4.2 Nearest and K-Nearest Neighbor

The intuition of using nearest neighbor is that the new validation accuracy curve to be predicted resembles the shape of one of the curves in our training set. To determine the nearest neighbor of the curve to be predicted, we need to define a loss function to determine the distance of one curve from another for the first 100 epochs. Here we use the absolute deviation function:

$$\mathcal{L} = \sum_{epoch=1}^{100} |y_{epoch} - \hat{y}_{epoch}| \tag{5}$$

After the nearest neighbor is determined, the algorithm picks the best-performing shape for this neighbor and use that shape to fit our input. To obtain the prediction, we simply generate the extrapolation points of the curve at the last three epochs using Eq. 4.

Our K nearest neighbor algorithm generally follows the same process as described in our nearest neighbor method. However, since we now have one best curve shape for each neighbor, we can have more than one best shapes. To resolve this conflict, we first predict the final validation accuracy using the same process as used in our nearest neighbor algorithm. Each prediction is multiplied by a weight corresponding to the fitting loss $\mathcal{L}$ in Eq. 5. The final prediction can be retrieved by summing up the product of the each prediction $\hat{y}$ with their corresponding normalized weight as shown in Eq. 6.

$$\hat{y}_{weighted} = \sum_{k=1}^{n} \frac{e^{-\mathcal{L}_k}}{\sum_{j=1}^{n} e^{-\mathcal{L}_j}} \cdot \hat{y}_k \tag{6}$$

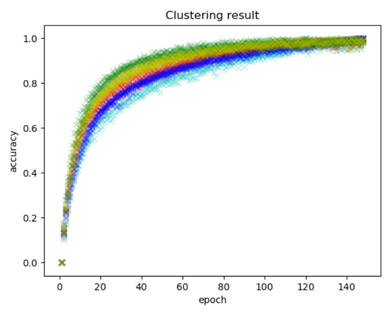

Fig. 2: Plots showing the results of an example of a 5-center clustering result.

### 4.3 Clustering

Nearest and K-Nearest Neighbor algorithms stated in Section 4.2 are facing the main problem of intensive needs of data storage, which would be more obvious when dealing with larger datasets and curve-fitting tasks. To relax the data storage requirement, we introduce the K-Means clustering algorithm to curve fitting tasks.

We assume that each curve is generated by sampling from time-series probabilistic distributions centered at corresponding deterministic curve functions. (The details of this assumption are discussed in Section 4.4.) Under this assumption, the variances of data points come from the differences in basic curve functions and sampling noises. When sampling noises become larger than or equal to the differences between basic curve functions, it is reasonable to merge the data points into one cluster and fit curves functions to the whole cluster.

We first apply mean-centered or point-centered normalization to the data points, and then cluster the data by Euclidean distance:

$$\mathcal{D} = \frac{1}{2}\sum_{epoch=1}^{150}\left(y_{1,epoch} - y_{2,epoch}\right)^2 \tag{7}$$

Fig. 2 shows an example result after 5-center clustering. The rest of the clustering-based algorithms and evaluation methods are similar to Nearest and K-Nearest Neighbor in Section 4.2.

### 4.4 Markov Chain Monte Carlo

As mentioned in Section 4.3, we can assume that accuracy curves are generated by sampling from time-series probability distributions. In that case, the accuracy $y$ at time step *epoch* should be sampled from the conditional distribution:

$$y_{epoch} \sim \mathcal{P}_{epoch}\left(y_{epoch}|y_{epoch-1}, ..., y_1; \theta_{epoch}\right) \tag{8}$$

Where $\theta_{epoch}$ is the parameter vector for distribution $\mathcal{P}_{epoch}$. Curve fitting based algorithms omit the dependance among time-series variables. To take that into account, we use sampling-based algorithms to learn the model parameters under first-order Markov condition:

$$\begin{aligned}\mathcal{P}_{epoch}\left(y_{epoch}|y_{epoch-1}, y_{epoch-2}, ..., y_1; \theta_{epoch}\right)\\ = \mathcal{P}_{epoch}\left(y_{epoch}|y_{epoch-1}; \theta_{epoch}\right)\end{aligned} \tag{9}$$

We apply Markov Chain Monte Carlo (MCMC) [7] [8] [9] algorithm and Metropolis-Hastings (M-H) [10] [11] sampling for sample extraction to maximize the parameters' posterior probability (MAP) and to reconstruct curve models.

We used Gaussian distribution to model the prior distribution of the parameters and we assumed that the accuracy data are sampled from a Gaussian distribution centered at the model's prediction, in other words, we regarded sampling noises as white noises. After fitting models using MCMC sampling, we follow the same procedure to make predictions and evaluate the performance as stated in Section 4.2.

### 4.5 Sliding Window Generation using Neural Network

Intuitively, if we can capture the pattern in the change rate of the curve, we can generate an approximation of the curve using a sliding window. Specifically in our algorithm, we iteratively generate the accuracy in epoch $N$ using validation accuracy from epoch $N - 70$ to $N - 1(N \in [101, 150])$.

The challenge then falls onto how we can efficiently capture the change rate pattern within the curve itself. To achieve this, we used a three-layer neural network [12] as specified in Fig. 3.

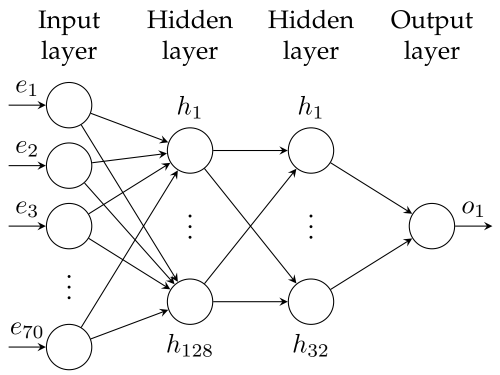

Fig. 3: Neural network architecture for sliding window prediction

We only use data from epoch 30 to 150 in our training set as the initial 100 epochs of data is already known. Notice that for each training curve, we can further split it into 50 training data since each point from epoch 100 to epoch 150 can be the labelled target and the previous 70 points will be the input.

To prevent overfitting, we have added dropout layers between every two consecutive layers and set the dropout rate to be 0.3. In addition, we also used early stopping and terminated the training after 15 epochs.

After training, the neural network is used to iteratively generate the prediction from epoch 101 to epoch 150 using a combination of the inputs and the generated predictions in a sliding window manner.

## 5 Results and Analysis

The results of our experiments are summarized in Table 2. It should be mentioned that in the state-of-the-art algorithm, only one data point is used for evaluation and only one significant number is reported. In this paper we use three data points for evaluation and report four significant numbers to reduce uncertainty in the results.

| Algorithms | Min Validation Error | Max Validation Error | Avg. Validation Error | Avg. Test Error |
|---|---|---|---|---|
| Direct Curve Fitting | 0.1761% | 0.7523% | 0.6331% | 0.5264% |
| Nearest Neighbor | 0.2250% | 0.6753% | 0.4276% | 0.3824% |
| K-Nearest Neighbor (K=3) | 0.1762% | 0.6593% | 0.3913% | 0.4555% |
| Clustering-6 | 0.1176% | 1.8678% | 0.8139% | 0.7335% |
| Markov Chain Monte Carlo | 0.1466% | 0.7043% | 0.3853% | 0.3825% |
| Sliding Window | 0.3218% | 0.8258% | 0.5953% | 0.5967% |
| Comparison: State-of-the-art [5] | 0.1% | - | - | - |

TABLE 2: Results of Experiments (Only the min validation error is provided in the state-of-the-art reference)

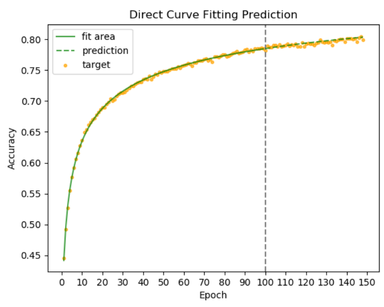

Fig. 4: Plot showing the results of direct curve fitting prediction.

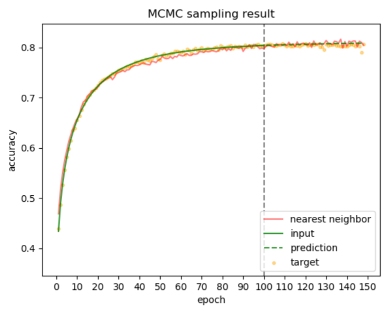

Fig. 6: Plot showing the results of MCMC method.

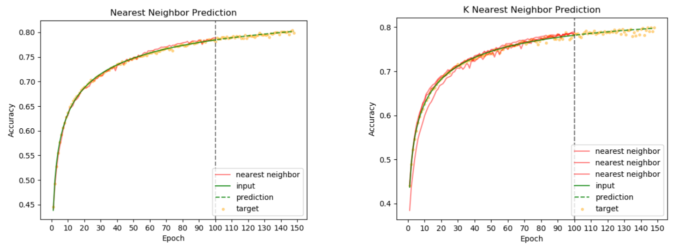

Fig. 5: Plots showing the results of Nearest Neighbor prediction (left) and K Nearest Neighbor prediction when K=3 (right).

## 5.1 Direct Curve Fitting

As a baseline algorithm in experiments, direct curve fitting naively minimizes the fitting error in the first 100 epochs and tries to predict the epochs from 100 to 150. The fitting algorithm can not foresee shapes of the whole curves and thus models are partially fitted. In addition, a curve shape capable of fitting the first 100 epochs of data isn't necessarily the best shape for predicting the later 50 epochs of data well. Unavoidably, there are gaps between predictions and targets as shown in Fig 4.

## 5.2 Nearest and K-nearest Neighbor

The experiment results are shown in Fig. 5. In our experiments, we tried two different ways of finding the best curve shape for the nearest neighbor of our input. One way is to fit the entire neighbor and use the curve that has the least absolute deviation. Another way is to use the curve that best predicts the final accuracy using the first 100 epochs of data in that nearest neighbor. The first method tries to find the best form of the generative model while the second one tries to find the best predictor. The results show that the first method marginally outperforms the second method, and are both capable of achieving approximately the same level of prediction accuracy.

## 5.3 Clustering

Since the clustering method can be regarded as a trade-off between curve matching performance and data storage requirement, different numbers of cluster centers have been tested and we decide that 6 cluster centers would be a reasonable trade-off, in which case only half of the training data needs to be saved and prediction performance is still fairly good.

## 5.4 Markov Chain Monte Carlo

Although first-ordered Markov assumption seems to be not enough for the curve-fitting problem, Markov Chain Monte Carlo method still benefits from taking the advantages of both parameterized fitting model and time-series analysis and provides better parameter set for model-based prediction. As shown in Fig. 6, time-series sampling helps the model fit the future accuracy better based only on the first 100 epochs. It can be expected that with more samples and more complicated Markov assumptions (second-ordered or higher-ordered Markov assumptions), the MCMC method is likely to perform even better.
One significant drawback of the MCMC sampling method is the time complexity. Due to sampling rejection, some samples are wasted, which makes MCMC less efficient and compute-demanding. It takes MCMC about 45 minutes while it only takes several seconds for nearest or K-nearest neighbor, or clustering-based algorithms.

## 5.5 Sliding Window Prediction

As shown in Fig. 7, the prediction performance of the sliding window is acceptable, but is worse than other algorithms

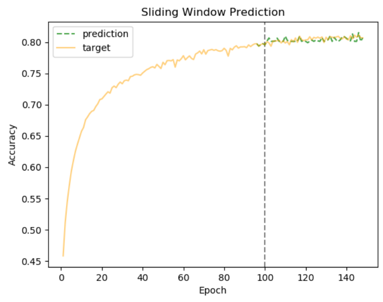

Fig. 7: Plot showing the results of sliding window prediction using neural networks.

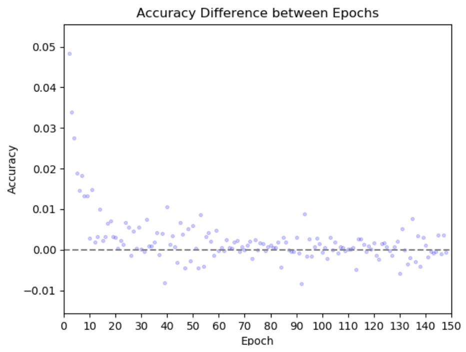

Fig. 8: Plot showing the accuracy difference between consecutive epochs.

and is below our initial expectation. The idea of sliding window prediction is to use the neural network to map from the input accuracy frame to the corresponding gradient of the curve so that it can catch the change rate of the curve given the time-series data. Thus, the gradient is very sensitive to random noises. As shown in Fig. 8, the different between consecutive validation data points is potentially too noisy for the neural network to handle. As a result, the learning performance of the neural network is lower than expected.

## 6 Future Work

Each validation curve data in our dataset took 2-6 hours to generate. Due to the limited time provided, we are not able to obtain a larger number of data and cover more advanced architectures. In the future, we can test our methods with more comprehensive dataset and examine whether our results still hold. Additionally, we can experiment with ways such as 1-D convolution/averaging that reduces the noise within the data itself and see if that can help our models to capture the trend of these curves. We can also build a automatic parameter searching system based on our models and evaluate their performance on such a task.

## 7 Conclusion

In conclusion, to predict the accuracy curves of neural networks, we start with the most intuitive model-based fitting method and then apply several advanced algorithms based on the basic idea of curve-fitting and time-series analysis. Among these different methods, we observed that K-nearest neighbor and Markov Chain Monte Carlo achieves the best overall performance.

The results we obtain look promising as the average error between our prediction and the actual final validation accuracy is only 0.38% using our best models. These models can be potentially applied to automatic architecture and parameter searching tasks such as AutoML, and can significantly reduce the time and computation spent on waiting for the validation accuracy to converge.

## 8 Group Member Contribution

Yunfeng worked on data generation and normalization, and algorithms including direct curve fitting, (K-)nearest neighbor, and sliding window generation with neural network. Chengzhe worked on data analysis and algorithms including direct curve fitting, clustering curve fitting and Markov Chain Monte Carlo sampling. Hangyi worked on validating some of the ideas in the sliding window generation method. All group members worked on the deliverables.

## 9 Acknowledgement

We would like to thank all the teaching staff in CS 229 who have given us support over the entire quarter.

## 10 Codebase

The project codebase can be found at
https://github.com/rogerxcn/predicting_cnn_learning_curve

## References

[1] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[2] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2188395

[3] K. Eggensperger, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," 2013.

[4] P. Kolachina, N. Cancedda, M. Dymetman, and S. Venkatapathy, "Prediction of learning curves in machine translation," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Jeju Island, Korea: Association for Computational Linguistics, Jul. 2012, pp. 22–30. [Online]. Available: https://www.aclweb.org/anthology/P12-1003

[5] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, pp. 3460–3468. [Online]. Available: http://dl.acm.org/citation.cfm?id=2832581.2832731

[6] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *ICLR*, 2017.

[7] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to mcmc for machine learning," *Machine learning*, vol. 50, no. 1-2, pp. 5–43, 2003.

[8] S. Singh, M. Wick, and A. McCallum, "Monte carlo mcmc: efficient inference by approximate sampling," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 1104–1113.

[9] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in python using PyMC3," *PeerJ Computer Science*, vol. 2, p. e55, apr 2016. [Online]. Available: https://doi.org/10.7717/peerj-cs.55

[10] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The american statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[11] I. Yildirim, "Bayesian inference: Metropolis-hastings sampling," *Dept. of Brain and Cognitive Sciences, Univ. of Rochester, Rochester, NY*, 2012.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.